AN AUTONOMOUS ROBOT CAPABLE OF SCALING A SELF-BUILT STRUCTURE

By

Jack Kolb

A capstone project submitted for
Graduation with University Honors

June 4, 2020

University Honors
University of California, Riverside

APPROVED

_____
Dr.Campbell Dinsmore
Department of Mechanicall Engineering

_____
Dr. Richard Cardullo, Howard H Hays Jr. Chair, University Honors

Abstract

Falls from scaffolding and ladders make up 34% of construction site injuries and deaths. To reduce human involvement on those structures, we developed a detailed design package and virtual prototype of a robot capable of autonomously constructing, ascending, and descending a structure. The robot consists of four main systems: a mechanical system, an electrical system, a software system, and the structure. The robot iteratively constructs a cubic structure around itself by elevating the structure and positioning the next block underneath the elevated structure. The robot is then able to climb the structure from the inside. Three electromechanical mechanisms are used to accomplish this: a lifting and climbing mechanism to elevate and climb the structure, a cube latch mechanism to pull in the next cube, and a rotating plate mechanism to allow the robot to perch on top of the cubes while climbing. The robot is able to construct and climb the cube structure to at least three feet. To verify the design, a finite element analysis was conducted on each externally-loaded component to determine their maximum deflections, maximum stresses, and factors of safety. These analyses showed that all components of the robot and structure can support their worst-case anticipated loads. The robot was also exported into the Gazebo 3D simulation environment, where the software package was tested on a virtual prototype to verify that the autonomous systems functioned correctly. Three main recommendations are to improve the cube structure design so that the maximum structure height can be increased, to research alternative commercial off-the-shelf components to lessen the manufacturing and operational limitations, and to research additional uses for this technology at construction sites. In conclusion, the proposed robot design successfully meets all design requirements, and future work could further develop the design for commercial use.

Executive Summary

Falls from scaffolding and ladders make up 34% of construction site injuries and deaths. To reduce human involvement on those structures, the engineering team developed a detailed design package and virtual prototype of a robot capable of autonomously constructing, ascending, and descending a structure. The robot consists of four main systems: a mechanical system, an electrical system, a software system, and the structure. The robot iteratively constructs a cubic structure around itself by elevating the structure and positioning the next block underneath the elevated structure. The robot is then able to climb the structure from the inside. Three electromechanical mechanisms are used to accomplish this: a lifting and climbing mechanism to elevate and climb the structure, a cube latch mechanism to pull in the next cube, and a rotating plate mechanism to allow the robot to perch on top of the cubes while climbing. The robot is able to construct and climb the cube structure to at least three feet. To verify the design, a finite element analysis was conducted on each externally-loaded component to determine their maximum deflections, maximum stresses, and factors of safety. These analyses showed that all components of the robot and structure can support their worst-case anticipated loads. The robot was also exported into the Gazebo 3D simulation environment, where the software package was tested on a virtual prototype to verify that the autonomous systems functioned correctly. Three main recommendations are to improve the cube structure design so that the maximum structure height can be increased, to research alternative commercial off-the-shelf components to lessen the manufacturing and operational limitations, and to research additional uses for this technology at construction sites. In conclusion, the proposed robot design successfully meets all design requirements, and future work could further develop the design for commercial use.

Table of Contents

1 Introduction

1.1 Problem Statement

Regions around the world, such as Southern California, have historically been an experiment in what happens when a metropolis builds “out” rather than “up”. Coming up against the realistic limits of urban sprawl, there is renewed interest in building taller structures that can support higher density populations. During construction, however, this approach to building exposes workers to dangerous heights. To minimize or even eliminate this danger, robots can be innovated to build the initial framing or scaffolding used for the structure. For this project, the goal is to design a robot that can efficiently and reliably build, ascend, and descend a free-standing scaffolding structure. Additionally, the robot must assemble the structure with no human intervention once the assembly process has started.

1.2 Problem Identification

As more people move into cities, buildings must adapt to accommodate the increasing population. To house more people, taller buildings are being built while construction sites must devise a method to safely transport workers and materials to higher elevations. The most common method construction sites use to work at heights beyond the worker’s reach are ladders and scaffolding. The Code of Federal Regulations (CFR) lists a set of safety requirements that the Occupational Safety and Health Administration (OSHA) mandates for ladders and scaffolding. OSHA has a variety of regulations for manufactured ladders brought onto the site, as well as for job-made wooden ladders made to scale between stories of a building [1]. OSHA also has limits on the loads carried by scaffolding and scaffolding construction methods in order to create and maintain a safe and stable structure for workers to place materials and work upon [2]. Despite these regulations, there are still thousands of construction site accidents caused by falls from ladders or scaffolding. These accidents usually result in injuries; however, there are many that result in fatalities. In 2009, OSHA reported that falls accounted for 34% of fatal occupational injuries. Of these injuries, 18% were from scaffolding or staging, and 16% were from ladders [3]. Therefore, around one-ninth of falls in a construction site are due to ladders and scaffolding.

To reduce the number of injuries and fatalities from scaffolding and ladders, ladder-climbing robots have been explored to transport materials up a ladder without human involvement. In the early 2000s, the Japanese were among the first to create robots and humanoids that could successfully ascend and descend ladders. The ASTERISK robot was developed in 2008 by Fuji and his team to be implemented into search-and-rescue missions [4]. This robot has six legs with claws at the end of each leg. To climb a ladder, the upper three legs would hold the top rung while the bottom three legs were used to stabilize the robot. This robot was only tested on a vertical ladder, and a control algorithm was used to prevent the robot and ladder from tipping as it climbs. This robot didn’t account for ladders at different angles, which would have been more realistic for search-and-rescue applications.

Another ladder-climbing robot was developed in 2008 by Yoneda and his group. This humanoid robot or “Multi-Locomotion Robot (MLR)” aimed to replicate animal-like instincts such as climbing ladders and biped and quadruped walking [5]. To climb the ladder, the research team proposed that the robot could maintain good posture to balance its momentum around the “Axis

of Yaw". Control algorithms and a recovery motion model were used to mitigate error and tipping as it climbs the ladder [5].

1.3 Motivation

Globally, more than 50% of humans live in cities [6]. This statistic is ever increasing as well. In 1885, the Home Insurance Building was constructed in Chicago, and set the pathway for many other American cities to follow in the creation of vertically integrated buildings [7]. As land becomes scarce to develop upon in the 21st century, multi-story buildings are considered to be the quintessential solution to vertically scale construction projects for a given population. But, the construction of these buildings come at a cost—the cost of construction workers' lives. This leads into our motivation of mitigating the percentage of falls in construction sites due to ladders and scaffolding. The goal of this project is to design a robot that can build, ascend, and descend a structure autonomously and without human intervention. Introducing this structure-building robot into construction sites will not only assist workers by transporting materials to higher elevations, but also reduce injury and fatality rates from falls.

1.4 Presumed Results

The goal is to create a virtual prototype of a robot that is capable of autonomously building a scaffolding-like structure as well as ascending and descending the structure. The robot shall take initial user input from a software perspective, but shall remain autonomous thereafter, thus requiring zero human involvement. The autonomy aspect is a crucial component of the project as it is the ultimate method in eliminating human fatalities at construction sites.

1.5 Report Summary

The following report encapsulates the entire engineering design process with regards to developing an autonomous structure-building robot. The problem of construction site fatalities will be analyzed to design and present an appropriate solution. Moreover, each component used in the design will be discussed and analyzed. Lastly, the analyses ensuring the solution meets the design requirements and functions properly will be presented in this report.

2 Problem Definition

2.1 Background Research

2.1.1 Construction Site Hazards

The Code of Federal Regulations contains a list of safety regulations that OSHA mandates for ladders and scaffolding at construction sites. These requirements mandate the maximum load ladders and scaffolding can hold as well as how to construct them [1]. Despite these regulations, thousands of accidents occur in the United States every year from falls. These accidents mainly result in injuries; however, fatalities are prevalent. OSHA reported in 2009 that there were up to 29,382 injuries and 86 fatalities from falls from scaffolding and ladders [8]. These statistics show

a great need to remove human interactions with scaffolding and ladders. A robot that autonomously builds a structure and is capable of ascending and descending the structure to carry materials could help mitigate human injuries and deaths at construction sites.

2.1.2 Scaffolding and Stable Structures

A primary goal of the project is to create a structure for the robot to be able to reliably build and climb. To develop a stable structure for the robot to climb, the most common types of ladders and scaffolding were analyzed. Ladders are used to elevate workers and materials slightly beyond their reach. They usually require support from both the wall and the ground, but cannot support heavy loads. The most common commercially available ladders are the single pole ladder, the extension ladder, and the step ladder [9]. The single pole ladder consists of two rails connected by rungs. It requires support from both the wall and the ground and is, therefore, not standalone. Extension ladders are single pole ladders with another set of rails and rungs to allow it to extend past its maximum height. Like single pole ladders, these are also not self-supporting as they require support from both a wall and the ground. Finally, a step ladder consists of two sets of rails and rungs attached in an A-shape [9]. This makes the step ladder standalone, so workers can climb and place materials within reach. Since all ladders have a maximum height and reach, ladders are not as versatile as scaffolding. Another way to reduce the number of falls from ladders and scaffolding is to choose the proper scaffolding for the application. The most common type of scaffolding is supported scaffolding [10]. This scaffolding is primarily made from wood or metal depending on the load the scaffold is predicted to hold. It consists of long horizontal and vertical members connected together to create a grid structure. Shorter members connect two grids together to create a 3D structure while diagonal members are added for stiffness [10]. This scaffolding is one of the strongest types of scaffolding as it is self-supporting and can hold heavy loads. Because it is easy to assemble, cheap, and standalone, it is one of the most common scaffoldings used in construction [10].

2.2 Specifications and Constraints

2.2.1 Functional Performance Requirements

FPR-1. The robot shall construct and climb a structure to at least three (3) feet high.

FPR-2. The robot shall construct and climb the structure within twenty (20) minutes.

FPR-3. The robot shall construct and climb to a user-input height within (±1) foot accuracy.

FPR-4. The robot shall be remotely shut down by the user and turn off within one (1) minute.

FPR-5. The robot shall be remotely signaled by the user to descend the structure, and descend within ten (10) minutes.

FPR-6. The robot shall construct, ascend, and descend the structure without human intervention.

FPR-7. The robot can use structure components continuously placed near the structure within (±1) inch of a target position.

2.2.2 Physical Requirements

PR-1. The robot shall contain mechanisms and processes allowing it to construct and ascend the structure.

PR-2. The built structure must be able to support the robot's weight during and after its ascent and descent.

PR-3. The robot must be within a 2ft by 2ft by 2ft size limit.

PR-4. The robot must not weigh more than 30lbs.

PR-5. The robot must be powered by either a NEMA 5-15 120V/15A power supply or onboard batteries.

2.2.3 Environment Requirements

ER-1. The robot must be able to operate in temperatures ranging from 32◦F to 100◦F.

ER-2. The robot must demonstrate a particulate and water resistance of IP50 [11].

ER-3. The robot must be able to operate on a flat hard-surface floor.

2.2.4 Life-cycle Requirements

LCR-1. The robot must construct, ascend, and descend the structure at least ten (10) times before mechanical failure.

LCR-2. Individual electrical components must be salvageable and the mechanical frame reusable or recyclable.

2.2.5 Human Factors

HF-1. A human shall be able to position the robot to a starting orientation.

HF-2. A human shall be able to connect the robot to a NEMA-compatible wall outlet.

HF-3. A human shall be able to position the structural elements around the robot if necessary

HF-4. A human shall be able to send commands to the robot using a remote interface.

3 Design Solution

3.1 Overview of the Final Design

The final design of the structure-building robot is divided into four main systems: (1) mechanical, (2) electrical, (3) software, and (4) structure. Starting from the top, the mechanical system is comprised of three main components critical in the construction and ascension of the structure. As depicted in **Figure 1**, these components include the lifting and climbing mechanism, cube latch mechanism, and rotating plate mechanism.

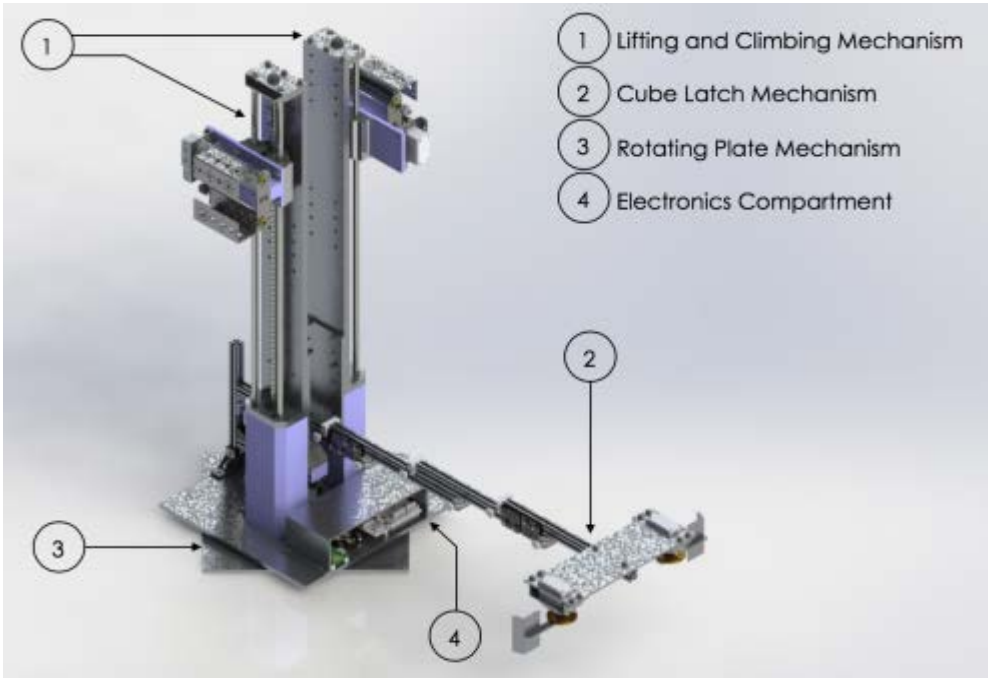


Figure 1: The final robot design featuring its main components.

The electrical system uses a microprocessor, a microcontroller, electromechanical actuators, and auxiliary components to drive the physical robot. The automation and user interface are managed by the microprocessor with actuation commands sent to the microcontroller for relaying to control the actuators. The Robot Operating System (ROS) framework is implemented to communicate with these two devices, which is where the software system comes in. As for the structure, it is built upon modified cube elements, each with one open face.

3.1.1 Building the Structure

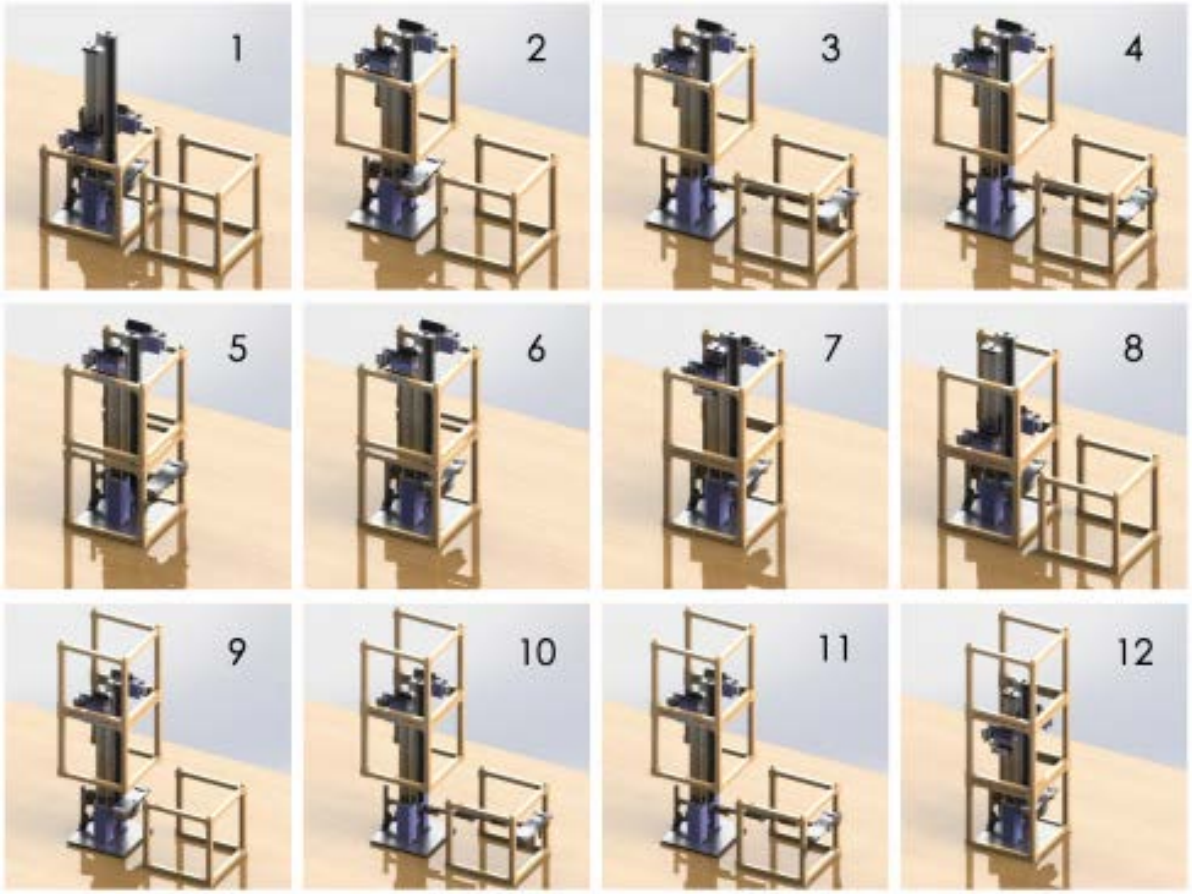


Figure 2: Step-by-step process of building the structure.

As mentioned in the Problem Definition (**See Section 2.2.1 Functional Performance Requirements**), the intent of the final robot design is to be scaled down to build a 3-foot high cube structure. The following steps are described in correspondence with the numbering in **Figure 2**:

1. The robot begins in its initialization phase. One cube element is already hooked onto the bottom hooks of the lifting and climbing mechanism. A second cube element is positioned on the robot's outer perimeter.
2. The linear actuator traverses upward, lifting the first cube to its maximum height with the dual hook assembly. There should be enough clearance underneath this cube for a second cube to be placed.
3. The linear slide assembly extends outward to its maximum distance.
4. The servo-controlled hooks of the cube latch assembly rotate at an angle of 180◦ and grab onto the vertical segments at the rear end of the second cube.
5. The linear slide assembly retracts and reels in the second cube until it is directly below the first cube.

6. The hooks of the cube latch assembly return back to their “closed” position, and the linear slide assembly retracts all the way in.
7. The linear actuator slowly lowers the first cube to complete a two-cube stack.
8. The linear actuator lowers even further, so the dual hook assembly can clamp onto the horizontal segments of the top face of the second cube. A third cube is introduced on the robot’s outer perimeter.
9. The linear actuator traverses upward, lifting the two-cube stack.
10. The linear slide assembly extends outward to its maximum distance.
11. The servo-controlled hooks of the cube latch assembly rotate at an angle of 180◦ and grab onto the vertical segments at the rear end of the third cube.
12. The linear slide assembly retracts and reels in the third cube until it is directly below the two-cube stack. The linear actuator slowly lowers to successfully build the structure with three cube elements.

3.1.2 Ascending and Descending the Structure

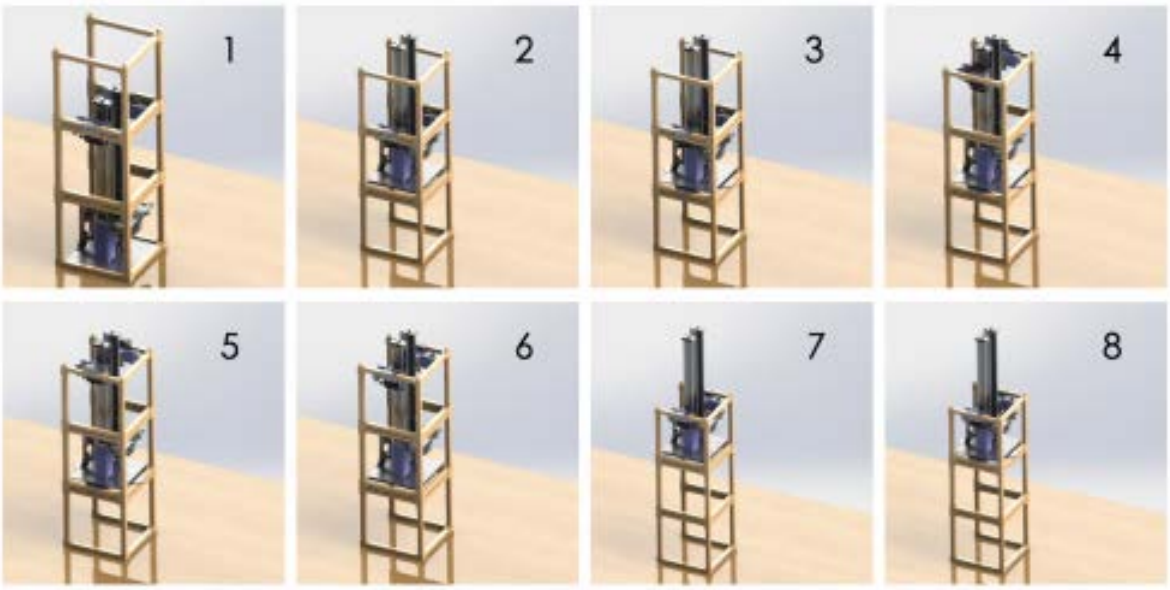


Figure 2: Step-by-step process of building the structure.

After the structure is built, the robot can proceed to climbing up it. As shown in Figure 3, the following steps demonstrate how the robot utilizes the lifting and climbing mechanism as well as the rotating plate mechanism to perform this task:

1. At the beginning of ascension, the top hooks of the lifting and climbing mechanism must grasp onto the horizontal segments of the bottom face of the first cube. By convention, the first, second, and third cube are the top, middle, and bottom cube, respectively.
2. Since the top hooks are secured onto the cube element, running the linear actuator would lift the entire robot upwards. The rotating plate should be above the horizontal segments of the bottom face of the second cube.

3. The servo rotates the rotating plate at an angle of 45◦ and locks the position of the robot at the second level.
4. The top hooks detach from the cube and travel on the linear actuator to its maximum height.
5. The top hooks now grasp onto the horizontal segments of the top face of the first cube
6. The servo rotates the rotating plate back an angle of 45°, so the rotating plate and base plate are aligned.
7. The linear actuator runs and lifts the entire robot upwards. The rotating plate should be above the horizontal segments of the bottom face of the first cube.
8. The servo rotates the rotating plate at an angle of 45° and locks the position of the robot. The robot successfully ascends up the structure.

   It is important to note that to descend down the structure, the robot must follow the steps above, but in reverse order. As noted by the step-by-step processes, the final design follows a systematic bottom-up approach that has a rhythm in building, ascending, and descending the structure.

3.2 Mechanical Systems

3.2.1 Base Plate

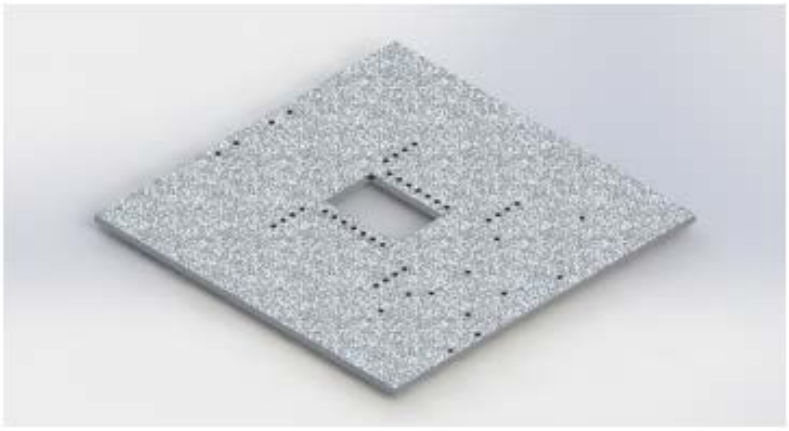


Figure 4: Isometric view of the base plate

The base plate is the foundation that supports all the primary mechanisms on the robot. As shown in **Figure 4**, the base plate is shaped as a square with side lengths of 10in and a thickness of 5mm (see **Figure B3.1** in Appendix B.3). Aluminum 6061 was chosen as the material because it not only has strong mechanical properties, but also is cheap for general-purpose use; this particular metal will be commonly used for several other parts in the final robot design [12].

The rectangular hole in the middle of the base plate has dimensions of 60mm by 39mm, which will be cut with a Dremel. A servo assembly is mounted on top of the base plate and controls the rotation of the rotating plate (see **Figure C3.1** in Appendix C.3). Since the servo horn is the pivot

point, the rectangular hole must be off-centered to position it at the center of the base plate accordingly. To complete the manufacturing of the base plate, several screw holes must be drilled to attach all the primary mechanisms.

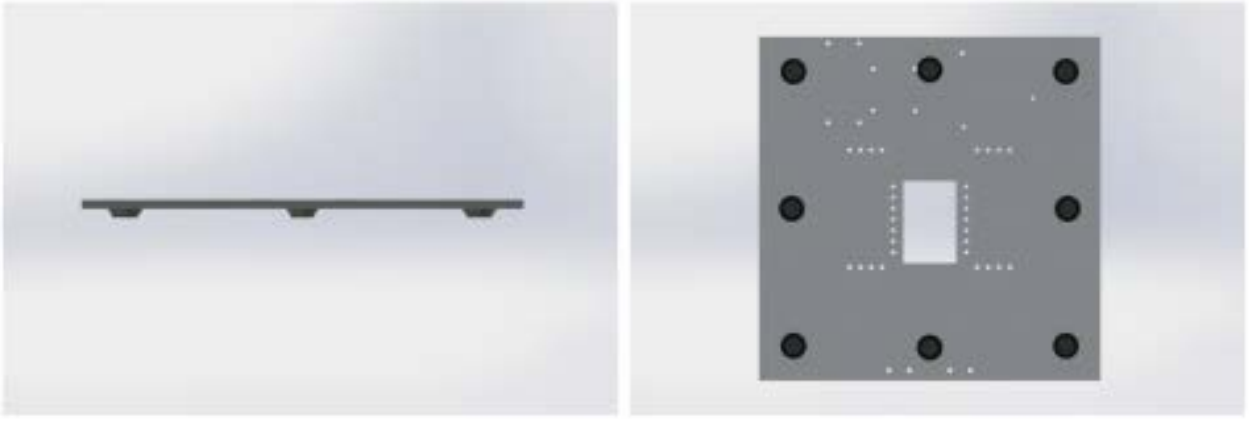


Figure 5: (Left) front view and (right) bottom view of the base plate

Eight rubber stoppers are to be attached onto the bottom of the base plate with an aluminum-friendly adhesive such as cyanoacrylate (i.e. super glue) [13]. The rubber stoppers each have a base diameter of 0.75in that tapers to an end diameter of 0.5in. Because the weight of the robot is concentrated on the servo horn connected to the rotating plate, these rubber stoppers act as a safety measure to reduce the stress on the servo horn. In other words, these rubber stoppers redistribute the robot's weight more evenly when they make contact with the rotating plate. It is important to note that the rubber stoppers along the cardinal axes are not redundant, but are necessary for when the rotating plate is angled at 45◦ during ascension and descension.

3.2.2 Lifting and Climbing Mechanism

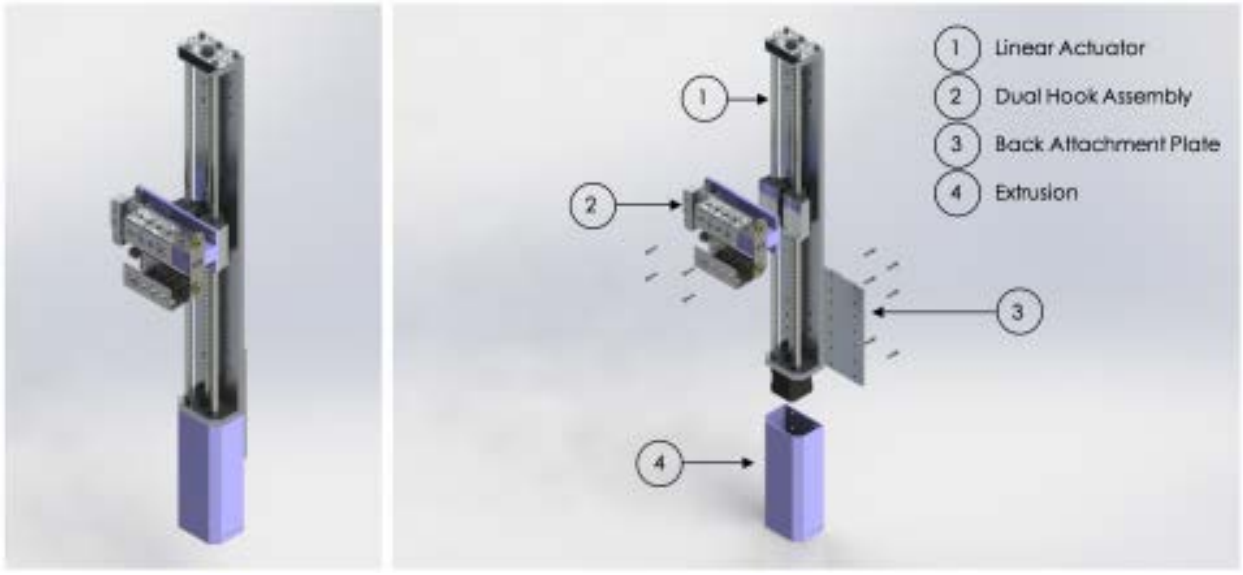


Figure 6: (Left) isometric view and (right) exploded view of the lifting and climbing mechanism

The lifting and climbing mechanism features a dual hook assembly that travels vertically along a linear actuator. As shown in **Figure 6**, the extrusion at the bottom of the mechanism is 3D printed with acrylonitrile butadiene styrene (ABS) plastic and heightens the linear actuator by 165mm (see Figure B1.10 in Appendix B.1). Moreover, the extrusion encases and protects the stepper motor that runs the linear actuator.

To attach the extrusion and linear actuator together, a piece of Aluminum 6061 sheet metal with a thickness of 4mm is required (see Figure B1.8 in Appendix B.1). Its length and height are 70mm and 154mm, respectively. Screw holes are to be drilled into the attachment plate, so it can be fastened onto the linear actuator.

To complete the lifting and climbing mechanism, the dual hook assembly must be screwed onto the double bearing mounts that run along the rods of the linear actuator. Figure 7 depicts how two individual lifting and climbing mechanisms are attached onto the base plate with commercial off-the-shelf L-beams.

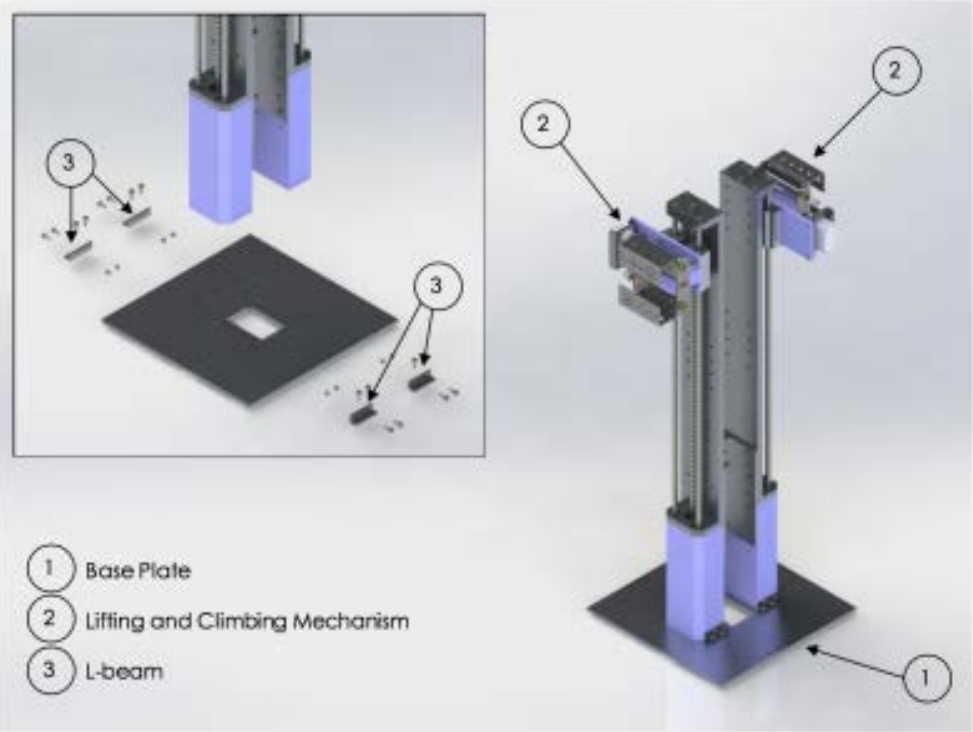


Figure 7: Isometric view of lifting and climbing mechanism on base plate with enlarged exploded view.

3.2.2.1 Linear Actuator

The function of the linear actuator is to translate the dual hook assembly in the vertical direction. Because there were no commercial off-the-shelf linear actuators that could accommodate a two-foot reach, a custom linear actuator was inspired by a contributor on GrabCAD and designed with several commercial off-the-shelf parts instead, which are listed in Figure 8 (see Figure C1.4 in Appendix C.1) [14]. The advantage of a custom linear actuator is that the length can be set to best fit the needs of the robot. The 500mm linear actuator enables the dual hook assembly to not only reach below the cube for lifting, but also raise above the cube for climbing.

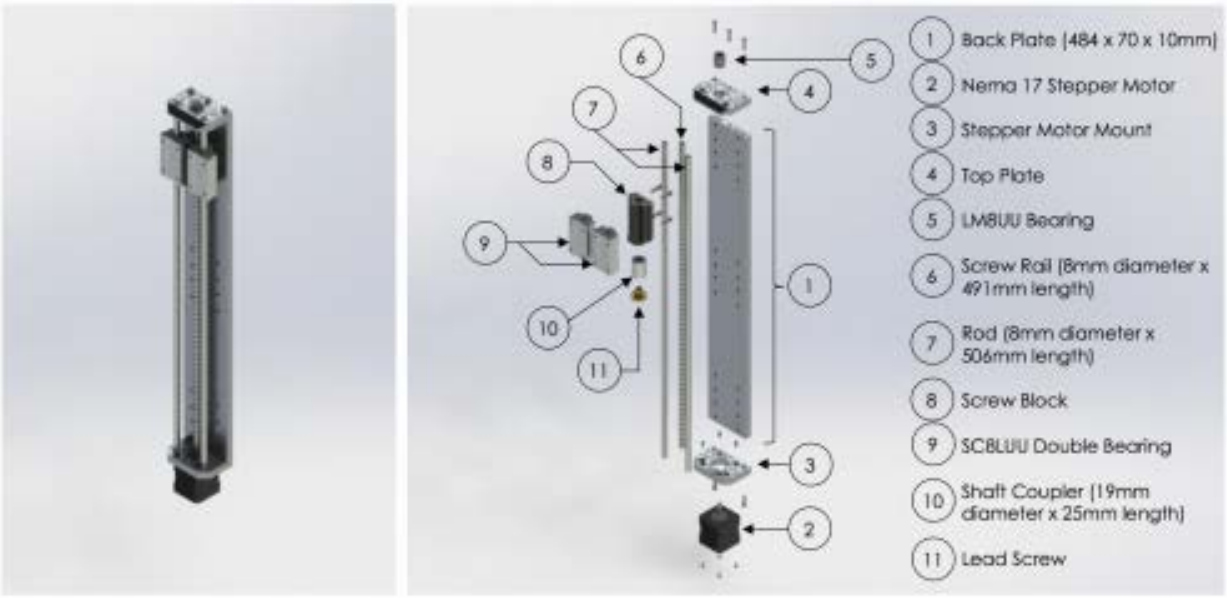


Figure 8: (Left) isometric view and (right) exploded view of the linear actuator

3.2.2.2 Dual Hook Assembly

Figure 8: (Left) isometric view and (right) exploded view of the linear actuator

Lifting the cube elements and climbing the cube structure both only require vertical translation, which is why they were combined into one mechanism. As shown in **Figure 10**, the dual hook assembly is composed of the 3D printed hook and servo mount, modified TETRIX parts, and a servo that controls the rotation of both hooks simultaneously (see **Figure C1.3** in Appendix C.1).

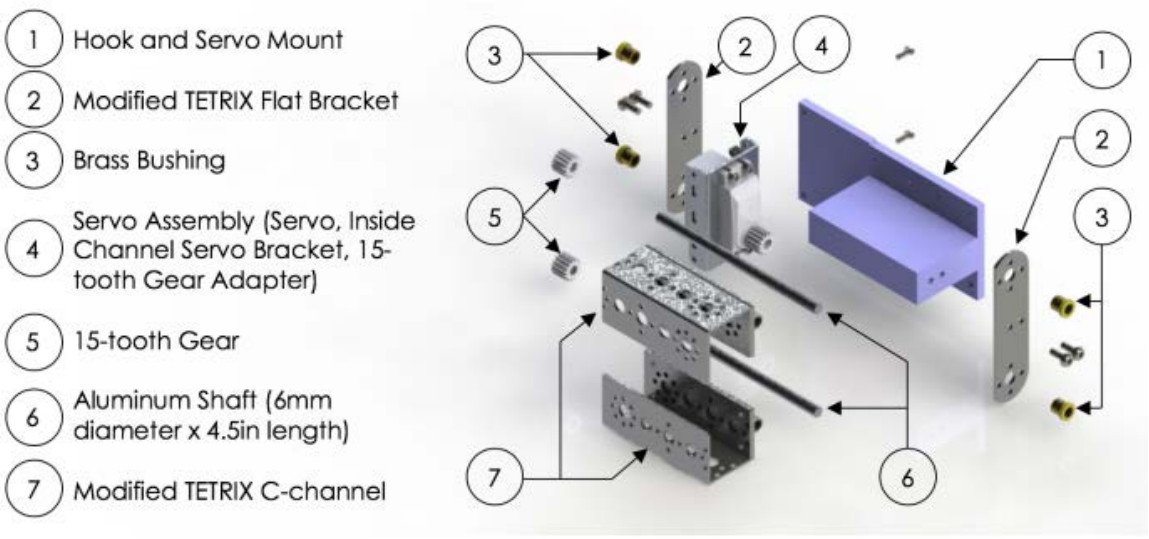


Figure 10: Exploded view of the dual hook assembly.

The *TETRIX* at brackets are trimmed on one edge to eliminate the interference with the cube-stacking (see **Figure B1.5** in Appendix B.1). Moreover, two holes are drilled in each at bracket, so they can be fastened onto the sides of the hook and servo mount. Brass bushings are placed on both ends of the at brackets; aluminum shafts are inserted through these brass bushings, in which the hooks can pivot. The lifting and climbing hooks use *TETRIX* C-channels that are riveted to a 4mm-thick piece of Aluminum 6061 sheet metal (see **Figure B1.7** in Appendix B.1). On the opposing side of the sheet metal, an aluminum rod must be spot welded on each end to solidify the pieces together (see **Figure B1.6** in Appendix B.1). The lifting and climbing hooks are secured between the two modified *TETRIX* at brackets. The two aluminum shafts each have 15-tooth gears that are aligned with one on the servo assembly. These three gears share a timing belt (not shown in Figures 9 and **Figures 10**) and, therefore, rotate together.

To visualize how the hooks function, three positions — lifting, closed, and climbing — are shown in **Figure 11**. As expected, the top and bottom hooks are used for climbing and lifting, respectively. When they are in their "closed" position, it can be perceived that the boundary box of the robot does not interfere with the cube element.

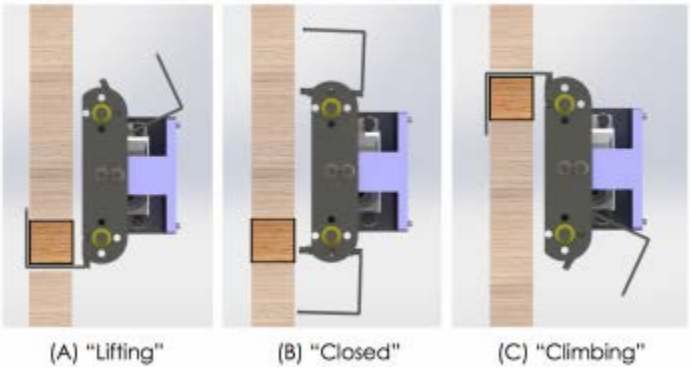


Figure 11: Three positions of the dual hook assembly

3.2.3 Cube Latch Mechanism

Figure 12: Isometric view of the cube latch mechanism when it is (left) retracted and (right) extended

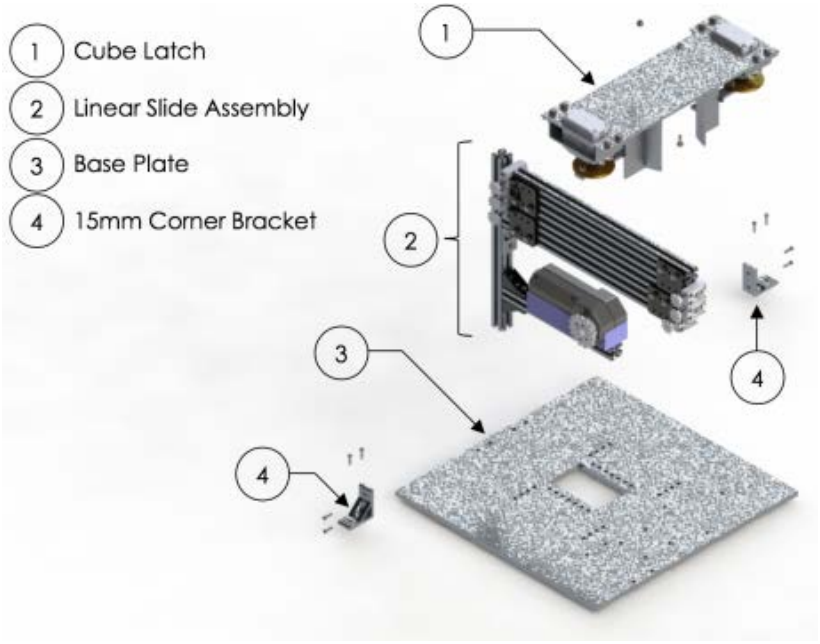


Figure 13: Exploded view of the cube latch mechanism

The cube latch mechanism features a cascading linear slide assembly with the cube latch secured on the free end. The linear slide assembly is mounted onto the edge of the base plate by two 15mm corner brackets. As alluded in **Figure 12**, the cube latch mechanism resembles a cantilever beam; however, the 15mm extrusion carrying the motor and motor mount is in fact supported by the servo assembly of the rotating plate mechanism. This decision was made to redistribute the weight of the cube latch and linear slide assembly as highlighted in **Figure 14**.

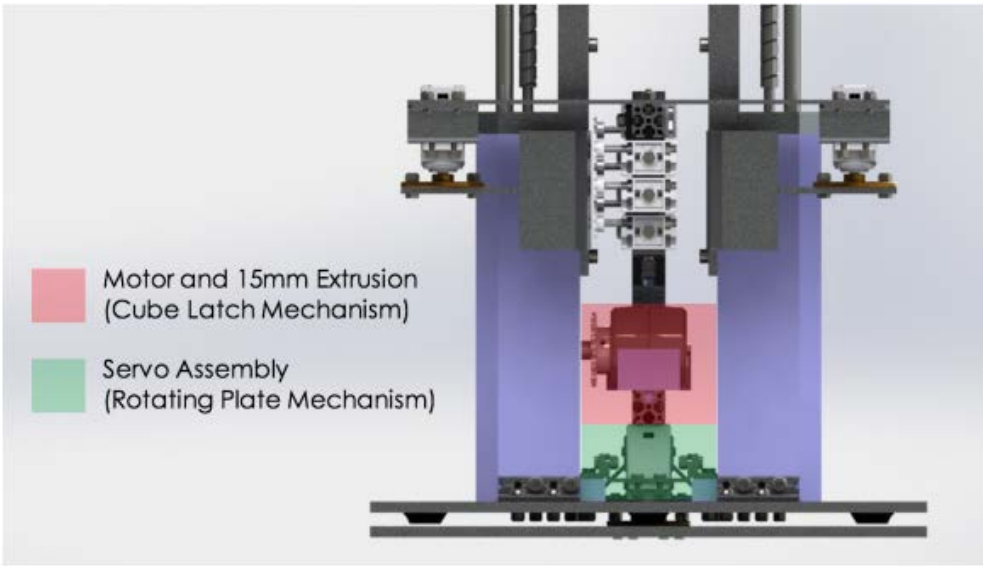


Figure 14: Exploded view of the cube latch mechanism

3.2.3.1 Linear Slides

Figure 15: (Left) isometric view and (right) exploded view of the linear slide assembly

The four-stage linear slide assembly was chosen to be implemented into the cube latch mechanism because it is compact and can extend a great distance. This linear slide assembly can be purchased commercially from *REV Robotics*, but several modifications are required to accommodate the robot's size and functional requirements [15] [16] [17]. First, the 15mm extrusions for the linear slide assembly must be shortened from 420mm to 218.2mm in length to be within the boundary box (see **Figure B2.2** in Appendix B.2). The commercial off-the-shelf linear slide assembly is intended to be used in a vertical orientation as it uses a string-and-pulley system for lifting. Gravity naturally lowers this system down. As a unidirectional pull is unable to retract the cascading linear slides in a horizontal orientation, the string-and-pulley system must be replaced with a chain-and-sprocket system to allow the DC motor to both extend and retract the linear slides (see **Figure C2.1** in Appendix C.2). Small 8-teeth sprockets are implemented to enable bidirectional movement for the cube latch mechanism.

To run the four-stage linear slide assembly, a DC motor is required. A custom 3D printed motor mount was designed to fit into the channel of the 15mm extrusion as well as align the sprocket on the motor to those on the linear slide assembly (see **Figure B2.5** in Appendix B.2). The motor mount will be made out of ABS plastic and have holes at the base, so it can be directly fastened onto the 15mm extrusion. To ensure that the DC motor will not sway, it will be secured onto the motor mount with a Velcro strap (see **Figure C2.2** in Appendix C.2) [18].

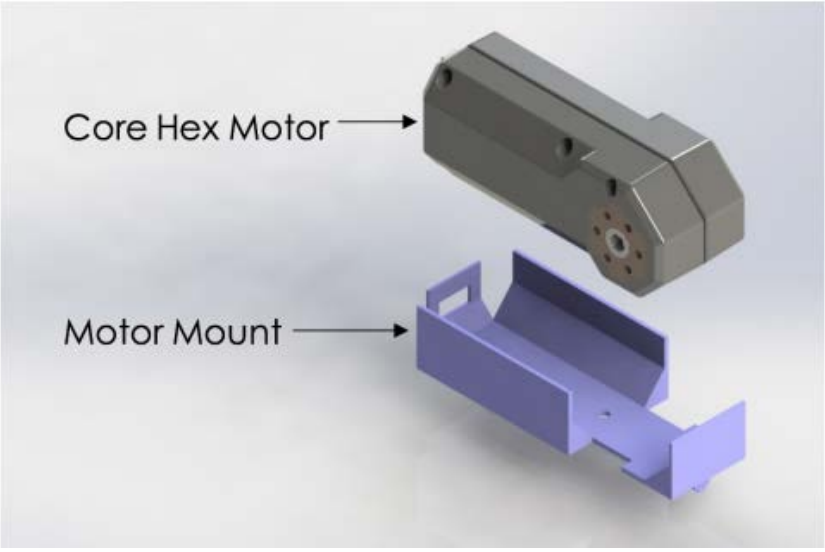


Figure 16: Exploded view of motor and motor mount.

3.2.3.2 Cube Latch Assembly

Figure 17: Isometric view of the cube latch assembly

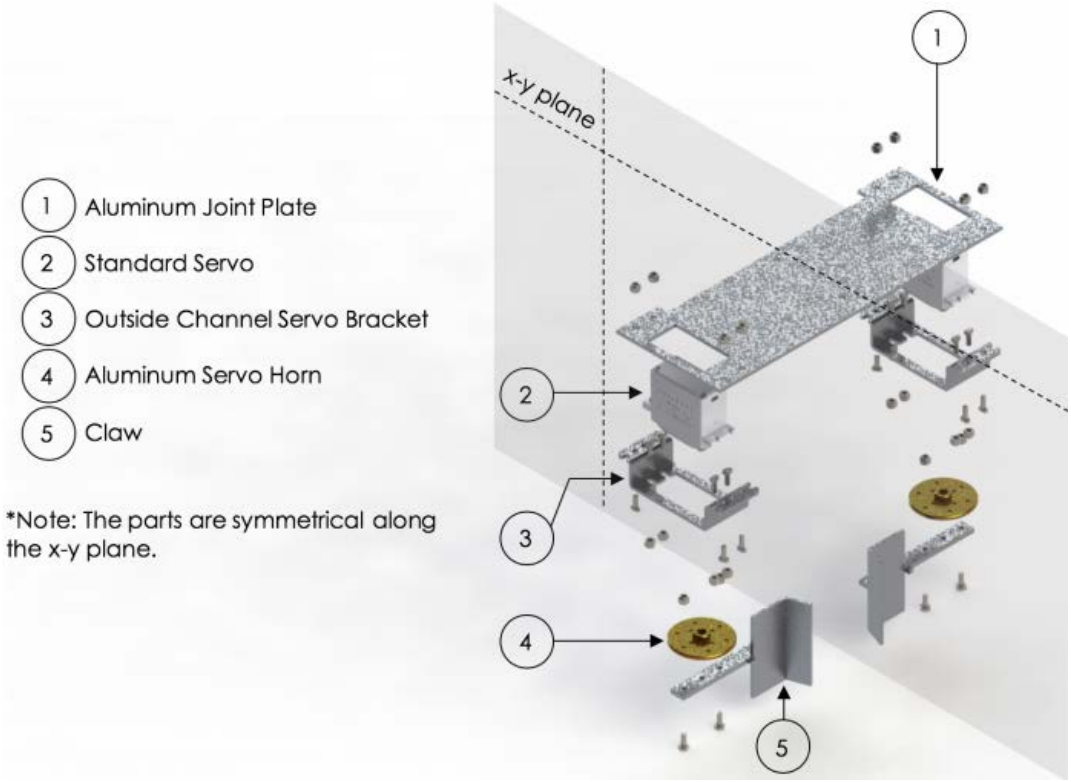


Figure 18: Exploded view of the cube latch assembly.

The cube latch assembly consists of a joint plate, two standard servos, and two claws that are limited to 180◦ rotation (see **Figure C2.5** in Appendix C.2). The joint plate is made out of Aluminum 6061 with a thickness, width, and length of 2mm, 217.9mm, and 80mm, respectively (see **Figure B2.9** in Appendix B.2). Two rectangular holes of 22mm by 42mm are required, so the servos can be inserted through. A notch on the top of the joint plate measuring 7.5mm by 153mm must be cut to prevent any interference with the lifting and climbing mechanism when the linear slide assembly is fully retracted. The rectangular holes and notch can be cut with a Dremel, but a CNC machine or stamp is recommended for higher precision. Additionally, several holes must be drilled to attach the servo mount onto the joint plate and the joint plate on the linear slide assembly.

Figure 19: Top view of the joint plate.

The claw consists of three components: a flat beam, bent piece of sheet metal, and tab. They are made out of Aluminum 6061 with a thickness of 3mm. The flat beam is cut to 78.94mm by 10mm, and three holes are drilled to attach the claw to the aluminum servo horn (see **Figure B2.7** in Appendix B.2). The hook begins with a flat piece of sheet metal with dimensions of 40.4mm by 53.8mm, then bent at an angle of 70◦ (see **Figure B2.6** in Appendix B.2). To manufacture the claw, the hook is riveted onto the flat beam. Thereafter, a small tab must be spot welded onto the edge of the flat beam; this tab acts as a stopper and limits the range of movement when the cube latch assembly grasps onto a cube element. The idea behind this claw design is to provide leeway for the cube latch mechanism to grab slightly misoriented cube elements (see **Figure C2.4** in Appendix C.2).

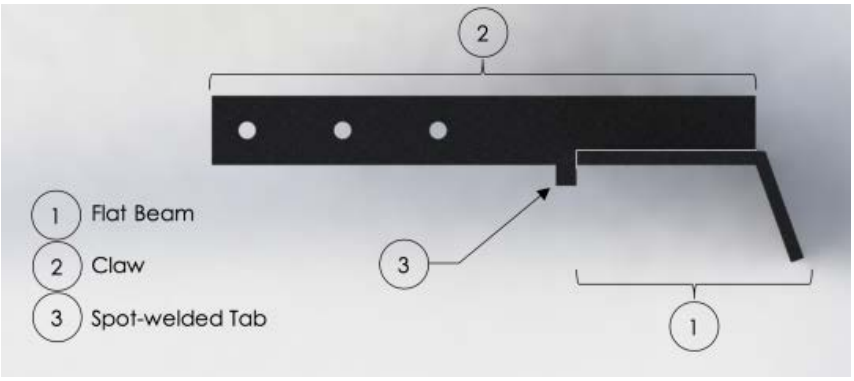


Figure 20: Top view of the claw.

3.2.4 Rotating Plate Mechanism

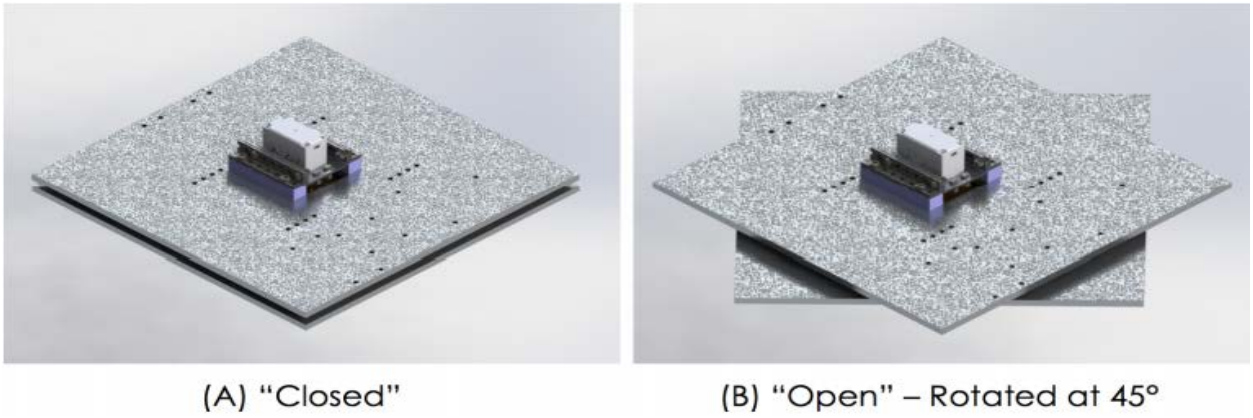


Figure 21: Isometric views of the rotating plate mechanism in two positions.

The function of the rotating plate mechanism is to perch the robot at each level as it ascends or descends the cube structure. The rotating plate is attached to the aluminum servo horn on the servo assembly. To close the gap between the base plate and rotating plate, two extrusions were designed to elevate the servo assembly (see **Figures B3.3** and **B3.4** in Appendix B.3). These extrusions are planned to be 3D printed with ABS plastic. Moreover, two L-beams are attached on top of the extrusions, in which the servo can be mounted (see **Figure B3.5** in Appendix B.3).

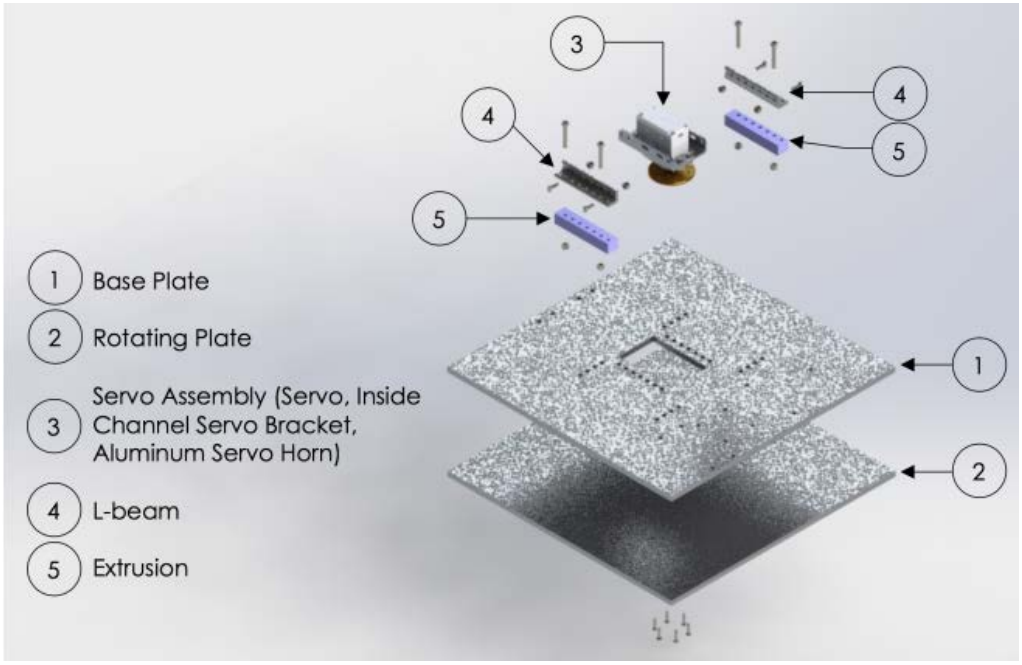


Figure 22: Exploded view of rotating plate mechanism.

3.2.4.1 Rotating Plate

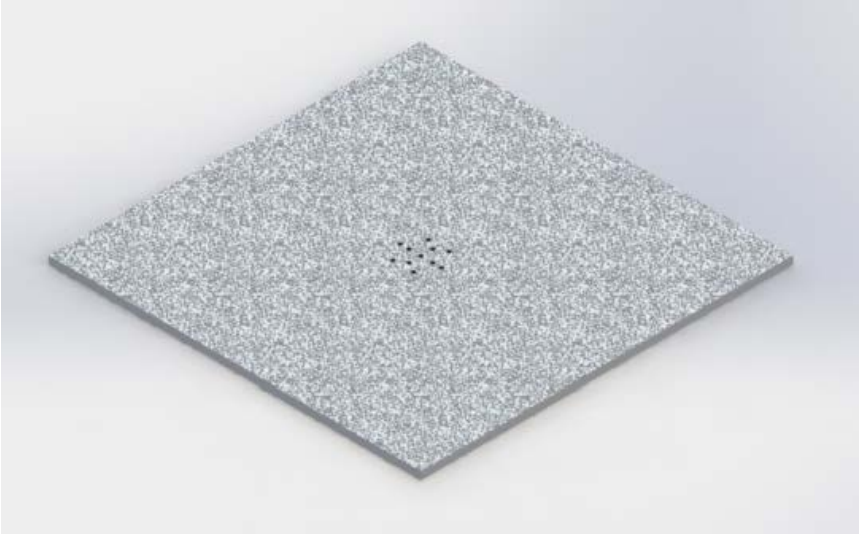


Figure 23: Isometric view of the rotating plate.

The rotating plate is made out of the same material and has the exact dimensions as the base plate. As shown in **Figure 23**, the holes drilled into the rotating plate must match the circular pattern on the aluminum servo horn (see **Figure B3.2** in Appendix B.3). When this manufacturing process is complete, it can be attached onto the servo assembly with six screws.

3.3 Electrical Systems

The electrical system entirely uses commercial off-the-shelf components. These components are connected either with off-the-shelf standard connectors or 14-gauge multi-strand wires. Components communicate either through wireless capabilities (Wi-Fi), direct serial communication, a binary digital signal, or a pulse width modulation (PWM) signal.

3.3.1 Overview of Electromechanical Systems

The detailed schematic of the electromechanical systems is shown in **Figure 24**. Green-shaded blocks are computers and blue-shaded ones are other components (i.e. actuators, power distribution board, etc.). Each arrow represents the communication between two components, and the components are organized by each main mechanism.

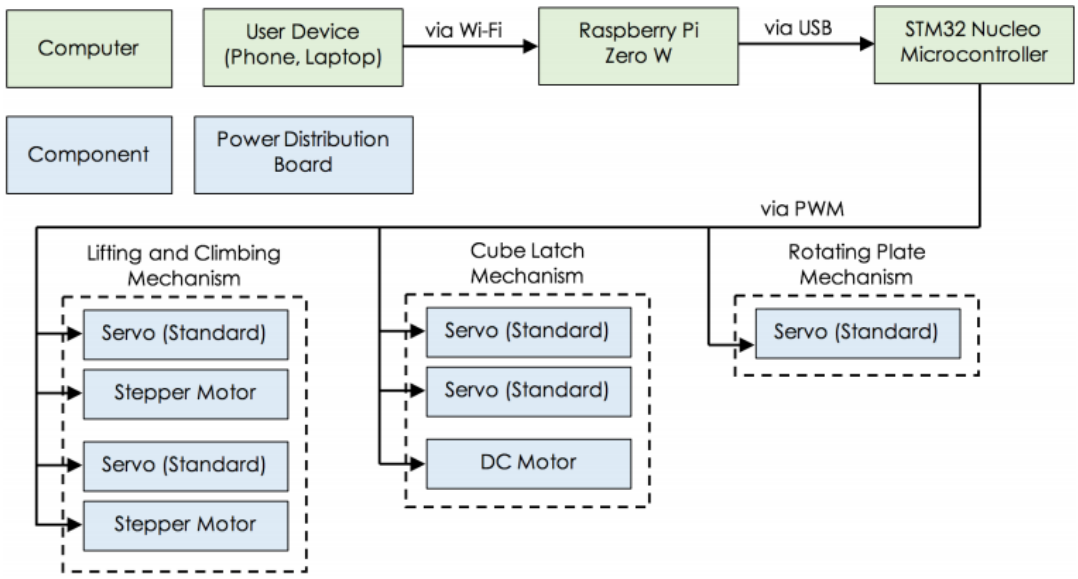


Figure 24: Electromechanical schematic.

Three separate computers are used for the robot: a microcontroller (STM32 Nucleo64), a microprocessor (Raspberry Pi Zero W), and a user device (cell phone or computer). The user device is supplied by the user and is external to the robot system. The microprocessor requires a 5V/1A DC power input and communicates with the microcontroller using a micro USB to mini USB wire connection. The microcontroller is also powered by the microprocessor through a USB connection.

Each of the five servos' PWM input connects directly to a PWM output terminal on the microcontroller. Each servo requires a 5V/1A DC power input.

The DC motor's power input line is connected to the emitter terminal of a metal-oxide-semiconductor field-effect transistor (MOSFET), with the collector terminal connected to a 10V/1A power input and the base terminal connected to a PWM output terminal on the microcontroller. This means a PWM signal from the microcontroller will affect the power input to the motor, allowing the microcontroller to be a control variable of the motor's speed. A wiring diagram of the MOSFET-to-DC motor connection is depicted in **Figure 25.**

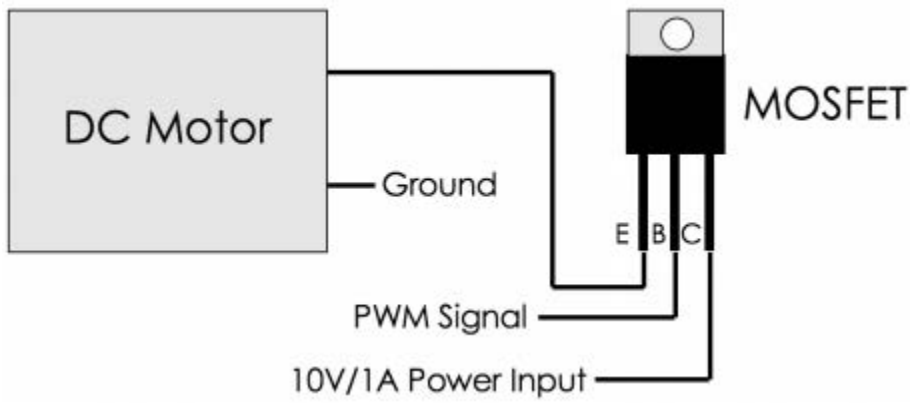


Figure 25: Electromechanical schematic.

The stepper motors each have two power input lines. Similar to the DC motor, each power input is connected to the emitter terminal of a MOSFET. The collector terminal is connected to a 10V/1A power input, and the base terminal is connected to a digital output terminal on the microcontroller. Instead of the MOSFET being used as a signal amplifier, this configuration means it will be used as a switch to allow the microcontroller to control the stepper motor's position and rotation speed. The MOSFET will be attached closely to the microcontroller, and the terminal soldering will be insulated with a heat shrink.

3.3.2 Power Distribution

A power distribution board is used to power all components of the robot. The power distribution board chosen was the MATEKSYS FCHUB-6S, a commercial off-the-shelf component that contains sufficient output terminals for the robot's computers and actuators [19]. The power distribution board's input terminals are connected to the DC output of a generic NEMA-compatible 12V/8A AC-to-DC power supply, and splits the power input into voltage-regulated outputs at several 5V and 10V terminals. These terminals are then connected to the power input lines of the actuators as per their voltage requirements. **Figure 26** demonstrates how the power distribution board connects to each actuator and the power supply. Several actuators share terminal ports due to the limited number of ports available on the chosen power distribution board; this is safe as the actuators chosen to share these ports either will have low current draws or will not be activated simultaneously.

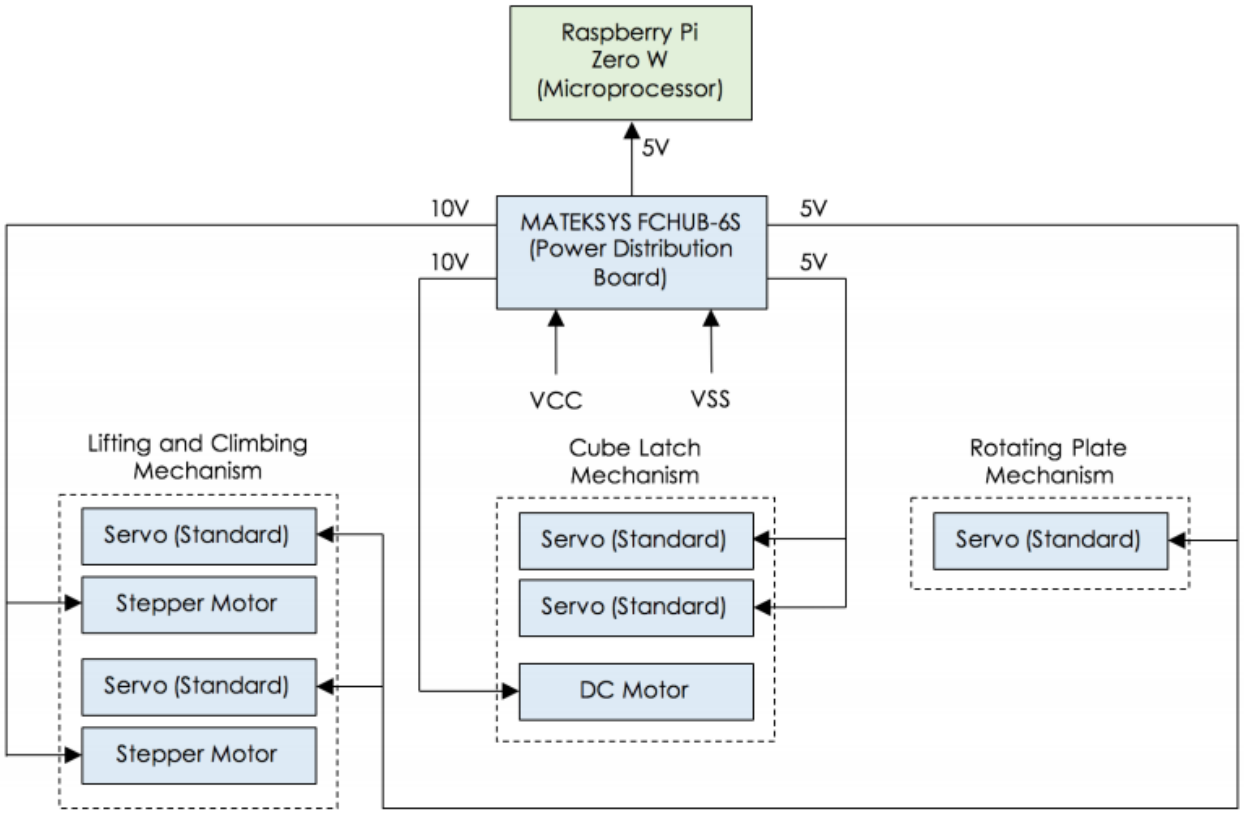


Figure 26: Wiring schematic for power distribution board

3.4 Software Systems

The software system has three core components: (1) a webserver for the user to interact with the robot, (2) a finite state machine for autonomous control, and (3) a relay to convert high-level desired actuator positions to actual actuator actuations. The microprocessor runs the webserver and the finite state machine, and the microcontroller runs the actuator relay. The two computers communicate using the Robot Operating System (ROS) framework. For the virtual prototype, the microcontroller code is replaced with custom plugins for the Gazebo simulation environment to control a virtual model of the robot. A flow chart of the overall software system is shown in **Figure 27**.

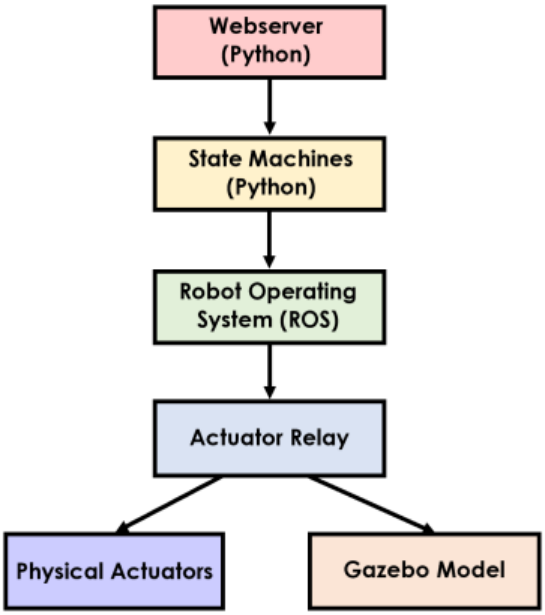


Figure 27: Data flow from user control (Webserver) to virtual prototype (Gazebo).

3.4.1 Webserver

The Raspberry Pi Zero W microprocessor was selected because it is cheap, contains sufficient memory and processing power, and has a Wi-Fi chip. The microprocessor uses its Wi-Fi chip to broadcast a Wi-Fi network, and also runs a webserver at the static IP address 192.168.1.1. This allows a user's cell phone or computer to connect to the robot; through the device's web browser, the user can start and stop the robot, specify the height of the structure, and provide other inputs. To control the robot, the user connects the user device to the microprocessor's Wi-Fi network, visits the defined IP address in the user device's browser, and interacts with the robot through the web interface. Commands to the robot are then sent using GET requests from the browser. The broadcasted Wi-Fi network is configured using the "dhcpcd", "dnsmasq", and "hostapd" Linux utilities. The webserver is written in the Python programming language and uses the Flask webserver library (see Appendix D.1) [20]. Alongside the webserver, the microprocessor runs the master ROS server and the finite state machines.

3.4.2 Robot Operating System (ROS)

Robot Operating System (ROS) is a software framework that facilitates communication across multiple devices and processes [21]. ROS works by having a central server (the "master" node) with which all other ROS-enabled processes connect to for their data exchange. ROS is also capable of facilitating communication between hardware via a USB serial connection. By using ROS, we avoid constructing our own serial communication framework between the microprocessor (Raspberry Pi Zero W) and the microcontroller (STM32 Nucleo64).

Communication in ROS is done by setting ROS variables called topics. Nodes—processes running on devices—can read and write to topics to get and set the values of these topics. For the structure-building robot, there is a topic for each actuator's desired position as well as additional topics for the states of the robot. The finite state machine sets the actuator topics, which are then read by the actuator relay to control the robot.

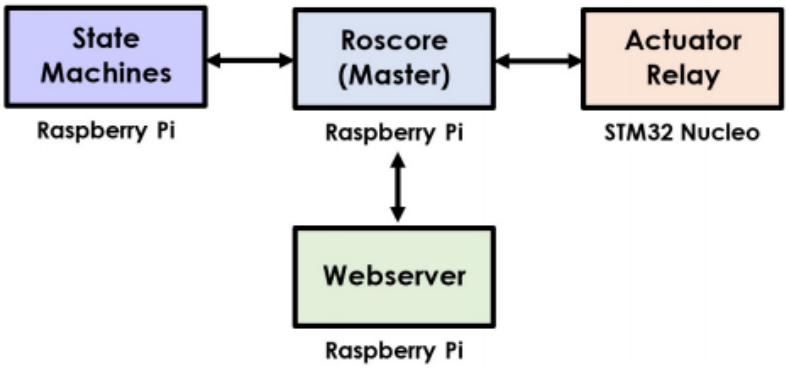


Figure 28: ROS communication network.

3.4.3 Finite State Machines

A finite state machine is a set of defined series' of actuations, interconnected by logic-driven transitions. The autonomous system contains finite state machines for the idle state, the construction state, the ascension state, the descension state, and the overall system. The finite state machines used in this system are written in the Python programming language and store information as ROS topics. **Figure 29** provides the finite state machine of the overall system. Keep in mind that each block is called a "state".

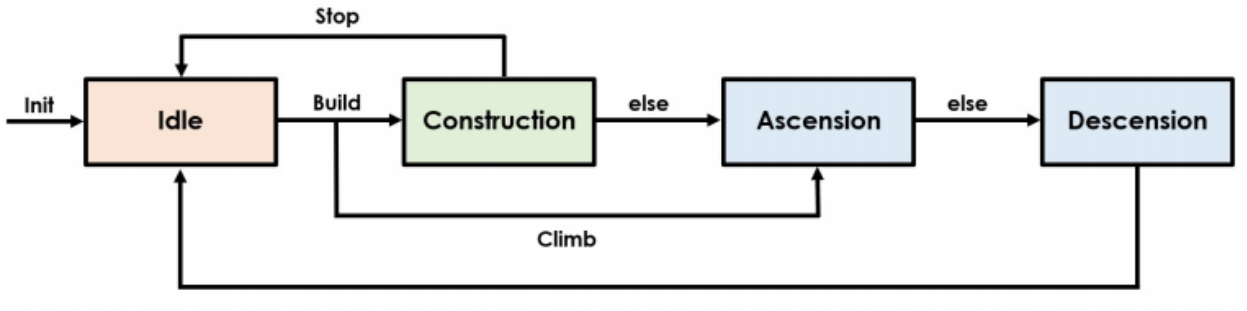


Figure 29: Overall state machine.

Initially, the state machine is in the "Idle" state until the robot is given either the "Build" or "Climb" commands by the user. If the "Build" command is given, the state machine transitions to the "Construction" state—a state machine that controls the actuator sequence to construct the structure. If the robot is told to stop during the construction phase, the robot returns back to the "Idle" state. Alternatively, if the construction is complete, then the robot will transition onto the "Ascension" state, which controls the actuator sequence to climb up the structure. After climbing the structure, the state machine transitions to the "Descension" state to climb down the structure, returning once more to the "Idle" state. A more complex state machine could be designed to allow the user more control over which level the robot ascends and descends to; however, this overall state machine satisfies the basic requirements of constructing, ascending, and descending the structure.

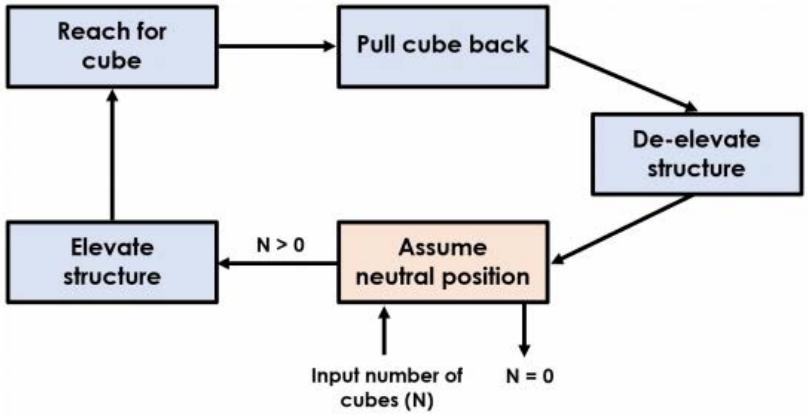


Figure 30: Construction state machine.

The construction state machine shown in **Figure 30** primarily consists of the series of actuations required to build the structure. While each state in the overall state machine from Figure

29 represents another state machine, each state in the construction state machine represents a defined series of actuations.

The state machine begins by assuming a neutral position. If the robot needs to build the structure with another cube element, it must go through the following steps: elevates the current structure, reaches for the next cube, pulls the cube back, de-elevates the current structure. This cycle repeats until there are no further blocks to construct. At this point, the construction state machine exits.

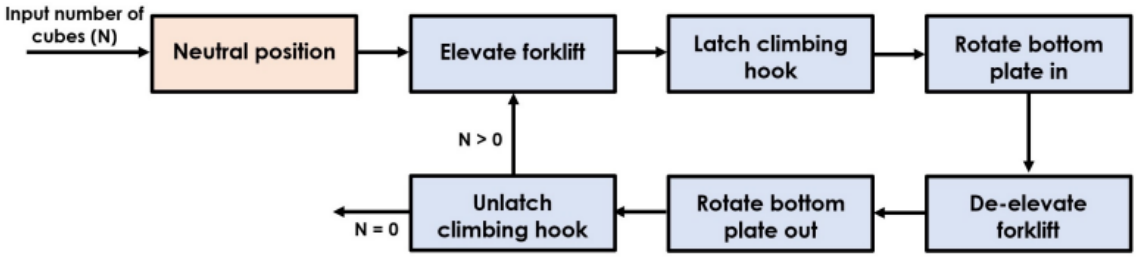


Figure 31: Construction state machine.

As with the construction state machine, the ascension state machine is a cyclical machine with states primarily consisting of defined series' of actuations as described in **Figure 31**. The robot begins by assuming a neutral position and transitions to climb the first cube. After each cycle is completed, if there are still a number of available stacked cubes for the robot to climb, the cycle repeats. Once the robot has ascended to the target level, the state machine exits.

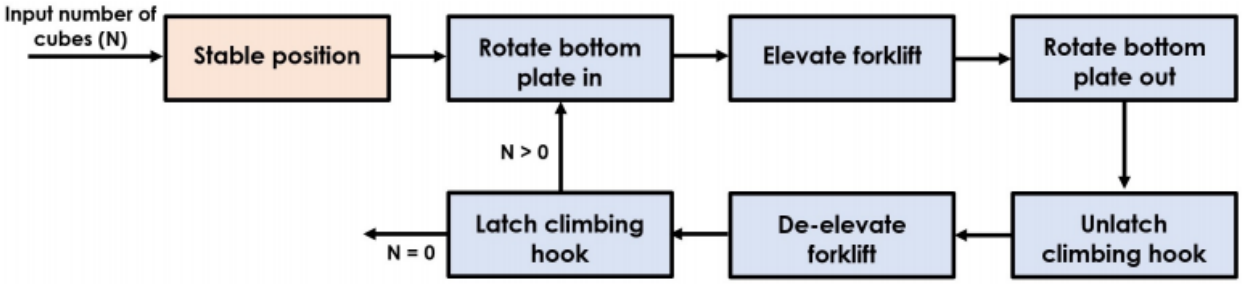


Figure 32: Construction state machine.

For the most part, the descension state machine is the reverse order of the ascension state machine. The state machine transitions through actuation sets to descend the structure one cube element at a time until the robot has reached to the floor level of the structure.

These state machines enable the robot to autonomously construct and scale the structure to variable heights. By using a state-based system as opposed to a single linear script, we are able to gain more flexibility with controlling how the robot moves. For example, a future expansion of this project may consider using the cube latch mechanism to collect an item once the robot has scaled the structure; the state machine system would account for that change by merely adding those state blocks wherever desired by the system.

3.4.4 Actuator Relay

When executing the state machines, ROS topics are set to move the robot at each state. These are high-level actuation parameters that need to be converted into raw signal outputs. For instance, the descension state machine shown in Figure 32 has the state “Rotate bottom plate ‘in’”. This is represented in the state machine by setting a variable for the rotating plate’s servo’s position to a specific angle θ. As the actual servo uses a PWM signal instead of a digital representation of θ, a conversion from θ to the equivalent PWM signal is necessary. Likewise, the stepper motor controlling the linear actuator require a conversion from the target height of the linear actuator to the number of stepper motor cycles. Furthermore, the DC motor for the cube latch mechanism requires a time-based conversion for its extension and retraction. These conversions and subsequent actuations are managed by the microcontroller. To send the digital value of an actuator’s desired position to the microcontroller, the microprocessor writes the desired value to a specific ROS topic; the microcontroller reads that ROS topic as joint positions in Gazebo.

The microcontroller converts the provided high-level actuator position to the actuator’s PWM value, cycle count, or duration. The microcontroller then sets its output pins accordingly to control the actuator and writes to a ROS topic when the actuation has completed. Due to the high manufacturing variability of low-cost actuators, the precise conversion will be tested and defined for each actuator. The microcontroller code is written in the C programming language. For the virtual prototype, the relay system is instead written as plugins to Gazebo, but performs the same functionality of reading a ROS topic and setting joint parameters (see Appendix D.7). These plugins are written in the C++ programming language.

To connect the three software components together, they are individually run as ROS nodes. Communication between these nodes is done by reading and writing to the ROS topics, which can be read by any node. This means individual nodes can be replaced by other systems, and if the interfaces match the rest of the system, it will not be affected. For example, to test the actuator relay, a script can be made that sets the actuator ROS topics manually, while normally they would be written to by the state machine. Likewise, this allows us to “drop in” the Gazebo simulation environment for the physical robot without requiring any software changes.

```
jack@apollo:~/LeviathanAUV/catkin_ws/src/lbr$ ./robot.sh
Starting overall state machine
Running overall state machine
Running IDLE state machine
Running CONSTRUCTION state machine
  Starting construction state machine
    In construction::neutral position
      There are 3 blocks to construct
    In contruction::elevate tower
    In contruction::reach cube
    In contruction::pull cube
    In contruction::deelevate tower
    In construction::neutral position
      There are 2 blocks to construct
    In contruction::elevate tower
    In contruction::reach cube
    In contruction::pull cube
    In contruction::deelevate tower
    In construction::neutral position
      There are 1 blocks to construct
    In contruction::elevate tower
    In contruction::reach cube
    In contruction::pull cube
    In contruction::deelevate tower
    In construction::neutral position
      There are no more blocks to construct
  Finished construction state machine
```

Figure 33: Terminal output of the overall state machine and construction state machine

```
jack@apollo:~/LeviathanAUV/catkin_ws/src/lbr/
/active
/bottom_servo_status
/climb_to_height_enable
/climb_to_height_target
/horizontal_actuator_status
/lbr_robot/bottom_plate_servo_position
/lbr_robot/cube_grabber_actuator_position
/lbr_robot/cube_grabber_servos_position
/lbr_robot/forklift_left_actuator_position
/lbr_robot/forklift_left_servo_position
/lbr_robot/forklift_right_actuator_position
/lbr_robot/forklift_right_servo_position
/lbr_robot/forklift_servos_position
/left_forklift_servo_status
/left_grabber_servo_status
/num_blocks
/reset
/right_forklift_servo_status
/right_grabber_servo_status
/rosout
/rosout_agg
/vertical_actuator_status
```

Figure 34: ROS topics used by the robot

3.4.5 Gazebo

Gazebo is a 3D rigidbody sandbox simulation program. In replacement of the physical prototype, Gazebo was used to simulate the robot in a 3D test environment [22]. To accomplish this, each moving component from the SolidWorks model was exported as an STL file. The robot was then reconstructed by creating Simulation Description Format (SDF) files that joined the parts by revolute, prismatic, or fixed joints. Inertia properties, collision properties, and joint limits were assigned to the components based off of SolidWorks's estimates. Gazebo was then able to import this reconstructed model, and we were able to write Gazebo plugins to relay the ROS topic values to these virtual joints. For simplification, the linear actuators and four-stage linear slide assembly were represented as prismatic joints instead of threaded interfaces or chain-and-sprocket systems. **Figure 35** depicts the structure-building robot in the Gazebo simulation environment.

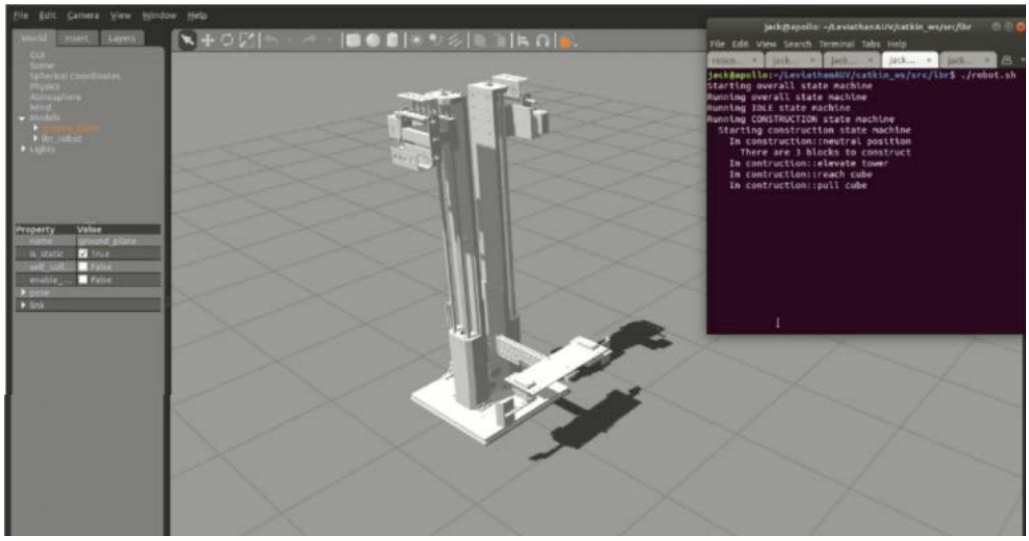


Figure 35: Robot model in the Gazebo simulation environment running the construction state machine.

The right side of **Figure 35** shows the terminal output of the construction state machine. The last item displayed is "In construction::pull cube". At this point of the state machine, the cube structure should be elevated, and the cube latch mechanism should be retracting its linear slides and pulling the cube to be placed underneath the structure. This matches the visual state of the robot shown in the Gazebo environment. The Gazebo environment and plugins respond quickly to the state machine outputs. Finally, the movement speeds of each joint were tuned to mimic those of their physical actuator equivalents.

3.5 Cube

Figure 36: Isometric view of the cube.

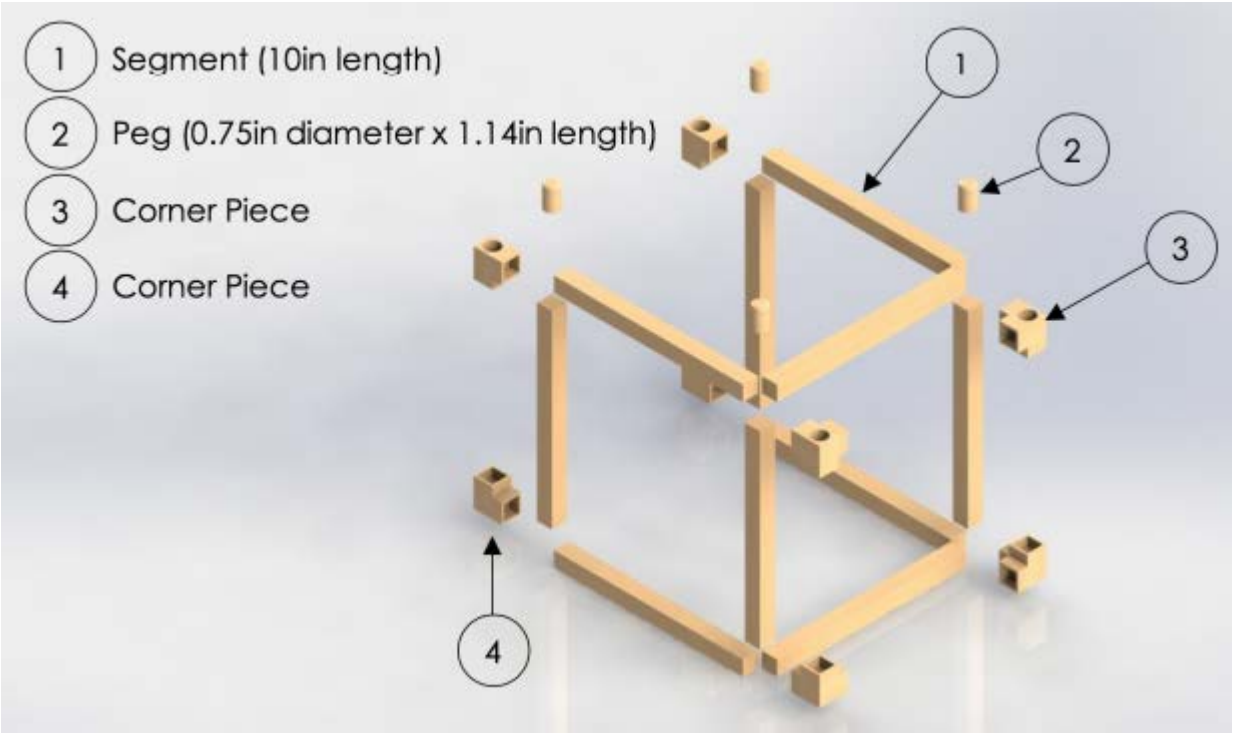


Figure 36: Exploded view of the cube.

When making a decision of the final robot design, the type of structure was an important factor. The cube element was selected because the robot design was made to scale from inside the structure. Since the center of mass of the robot does not shift when it climbs up the structure, there is no significant moment produced that could tip the robot and structure over. As seen in the step-by-step process of building the structure in **Figure 2**, the cube element was modified to have an open face to eliminate any interference with the linear actuators from the lifting and climbing mechanism. The cube element is made out of balsa wood, a light and sufficiently strong type of wood. As depicted in **Figure 37**, the cube element consists of four components: (1) segment, (2) peg, (3) corner piece, and (4) corner piece on the open face (see **Figure C4.6** in Appendix C.4). All components assemble a cube element with side lengths of 1ft. Each cube requires 10 segments, 4 pegs, 4 corner pieces, and 4 corner pieces on the open face.

The horizontal and vertical segments both have square cross-sections with side lengths of 0.8in, and the length of each segment is 10in (see **Figure B4.1** in Appendix B.4). On the other hand, the two variations of the corner piece have either two or three holes to interlock these segments (see **Figures B4.2** and **B4.3** in Appendix B.4). These holes are 0.5in in depth to fit the wooden segments. In addition, 0.75in-diameter holes must be drilled to allow for the pegs to fit and the cube elements to stack. Because the corner pieces require tight tolerances, blocks of balsa wood with side lengths of 1.5in must be obtained and cut with a CNC machine at high precision. Otherwise, the cube elements risk not being uniform, which can cause mechanical failure during construction or ascension. Wood glue is used to firmly attach all four components of the cube element together.

4 Design Verification

4.1 Overview of Design Verification

The virtual prototype was constructed to verify that the final design solution would function properly in a real-world environment. Because the final design solution is not purely mechanical but includes a software component, two different analyses were conducted to verify the solution. A structural analysis was conducted on every component to confirm whether or not they would fail under normal operating conditions. A kinematic analysis was also conducted to ensure that the robot would perform its tasks as expected with commands from the software package.

The structural analysis was conducted using two finite element analysis (FEA) software: SolidWorks Simulation and Abaqus. Each component was placed under the worst-case conditions (i.e. maximum load, etc.); if the components did not fail under these conditions, then it would be deemed that they would also not fail in a testing environment. On the other hand, the kinematic analysis was conducted by simulating the robot using Gazebo. With the implementation of this software, each motor, servo, and actuator can be simulated to mimic their real-world characteristics. Moreover, Gazebo includes a physics engine, which can demonstrate how the robot will interact with its environment. If the robot performs as expected in this virtual simulation, we can assume that it will be suitable in a real-world environment.

4.2 Assumptions

The models for each simulation do not differ drastically from the detail design. For the SolidWorks and Abaqus simulations, the material for every component was assumed to be uniform. For example, each 3D printed part was assumed to have 100% infill to simplify the analysis. As none of the 3D printed parts are being stressed against the plastic's printing grain, this assumption is supported. Each component was assumed to be adequately bonded together, and fasteners would not break under normal operating conditions. The simulation also does not include any wires, screws, or small cavities, which eliminates the interference with all other components on the robot. For the Gazebo simulation, code was written to generate a custom linear actuator modeled after those found commercially, which is listed in the Bill of Materials (see Appendix A). To reduce the computational stress of the simulation, the linear slides and chain-and-sprocket system were replaced with prismatic actuations. The Gazebo simulation assumes rigid body dynamics, which

is valid for our model as FEA reveals that the maximum bending of components is on the millimeter scale.

4.3 Finite Element Analysis (FEA)

4.3.1 Lifting and Climbing Mechanism

4.3.1.1 Lifting and Climbing Hooks

For the lifting and climbing mechanism, the dual hook assembly consists of two Aluminum 7005 hooks powered by a single servo. The bottom hook is used to lift cube elements up whereas the top hook is used to pull the robot up to each level. Therefore, the hooks must be able to support the weight of the cube structure as well as the weight of the robot. Two SolidWorks simulations were conducted to ensure that both the lifting and climbing hook would not fail.

Assuming the servo will constrain the position of the hooks as the robot either lifts or climbs the structure, a fixed boundary condition was applied to the shaft. For lifting the structure, a pressure load equal to the weight of a two-cube structure was applied on the inner face of the hook. Although the cube structure is built with only three cubes, only the top two are lifted and the third remains on the ground. For climbing the structure, a pressure load equal to the weight of the robot was applied on the inner face of the hook. The load and boundary condition on the hook is shown in **Figure 38**. Keep in mind that this takes the worst-case scenario. A more realistic analysis would be applying half the weights of the robot and two-cube structure since two hooks are used for lifting and climbing.

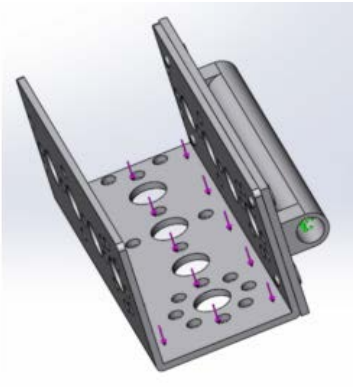


Figure 38: The load and boundary condition on the hook

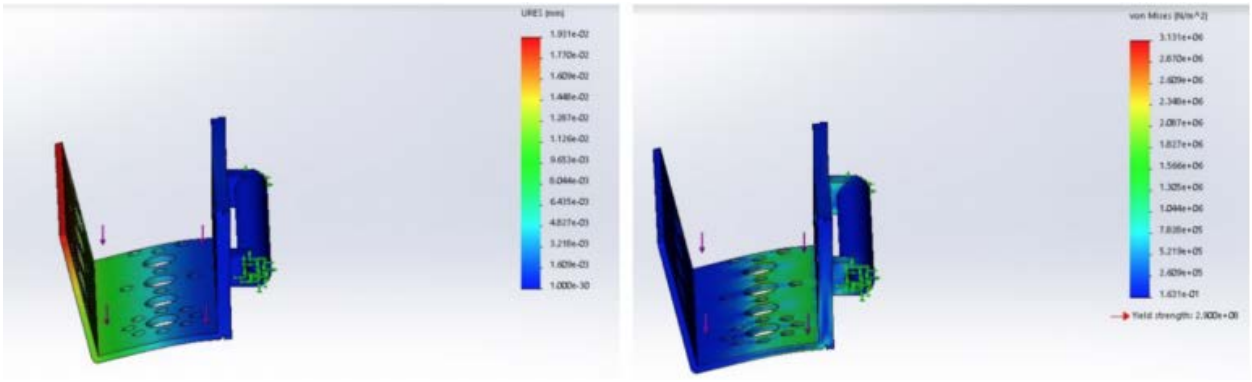


Figure 39: Contour plots of (left) displacement and (right) von Mises stress for the lifting hook.

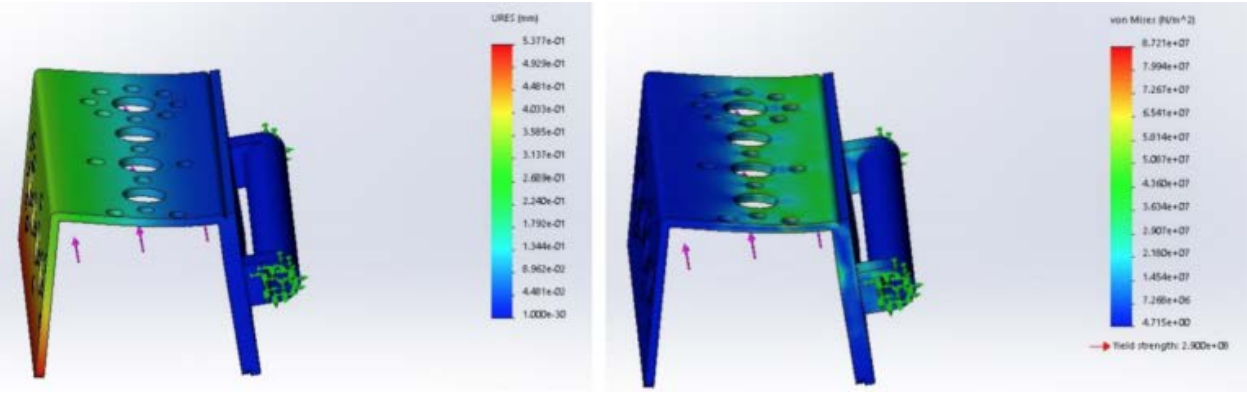


Figure 40: Contour plots of (left) displacement and (right) von Mises stress for the climbing hook.

To determine the stress on the lifting and climbing hooks, the von Mises stress was considered because it deals with failure criteria. The von Mises stress takes into consideration principal and shear stresses for an element in the 3D Cartesian plane, which is governed by **Equation 1**.

$$\sigma = \sqrt{\frac{1}{2}\left[(\sigma_{xx} - \sigma_{yy})^2 + (\sigma_{yy} - \sigma_{zz})^2 + (\sigma_{zz} - \sigma_{xx})^2\right] + 3(\tau_{xy}^2 + \tau_{yz}^2 + \tau_{zx}^2)} \qquad (1)$$

In both contour plots of displacement in **Figures 39** and **40**, it is evident that the maximum resultant displacement occurs at the tip of the hooks despite the load being applied in opposite directions. As for the maximum von Mises stress, it is only concentrated on the inner face of the C-channel closest to the aluminum tube. From the angle of the contour plots, the maximum von Mises stress (in red) is hidden. Since the loads are perpendicular to the surface, it causes bending stress and very minimal shear stress. **Table 1** organizes the maximum values of displacement and von Mises stress for each hook.

Table 1: Maximum displacement and von Mises stress for the lifting and climbing hooks.

Hook	Type of Load	Applied load (lb)	Max displacement (mm)	Max stress (MPa)
Lifting	Two-cube stack	0.88	0.01931	3.131
Climbing	Robot	24.51	0.5377	87.21

As previously mentioned, the lifting and climbing hooks are made out of Aluminum 7005, which has a yield stress of 290 MPa [23]. Since the maximum stress for both hooks are well below this yield stress, they will not fail. Nonetheless, it should be trivial that the maximum stress of the climbing hook is significantly higher than that of the lifting hook because of the differences in weight between the robot and the two-cube stack. Since the minimum factor of safety is the ultimate stress divided by maximum working stress, the minimum factor of safety of the lifting hook is 92.

4.3.2 Cube Latch Mechanism

The cube latch mechanism is split into two sub-mechanisms: the cascading linear slide assembly and the claw. Two finite element simulations were conducted to ensure that each mechanism would not fail under normal operating conditions.

4.3.2.1 Linear Slides

The linear slides consist of four cascading 15mm extrusions connected by several plastic joints, which provide linear motion for a well-defined range. As mentioned in **Section 3.2.3 Cube Latch Mechanism**, the linear slides act like a complex cantilever beam with a downward force on the free end (i.e. weight of the cube latch) and a support on the fixed end attached to the base plate. Due to the complex geometry of the linear slides as well as the small cavities between the joints, FEA could not be performed in SolidWorks Simulation; however, Abaqus was implemented instead.

To simplify the linear slide assembly for analysis, the 15mm extrusions were approximated as solid rectangular prisms with a homogeneous section. Since the extrusions are made out of Aluminum 6063, the material properties were set uniformly for the model [24]. Because the 15mm extrusions are similar to 80/20 extrusions, the geometry is too complicated to apply an accurate mesh. Therefore, the decision of using solid rectangular prisms was made. In actuality, the 15mm extrusions are connected by plastic joints, but the analysis assumed that there was no clearance between them. With this assumption, it is guaranteed that there would be a clear difference between the theoretical and actual deflections as the linear slide assembly is fully extended. An encastre boundary condition was set on the fixed end of the linear slides; in other words, the translation and rotation in all directions of the 3D Cartesian coordinate system was zero. As for the loads, the force due to gravity of the linear slides as well as the weight of the cube latch (0.66lbs) were applied onto the model. The loads and boundary conditions are displayed in **Figure 41**.

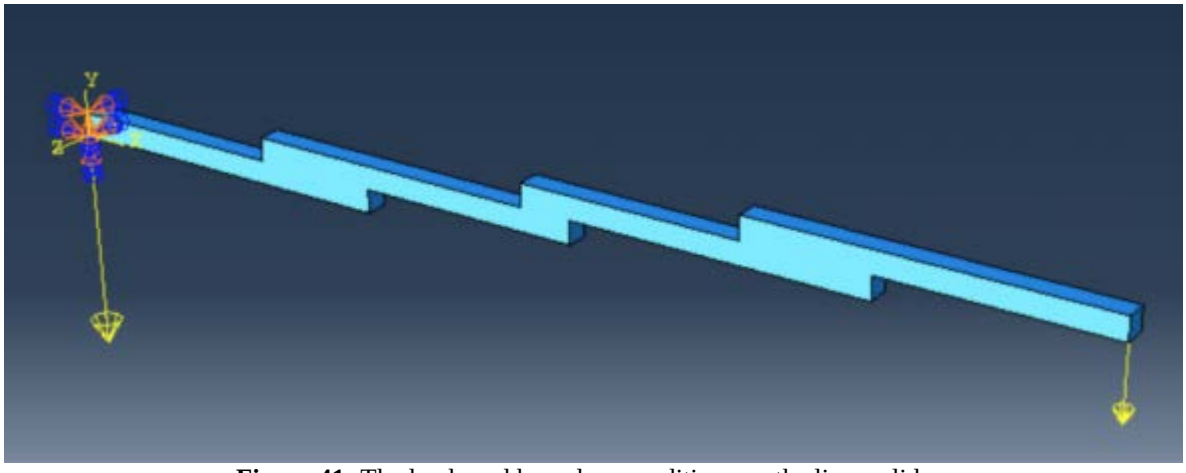


Figure 41: The loads and boundary conditions on the linear slides.

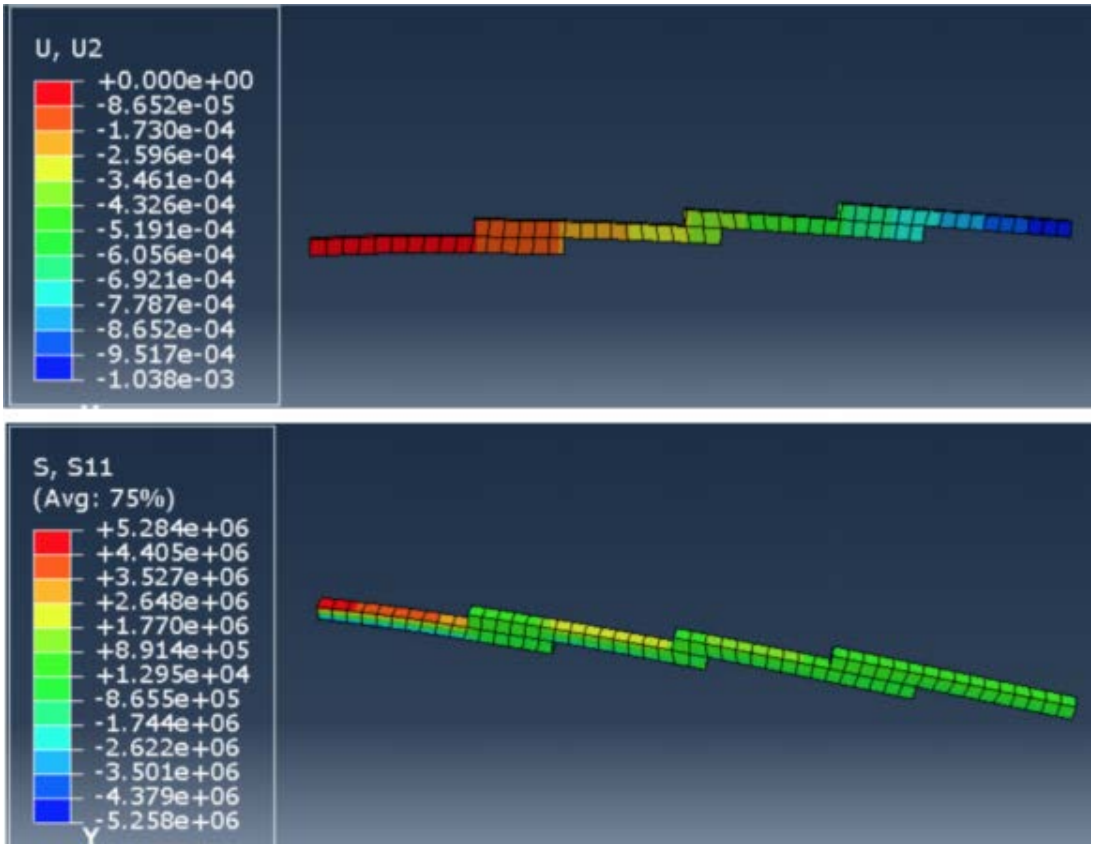


Figure 42: The loads and boundary conditions on the linear slides.

In **Figure 42**, the maximum displacement in the vertical direction (u2 = 1.038mm) occurs on the tip of the free end of the linear slide assembly. This is to be expected because of the additional load from the cube latch. Despite the assumptions and parameters used for this analysis, the maximum bending stress (S11 = 5.284 MPa) was not substantial and remained well below the yield stress of Aluminum 6063, which was 145 MPa [24]. It is evident that the weight of the cube latch on the free end and the total length of the fully extended linear slide assembly cause maximum bending on the fixed end. As the contour plot of bending stress shows, tension (in red) is concentrated at the top whereas compression (in blue) is concentrated at the bottom (hidden). Furthermore, the joints are made out of Delrin plastic with a yield stress of 62 MPa [25]. After taking the probe value at each node along the connecting extrusions, the maximum bending stress of the joint occurred at the intersection between the two extrusions on the left. This maximum value was 0.4 MPa, which was below the yield stress. Therefore, the linear slides can fully extend without significant deflection or stress on each extrusion.

4.3.2.2 Claw

The claw on the cube latch assembly was studied using SolidWorks Simulation to determine how it would be affected when pulling in a cube element from the outer perimeter. The cube latch assembly is comprised of two claws, each powered by a servo. Assuming that the servo supplies enough torque to keep the claw's position fixed as it reels in a cube, this was taken into consideration when applying the loads and boundary conditions into the FEA model. The boundary

condition was applied on the edge of the flat beam. In contrast, the load had to be theoretically calculated using a free-body diagram (FBD), which is shown in **Figure 43.**

Sum of forces in the y-direction:

$$\Sigma F_y = F_N - F_G = 0$$

$$F_N = F_G \rightarrow F_N = 0.44lb \quad (2)$$

Sum of forces in the x-direction:

$$\Sigma F_x = F_f - F_{Pull} = 0$$

$$F_{Pull} = F_f = \mu_k F_N = (0.5)(0.44lb) = 0.22lb \quad (3)$$

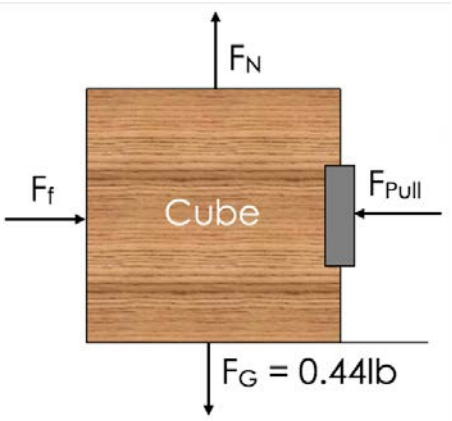


Figure 43: The free-body diagram of the cube.

Several assumptions had to be made in this calculation. First, in a real-world environment there would be friction between the cube and the surface it is on; however, the coefficient of kinetic friction was assumed to be 0.5. Second, because it is difficult to quantify the velocity and acceleration that the cube travels when it is pulled, the velocity is assumed to be constant. Third, although this scenario involves rigid body dynamics, the cube is representative of a point, so no moments are considered. Therefore, the load applied onto the claw was the pulling force or 0.22lb.

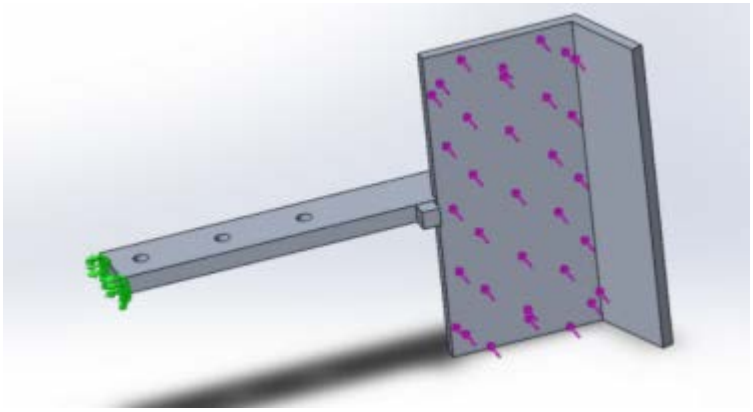


Figure 44: The load and boundary condition on the claw

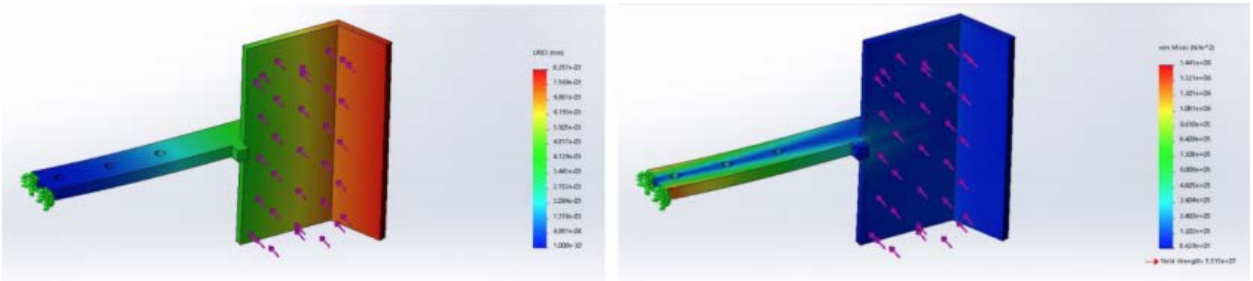


Figure 44: Contour plots of (left) displacement and (right) von Mises stress for the claw.

From **Figure 45**, the maximum displacement and maximum von Mises stress are 8.257 µm and 1.441 MPa, respectively. Since the maximum displacement is approximated to zero, it is negligible. As mentioned in **Section 3.2.3.2 Cube Latch Assembly**, the claw is made out of Aluminum 6061, which has a yield stress of 276 MPa [12]. It is apparent that the claw will not yield when pulling in a cube element.

4.3.3 Rotating Plate Mechanism

The rotating plate mechanism consists of the base plate and rotating plate connected together by a servo. As mentioned in **Section 3.2.4.1 Rotating Plate**, eight rubber stoppers are glued to the bottom of the base plate to redistribute the load from the servo to the base plate. The finite element model was used to determine if the rotating plate mechanism would break or deform under the weight of the robot. After applying all material properties into SolidWorks, the theoretical weight of the robot was 23.51lbs. To simplify the model for analysis, the servo and screws were removed from the model. A fixed constraint was then applied to the top face of the base plate; a pressure load equal to the robot's weight was applied to the bottom face of the rotating plate. **Figure 46** shows the load and boundary conditions applied to this mechanism.

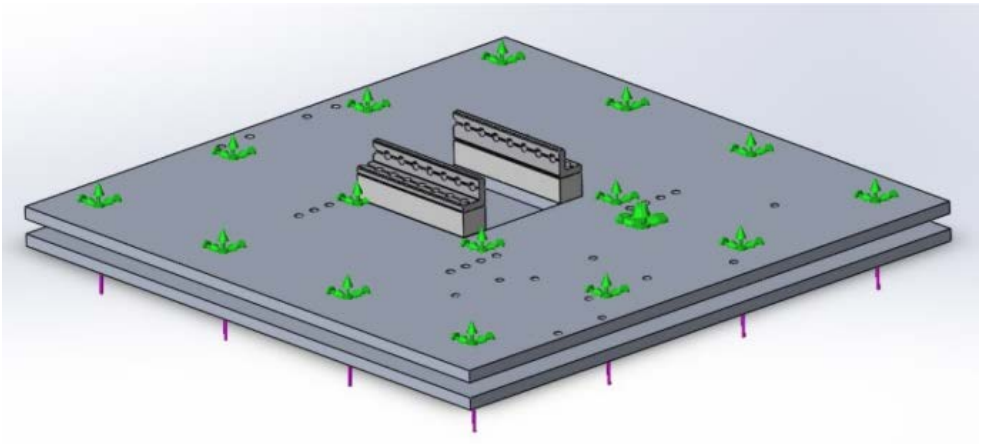


Figure 46: The load and boundary condition on the rotating plate mechanism

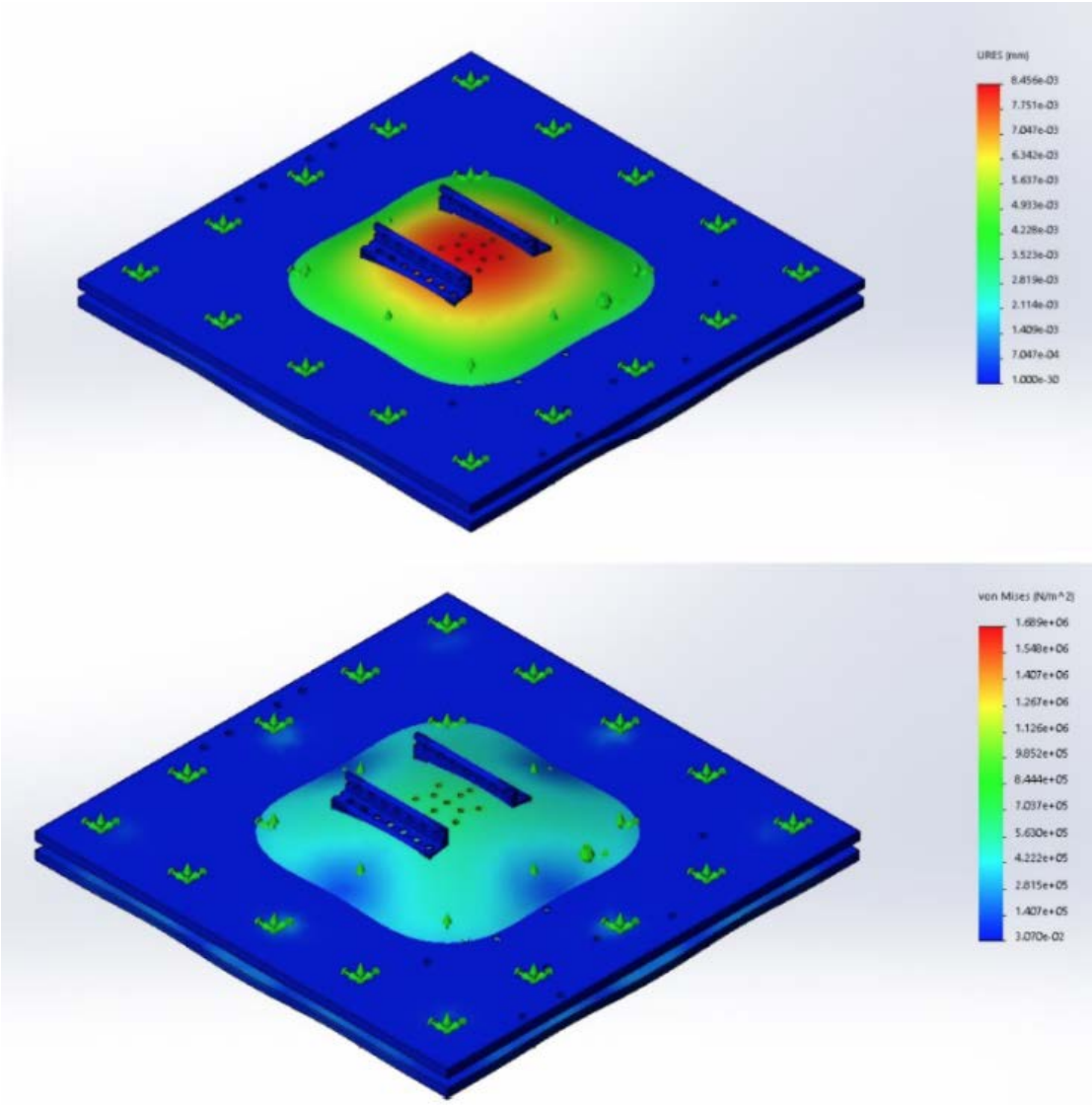


Figure 47: Contour plots of (top) displacement and (bottom) von Mises stress for the rotating plate mechanism

The von Mises stress was used once again as it relates to failure criteria. Although the rotating plate mechanism experiences very minimal shear stress, the von Mises still considers it as expressed in Equation 1. From Figure 47, the displacement is concentrated at the center of the rotating plate as expected. Although the contour plot of displacement exaggerates the maximum displacement, its value 8.46μm is so small that it is negligible. On the other hand, the maximum von Mises stress was 1.69 MPa. Since the yield stress of Aluminum 6061 is 276 MPa, the rotating plate mechanism should be able to support the robot's weight without fail [12]. The factor of safety is expressed as Equation 4, which is the ratio of ultimate tensile strength and maximum stress from FEA.

$$FS = \frac{\sigma_{ult}}{\sigma_{max}} \tag{4}$$

Since the ultimate tensile strength of Aluminum 6061 is 310 MPa, the factor of safety for the rotating plate mechanism is 18.6 [12].

4.3.4 Cube Structure

The structure is constructed by stacking several five-sided cube elements on top of one another. To ensure that the structure can be sound and function as expected, several finite element analyses were conducted: (1) on stacked cubes for entire structure and (2) on cube supported by the robot's weight at each level of the structure. To test the first case, the three-cube structure was placed into the SolidWorks Simulation. A fixed constraint was applied to the bottom faces on all bottom-faced corner pieces since they were the only points of contact with the ground. Moreover, a body force equal to gravity was placed on the structure, which can be seen in **Figure 48.**

From **Figure 49**, the maximum displacement and maximum von Mises stress on the cube structure in its natural state are 1.101µm and 5.277 kPA, respectively. Although no considerable weight is applied to the cube structure for the first case, the maximum displacement occurs at the very top; since the cube element was modified to have an open face, there are no fixed horizontal segments, which makes the top of the cube structure sink. This sets a baseline as to how the cube structure could be deformed when the robot's weight is applied on each level during ascension or descension.

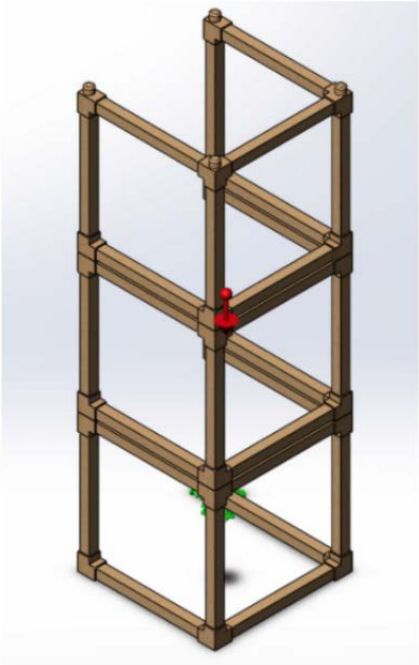


Figure 48: The load and boundary condition on the cube structure.

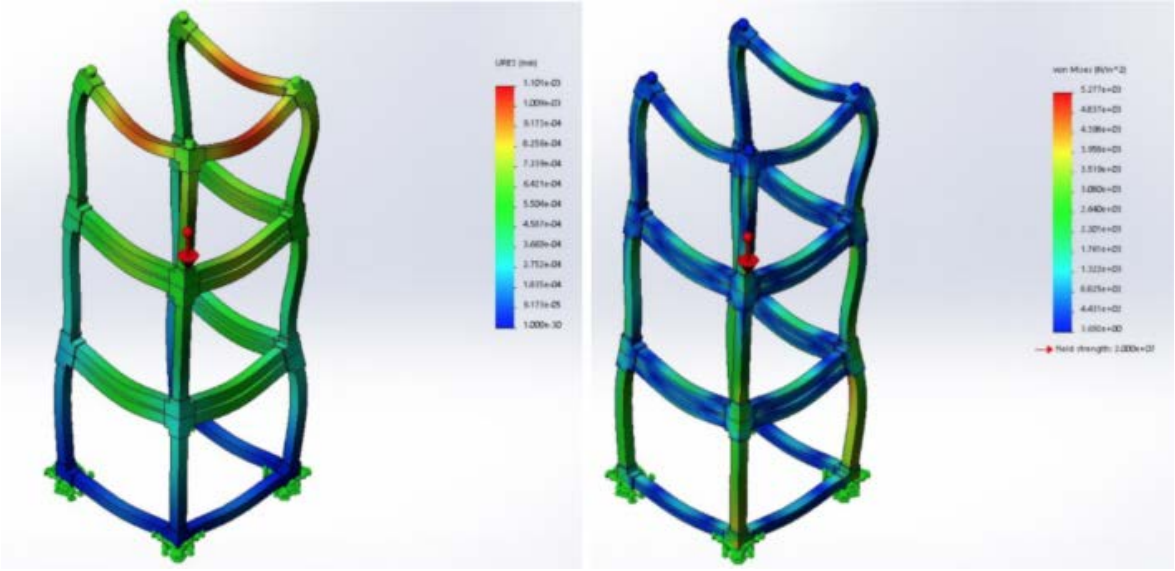


Figure 49: Contour plots of (left) displacement and (right) von Mises stress for the cube structure.

Since the cube structure must be able to support the robot's weight at every level, the simulation was modified to include the additional load. Although the rotating plate only makes contact with part of the horizontal segments of the cube structure, the analysis assumed that it made full contact instead. The loads and boundary conditions for this second case of the cube structure is shown in **Figure 50**.

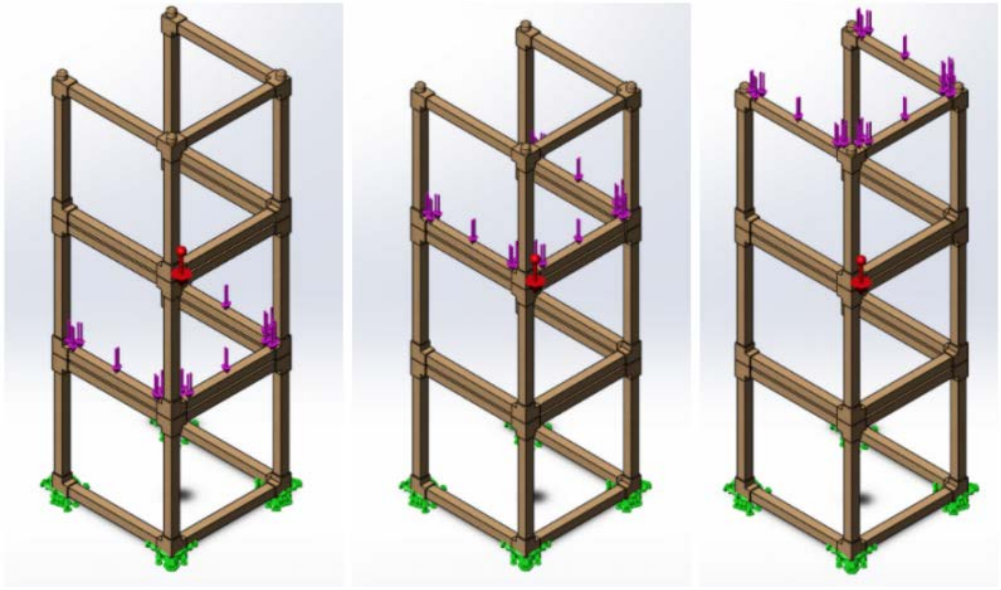


Figure 50: The loads and boundary conditions on the cube structure when it is supported by the robot at the bottom, middle, and top.

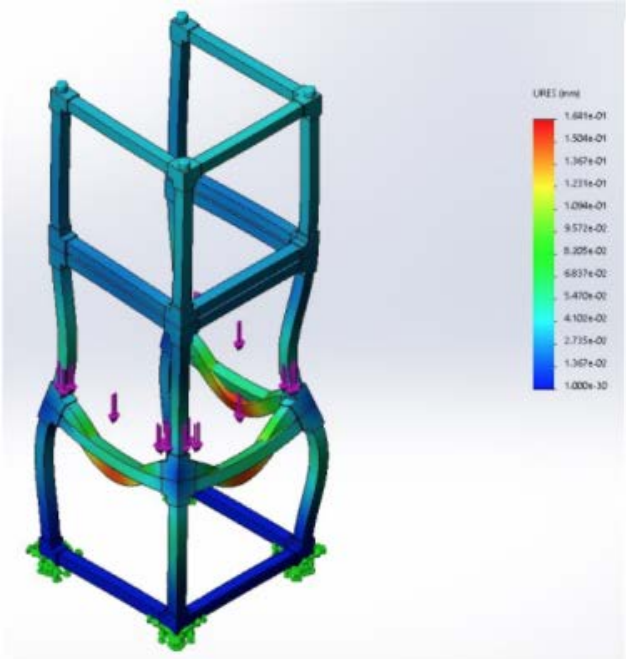


Figure 51: Contour plot of displacement when the robot is supported on the bottom level.

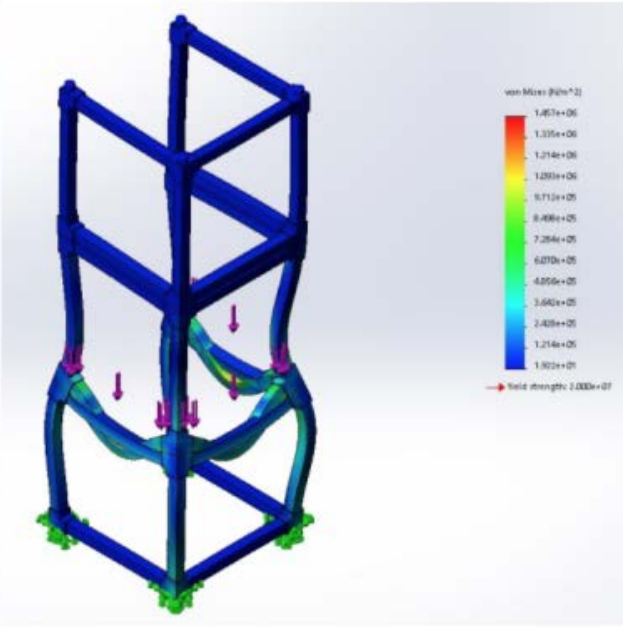


Figure 52: Contour plot of von Mises stress when the robot is supported on the bottom level.

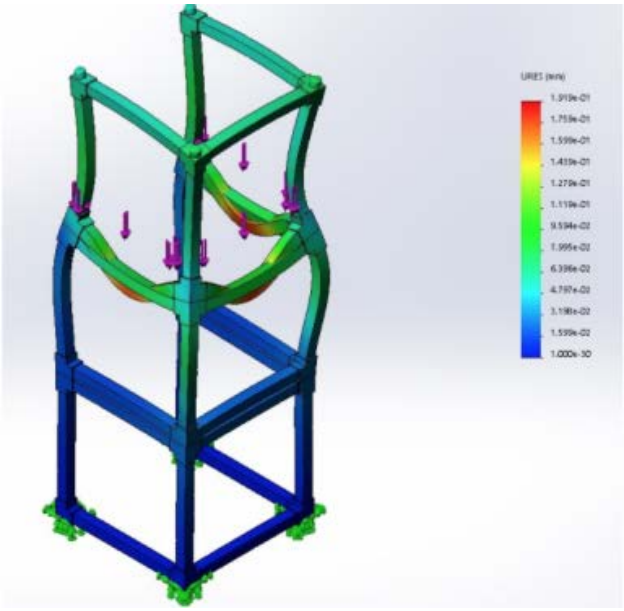


Figure 53: Contour plot of displacement when the robot is supported on the middle level.

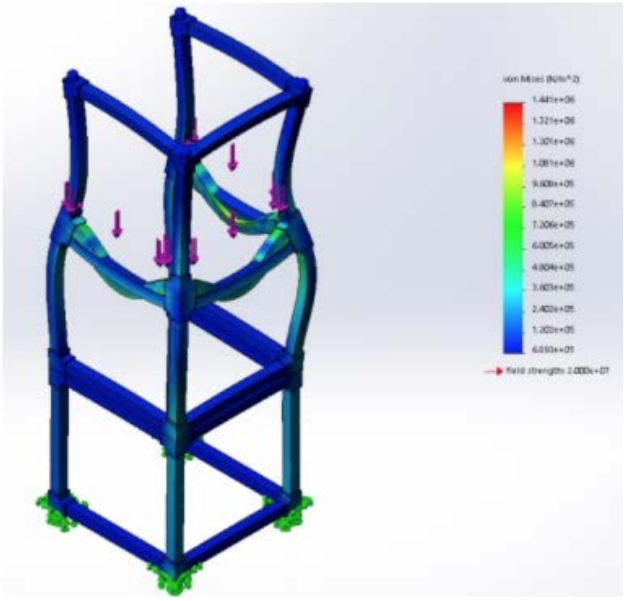


Figure 54: Contour plot of von Mises stress when the robot is supported on the middle level.

From **Figures 52** and **54**, the maximum von Mises stress is similar when the robot is supported by the cube structure on the bottom and middle levels. In terms of maximum displacement, they also do not differ drastically. On the other hand, in **Figures 54** and **56**, there was a jump in both these values when the robot is supported by the cube structure on the top level. Despite that the structure is only three cubes tall, both maximum displacement and maximum von Mises stress increases with increasing height of structure. **Table 2** organizes these values accordingly.

Table 2: Maximum displacement and von Mises stress at each level when the robot is supported by the cube structure.

Level	Max displacement (mm)	Max von Mises stress (MPa)
Bottom	0.1641	1.457
Middle	0.1919	1.441
Top	0.4143	2.753

Since the cubes are made out of balsa wood, they have a yield stress of 20 MPa [26]. Referring to each maximum von Mises stress from **Table 2**, there is no chance that the structure will fail each time the robot scales to a new level.

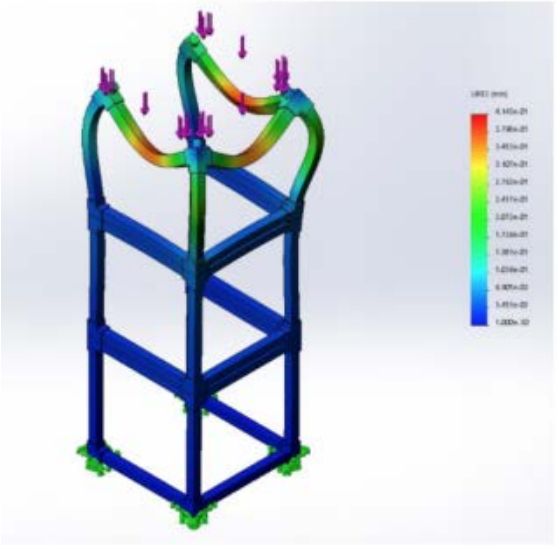


Figure 55: Contour plot of displacement when the robot is supported on the top level.

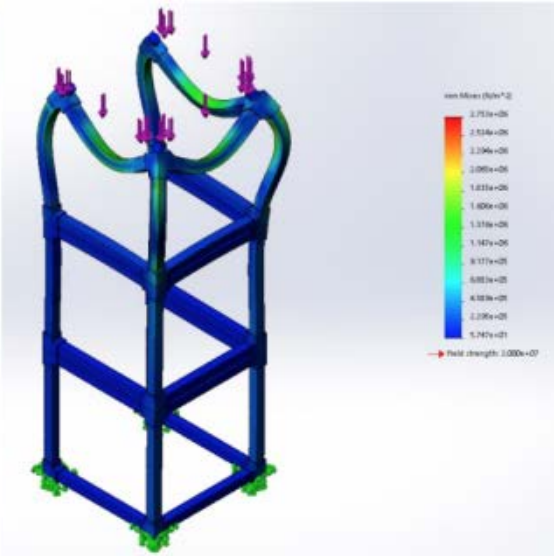


Figure 56: Contour plot of von Mises stress when the robot is supported on the top level.

4.4 Gazebo Simulation

As confirmed by finite element analysis (see **Section 4.3 Finite Element Analysis (FEA)**), the deflection of all contact-interface components is negligible due to maximum deflections around 1mm. This supports an approximation of the physical structure as a rigid body system. Gazebo's simulation environment contains a physics engine that uses the rigid body approximation; therefore, Gazebo can be used to verify the mechanical motion of the design. The state machine complex and the ROS communication framework were tested using Gazebo as the physical environment. The component software package was run on an Ubuntu Desktop environment, which is similar to the Ubuntu MATE environment used by the Raspberry Pi. Utilizing a similar operating system verifies that the software components would run as expected on the physical model. Gazebo's virtual environment verified that the actuators' responses to the state machines' outputs accurately reflect the intended autonomy of the system.

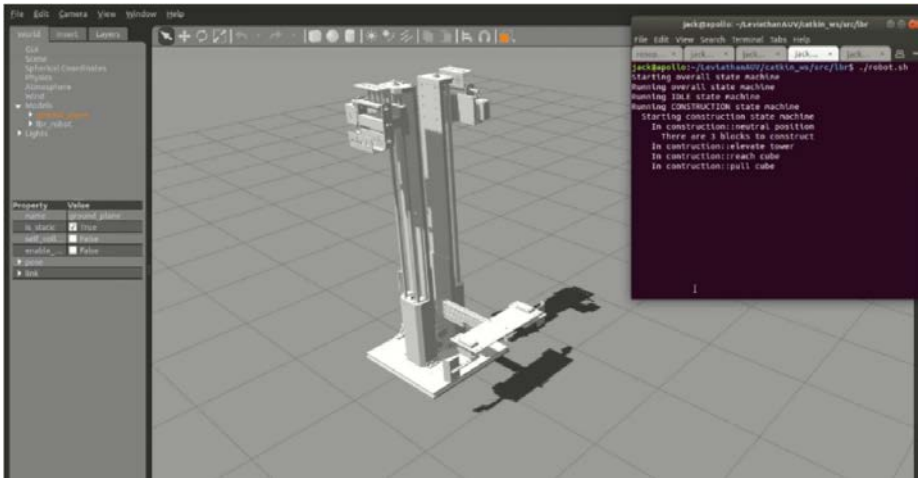


Figure 57: Robot model in the Gazebo environment, running the construction state machine

5 Approach to Solution

5.1 Brainstorming Methods

When the team was first assigned this project at the beginning of the 2020 Winter quarter, the ambiguity of the problem statement led to many different interpretations on how to approach to a solution. These interpretations include the type of structure the robot will build and how the robot will scale the structure. Ultimately, the latter depends on the former. After thoroughly understanding the backbone of the problem (see **Section 3 Problem Definition**), the brainstorming method was used to generate concepts for both the structure and scaling method.

Formally known as the 6-3-5 method, the 4-3-5 method was used for this brainstorming process. Here, four team members draw three unique designs on paper in five-minute rotations [27]. This process is repeated until all four papers have reached every team member or no unique solutions can be generated. The brainstorming session resulted in 48 concepts generated in 20 minutes.

5.2 Structure

The brainstorming process was also applied to the structure, which was condensed down to three main elements: ladder, cube, and pole. Each element has its own advantages and disadvantages, but are elaborated further in the following sections.

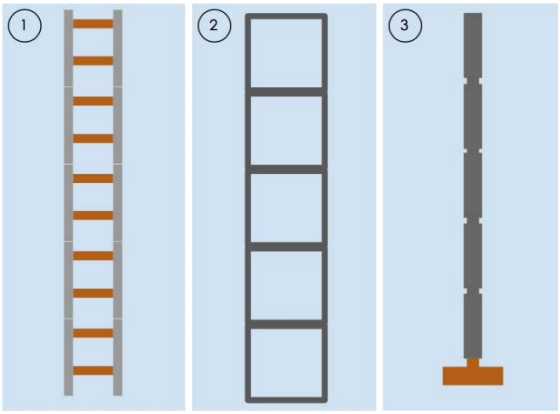


Figure 58: (1) Ladder, (2) cube, and (3) pole structures

5.2.1 Ladder

The ladder is a common structure used to help people (i.e. construction workers, etc.) elevate themselves to greater heights. The most common commercially available ladders are the extension ladder, the step ladder, and the dual purpose ladder [9]. Although these all vary in size, shape, and material, the traditional ladder was considered in the design. The ladder is constructed using interlocking rungs, but must be supported by both a wall and the ground. Although the rungs can be sectioned to easily attach to the overall structure, the design itself poses many challenges. Because the ladder is not a standalone structure and must be supported by a wall, the ladder must be at an angle θ above the ground. If this angle is too steep, the ladder will be prone to tipping due to the robot's weight and center of mass. Additionally, as more rung elements are added onto the ladder, it will be more vulnerable to bending and deflection, which could cause the structure to break and fail.

5.2.2 Cube

The cube structure was considered to allow the robot to climb through the middle of the structure. This is advantageous because the center of mass of the robot is close to that of the structure, decreasing the probability of tipping. The cube is a self-supporting structure consisting of several members connected together to make a six-sided shell. Because multiple cubes can be stacked on top of one another to create a standalone structure, this design is more promising as it does not require support from a wall. On the other hand, the disadvantages of the cube structure are that it is more costly and difficult to manufacture.

5.2.3 Pole

The pole structure consists of interlocking pole elements that can stack. This particular design requires a foundation or fixture for the robot to build off of. Although this design is considered, it is inherently less stable than either the cube or ladder. As more pole elements are added to the structure, it is more vulnerable to bending and deflection. In addition, the pole must be under optimal conditions (i.e. no external forces, no environmental changes, etc.) to function properly.

5.3 Concept Generation

After reviewing each design from the 4-3-5 method as a team, we collectively agreed and narrowed down the top five most promising robot designs. Since each design must be compatible with a specific type of structure, the decisions made were not swayed by the structure itself, but the robustness, effectiveness, and uniqueness of the robot designs that best suited the defined requirements (see **Section 2.2 Specifications and Constraints**). The top five robots designs were the “Quiver”, “Cart”, “Ladder”, “Spider”, and “Pole”.

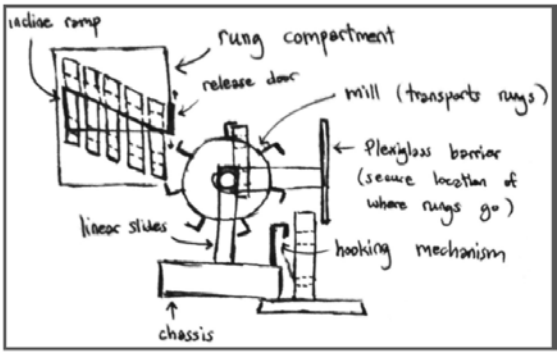


Figure 59: Sketch of Design #1: "Quiver".

5.3.1 Design #1: "Quiver"

Quiver was first designed to enable the robot to simultaneously build the ladder structure as it climbs. In **Figure 59**, the sketch features a rung compartment located behind the robot and a rotating mill to place rungs onto the structure. An inclined plane is found inside the rung compartment to assist the rungs to fall down due to gravity. A release door mechanism on the rung compartment controls the rate at which the rungs can enter the rotating mill. As each rung passes through the mill, a Plexiglass barrier ensures that the rung will be placed on top of the previous one with minimal error. A set of linear slides with hooks is located at the front to allow the robot to climb the structure.

5.3.2 Design #2: "Cart"

Cart was designed to allow the robot to travel along a bidirectional track. A rung compartment is located on one end of the track whereas the structure is located on the other end. The robot must collect a rung, drive to the other end, stack the rung on the structure, return to the rung compartment, and repeat. This design utilizes a conventional ladder with a foundation or fixture as a support. The robot also features a set of linear slides that share lifting and climbing hooks used to elevate the rungs and scale the structure, respectively. As opposed to simultaneously building and climbing the structure like Design #1, this design builds the entire structure first, then scales it.

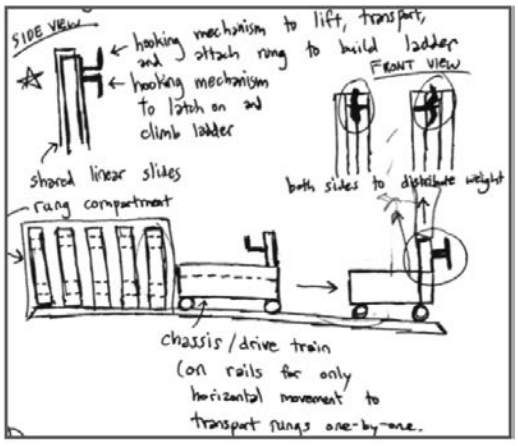


Figure 60: Sketch of Design #2: “Cart”.

5.3.3 Design #3: "Ladder"

Similar to Design #1, this design simultaneously builds the ladder structure as it climbs. It also includes a rung compartment attached to the back of the robot; however, the transportation of the rung elements is different. A two-link arm mechanism on the robot reaches into the compartment, grabs a rung, and places it onto the structure. To ascend the ladder, four claws or grippers are clamped onto the vertical members of the ladder, and a driven wheel allows the robot to scale upwards.

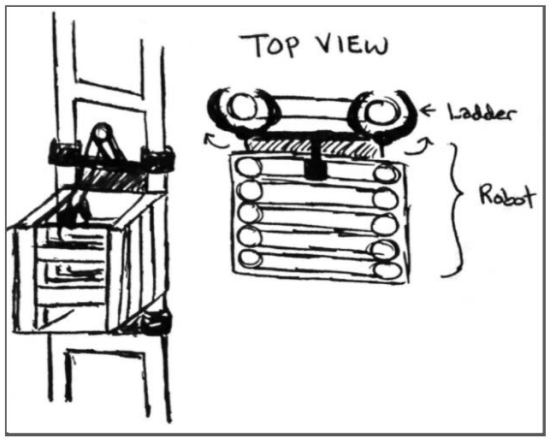


Figure 61: Sketch of Design #3: "Ladder".

5.3.4 Design #4: "Spider"

As opposed to the previous three design of using a conventional ladder, Spider was designed to be compatible with cube elements. This design employs a bottom-up approach in constructing the cube structure, which follows the description in **Section 5.2.2 Cube**. As shown in **Figure 62**, the three main features are the cube latch, forklift, and climbing mechanism. The step-by-step process of building the structure is as follows: (1) the robot raises a cube with its forklift, (2) extends the cube latch to the outer perimeter, (3) grabs onto a cube, and (4) pulls the cube in to be stacked. This procedure repeats until the number of cube elements is exhausted and the structure is entirely built. Then, shall the design utilize its climbing mechanism to scale from the inside of the structure.

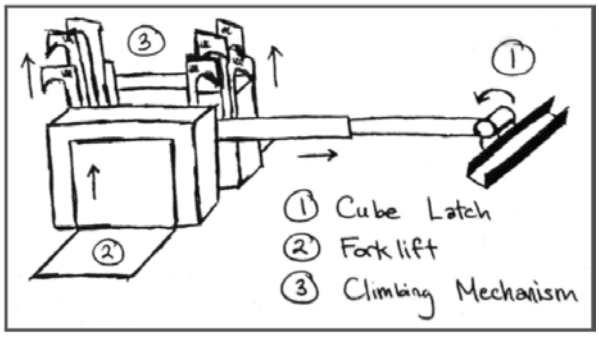


Figure 62: Sketch of Design #4: "Spider".

5.3.5 Design #5: "Pole"

As opposed to the previous four designs, Pole introduces a new element. The structure consists of interlocked cylindrical elements that assemble like Lego pieces. A two-link arm mechanism on this robot is used to grab a pole element from a separate compartment and stack it on top of the structure. As shown in **Figure 63**, the robot is ring-shaped with three interior wheels that are fully in contact with the pole structure, so it can translate vertically.

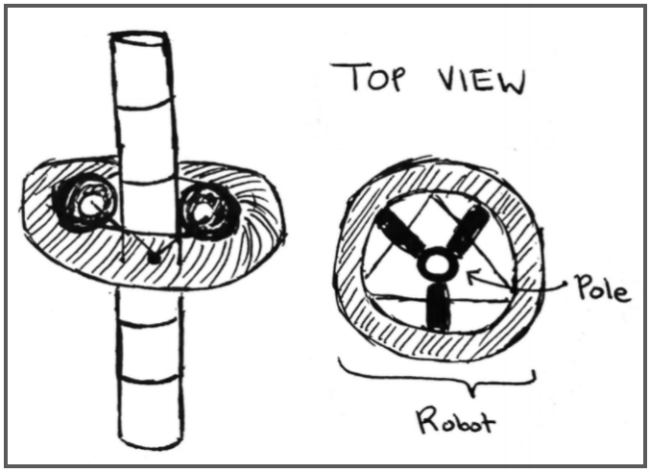


Figure 63: Sketch of Design #5: “Pole”.

5.4 Concept Selection

After the team mutually agreed on the top five robot designs in the concept generation phase, we moved towards selecting a final design. In order to do so, we considered the advantages and disadvantages of each design, generated a weighted decision matrix based on the requirements from the problem definition, and made a justification for the design with the highest weighted total score.

5.4.1 Advantages and Disadvantages

Design #1, “Quiver”, was a design that posed several challenges. Despite that it satisfies several functional requirements, this design would be difficult to achieve in reality. As it utilizes a traditional ladder with a fixture on the ground, the probability that the robot will tip and fall is high. Since the robot’s weight and center of mass are not aligned with those of the ladder structure, a moment is produced. In this design, the height of the ladder is greatly limited by the rung capacity that the compartment can hold. Another obstacle is determining how the rung compartment would attach to the robot without interfering with the rotating mill and chassis. Design #2, “Cart”, experiences similar issues with the previous design in terms of the tipping moment as well as the limitation of the rung capacity. Despite that the robot’s motion is constrained on the bidirectional track, it would be difficult to program it execute the states perfectly. Design #3, “Ladder”, also faces the same barriers as Design #1. On the other hand, Design #4, “Spider”, is a novel design that enables the robot to climb from the middle of the cube structure. The main challenge with this design, however, is with ascending and descending the structure due to the geometric constraints of the structure. Finally, Design #5, “Pole”, is definitively the simplest to implement. But, the cost of this simplicity is that the structure may fail due to bending and deflection before the robot scales to the very top. If the three driven wheels are not fully synchronized, then they could lock and immobilize the robot.

5.4.2 Weighted Decision Matrix

In the concept selection phase, a weighted decision matrix was generated to fairly determine the best design solution. A weighted decision matrix is a table used to rank the designs based on a set of requirements and weights [27]. The weights are essentially a magnitude to give each requirement more influence in the final decision; the weight was scaled from 1-10 depending on how important each requirement was to the overall goals of the project. After completing the weighted decision matrix, the design with the highest weighted total would be considered the best design. The selected requirements are listed as follows: cost, ease of manufacturing, construction speed, feasibility, size, accuracy, strength, and scalability.

Cost was weighted the highest at a 10 since the solution cannot exceed the budget of $300. Feasibility, scalability, and strength were also deemed important, so they were weighted a 9. Since the modeling, prototyping, and analysis must be completed within the 10-week time frame in the 2020 Spring quarter, the design solution must be feasible to ensure that it can be completed. Moreover, the solution must also be scaled such that it can be supported in a real-world environment. Since the weight requirement cannot exceed 30lbs, the structure must be able to support the robot’s load at the top without failure. Accuracy in grabbing each element and building the structure was weighted a 7; without a rigid structure, the robot is unable to ascend and descend it. In terms of manufacturability, it was weighted a 6 because ideally, we would want to use as many commercial off-the-shelf parts as possible and refrain from making custom ones. Lastly, construction speed and size were weighted 4 and 5, respectively, because they do not play a role in the primary objectives of the robot—to build, ascend, and descend the structure.

Requirements	Weight	Design #1: Quiver		Design #2: Cart		Design #3: Ladder		Design #4: Spider		Design #5: Pole	
		Score	Weighted Total	Score	Weighted Total	Score	Weighted Total	Score	Weighted Total	Score	Weighted Total
Fits within $300	10	4	40	7	70	8	80	6	60	10	100
Manufacturability	6	5	30	7	42	8	48	6	36	8	48
Rate of Construction	4	7	28	8	32	7	28	5	20	7	28
Feasibility	9	6	54	6	54	7	63	6	54	8	72
Within Size Requirements	5	4	20	9	45	10	50	10	50	10	50
Accuracy in Building Structure	7	8	56	4	28	7	49	7	49	8	56
Can support 30lb at the Top of Structure	9	2	18	6	54	5	45	10	90	3	27
Scalability	9	3	27	4	36	7	63	9	81	4	36
TOTAL			273		361		426		**440**		417

Figure 64: Weighted decision matrix.

5.4.3 Design Justification

After carefully assigning all scores for each requirement to each design, the completed weighted decision matrix is shown as **Figure 64**. As a collective unit, the team believed that Design #4, “Spider”, was the best design that satisfies all the requirements. Although the weighted total score of this design was slightly above those of the “Ladder” and “Pole”, a further analysis of these three designs validated that our selected design was the most feasible. Unlike the ladder and pole structures, the cube structure is inherently stable and self-supporting. All other designs had high chances of bending and tipping due to shifts in the center of mass, but since “Spider” remains in the middle of the structure as it climbs, this is not an issue.

Reviewing several of the physical requirements (see **Section 2.2.2 Physical Requirements**), “Spider” satisfies both PR-1 and PR-2. As described in **Section 5.3.4 Design #4: “Spider”**, the design features three main mechanisms: cube latch, forklift, and climbing mechanism. Out of the top five designs, “Spider” is the only one that can support itself at the very top of the cube structure. Moreover, depending on the size of the robot and each cube element, “Spider” will also satisfy several functional performance requirements (see S**ection 2.2.1 Functional Performance Requirements**). Since FPR-1 states that the robot must climb to a height of at least three feet, the cube element can have side lengths of 1ft, so only three cubes can be stacked to fulfill this requirement. Although time is relative, the repetitive process of building, ascending, and descending should reduce the amount of time for completion. This applies to FPR-2, FPR-4, and FPR-5. Finally, with this design, human factor is limited, but includes user interface (i.e. cell phone or laptop) and positioning cube elements on the outer perimeter (see **Section 2.2.5 Human Factors**). Although most requirements are met, the issues of cost and manufacturability remain important, unanswered questions.

6 Conclusion

Through the use of the engineering design process, this project has evolved into the detailed design package and virtual prototype demonstrated in this report. Starting from detailing the core requirements for a robot capable of autonomously constructing and scaling a structure, we brainstormed potential concepts, selected a final solution, and developed the solution into a final product. Using a virtual prototype, we verified that the project meets all project requirements. An extensive finite element analysis revealed insight into the components' factors of safety, and a 3D virtual simulation verified that the autonomous software package allows the robot to construct, ascend, and descend the structure. While this robot is just a proof-of-concept, it successfully demonstrates the potential for robots in construction sites to reduce human involvement on scaffolding structures. To commercialize this project, several aspects of the design would need to be revised, including redesigning the cube structure to increase the maximum structure height, searching for alternative commercial off-the-shelf components for some actuators, and finding exact use cases that this robot could address. Overall, our design verification has demonstrated that the product design is sound and ready for future testing and potential commercialization.

7 Recommendations

The unique aspect about this project has been that it requires a series of subsystems to consider during the engineering design process. The overall project is divided into four main systems—mechanical, electrical, software, and structure, each with their own subsystems. Strictly looking at physical components of the project, the mechanical subsystems and the structure had to be designed in conjunction with one another. Moreover, if a design of the structure was revised, then the mechanical design of the robot had to be reanalyzed in order to ensure the modification of the structure did not affect the intended functionality of the robot (and vice versa). Many times, redesign was required for both components, which was a challenging aspect of the engineering design process. Recognizing this hurdle, the main recommendation for design implementation would be to begin the engineering design process with the understanding that the robot and structure are two different entities that must be designed dependent on one another.

To improve upon the design, the cube element should be analyzed more closely in order to understand the operational constraints of the structure because it does not contain two edges to form a full six-sided cube. Though the structure has been determined to withstand high stresses, there is a height limit that the structure will begin to collapse due to the robot's load while ascending and descending the structure. This does not warrant any concern in regards to a virtual prototype; however, if the product is to be intended for future manufacturing, the design should be examined in greater detail to understand the height limit the robot can operate at which can affect the intended market focus.

Additionally, if the schedule were more lenient, the commercial off-the-shelf and modified offthe-shelf components chosen from the supplier could be chosen more carefully. Given the results from the finite element analysis, certain high stress areas were noted, and a commercial off-theshelf component can be modified to mitigate the stress concentration. However, if a commercial off-the-shelf product is modified too much, the time required to perform the modifications is greater than the benefit of using the part. Therefore, more research should be done to determine if other commercial off-the-shelf parts will function better in the design, and the operational envelope can be extended even further.

Finally, it would be wise to complete more research into specific areas at a construction site that the robot could be used for. The main goal of the project was to reduce the amount of injuries and fatalities due to falls from scaffolding and ladders; however, use cases were not fully defined. For example, it would be helpful to see if the robot could be used as a lift for materials and tools or as an elevator that can operate without human involvement. This research can help bring the prototype to the market quickly and more effectively.

References

[1] Occupational Safety and Health Administration.“Stairways and Ladders - A Guide to OSHA Rules.” OSHA, U.S. Department of Labor, 2003, www.osha.gov. Accessed 16 January 2020.

[2] Occupational Safety and Health Administration.“Safety and Health Regulations for Construction.” OSHA, United States Department of Labor, 11 April 2014, www.osha.gov. Accessed 16 January 2020.

[3] Occupational Safety and Health Administration.“Construction Focus Four: Fall Hazards.” OSHA, U.S. Department of Labor, 2011, www.osha.gov. Accessed 29 January 2020.

[4] Fujii, Shota, Inoue, Kenji, Takubo, Tomohito, Mae, Yasushi, and Arai Tatsuo. “Ladder Climbing Control for Limb Mechanism Robot ‘ASTERISK’.” Proceedings of the 2008 IEEE International Conference on Robotics and Automation. Pasadena, CA, USA, May 19-23, 2008. DOI:978-1-4244-1647-9/08.

[5] Yoneda, Hironari, Sekiyama, Kosuke, Hasegawa, Yasuhisa and Fukuda, Toshio. “Vertical Ladder Climbing Motion with Posture Control for Multi-Locomotion Robot.” Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems. Nice, France, September 22-26, 2008. DOI: 978-1-4244-2058-2/08.

[6] “68% Of the World Population Projected to Live in Urban Areas by 2050, Says UN—UN DESA Department of Economic and Social Affairs.” United Nations, United Nations, 16 May 2018, www.un.org.

[7] “Home Insurance Building.” The Home Insurance Building. Adams and LaSalle, Chicago Map Data OpenStreetMap. Chicago Architecture Info, www.chicagoarchitecture.info.

[8] Occupational Safety and Health Administration. “Worker Safety Series - Construction.” OSHA, U.S. Department of Labor, 2005, www.osha.gov. Accessed 29 January 2020.

[9] “Ladders in Construction.” SiteSafe, 2019, www. sitesafe.org.nz.

[10] Webb, B. “Major Types of Scaffolding in Construction.” Avontus, 2017, www.avontus.com.

[11] DSM&T. “IP Rating Chart.” DSM&T, 2020, www.dsmt.com/. Accessed 29 January 2020.

[12] “Aluminum 6061-T6; 6061-T651.” ASM Aerospace Specification Metals Inc., http://asm.matweb.com/search/SpecificMaterial.asp?bassnum=MA6061T6. Accessed 1 April 2020.

[13] “Loctite Ultra Gel Control Super Glue, 4-Gram Bottle (Pack of 2).” Amazon, https:// www.amazon.com/Loctite-Ultra-Control-Super-Glue/dp/B07VMC2YMT/ref=zg bs 256243011 2? encoding=UTF8&psc=1&refRID=0A8WRG21QDM57ES1XKD6.

[14] Xgentec Jason, “Linear Rail 400mm x 8mm Rail.” GrabCAD, https://grabcad.com/ library/linear-rail-400mm-x-8mm-rail-1. Accessed 13 April 2020.

[15] “Technical Resources.” REV Robotics, http://www.revrobotics.com/resources/.

[16] “15mm Linear Motion Kit v2.” REV Robotics, https://revrobotics.com/rev-45-1507/.

[17] “15mm Linear Motion System v2 (Guide).” REV Robotics, 2018, https://revrobotics.com/content/docs/15mmLinearMotion-Guide.pdf.

[18] “Velcro Brand All Purpose Strap.” Velcro, https://www.velcro.com/products/ties-andstraps/900600 all-purpose strap/?shape=strap&size=18in-x-1in&color=black.

[19] “FCHUB-6S w/ Current Sensor 184A, BEC 5V&10V, 6S Max (Manual).” MATEK Systems, https://www.mateksys.com/downloads/FCHUB-6S manual.pdf.

[20] Armin Ronacher. “Flask: A simple framework for building complex web applications,” PyPI. https://pypi.org/project/Flask/.

[21] “Robot Operating System,” Open Source Robotics Foundation. https://ros.org.

[22] “Gazebo - Robot Simulation Made Easy.” Open Source Robotics Foundation, https://www.gazebosim.org.

[23] “Aluminum 7005-T6, 7005-T63, and 7005-T6351.” MatWeb, http://www.matweb.com/search/ datasheet.aspx MatGUID=34c308934f7a4be589a80ecbee94406e&ckck=1.

[24] “Aluminum 6063-T6.” ASM Aerospace Specification Metals Inc., https://asm.matweb.com/ search/SpecificMaterial.asp?bassnum=MA6063T6.

[25] “Delrin (Acetal Homopolymer)” San Diego Plastics Inc., http://www.sdplastics.com/delrin/ delrin[1].pdf.

[26] Green, David W., Winandy, Jerrold E., and David E. Kretschmann. “Chapter 4: Mechanical Properties of Wood.” Wood Handbook. https://www.fpl.fs.fed.us/documents/ fplgtr/fplgtr113 /ch04.pdf.

[27] Dieter, George E. and Linda C. Schmidt. “Engineering Design, 4th edition.” The McGraw-Hill Companies, Inc., 2009.

Appendices

A Bill of Materials (BOM)

Product	Manufacturer	Part Number	Quantity	Unit Cost	Total Cost
LIFTING AND CLIMBING MECHANISM					
TETRIX Flat Bracket	Pitsco	W39061	4	$6.95	$27.80
TETRIX C-Channel	Pitsco	W39066	4	$12.95	$51.80
ABS Filament	MatterHackers	-	1	$19.99	$19.99
Aluminum Tube	McMaster-Carr	7237K18	1	$1.25	$1.25
Brass Bushing	McMaster-Carr	5448T1	8	$3.38	$27.04
Shaft 6mm	Gobilda	21100-0006-0150	4	$0.89	$3.56
Small Gear	RevRobotics	REV-41-1364	1	$4.00	$4.00
LM8UU Bearing	Adafruit	1181	2	$2.95	$5.90
Screw Rail	ZYLtech	-	2	$8.95	$17.90
8mm Rod	Amazon	PDTech: 30113901	4	$6.50	$26.00
SC8LUU Double Bearing Mount	Ebay	-	4	$13.99	$55.96
Extrusion L Beam Attachment	RevRobotics	REV-41-1709	2	$1.50	$3.00
CUBE LATCH MECHANISM					
Extrusions	RevRobotics	REV-41-1017	4	$10.00	$40.00
Linear Motion Kit 15mm - V2	RevRobotics	REV-45-1507	1	$12.00	$12.00
3.2mm Shaft	RevRobotics	REV-41-1347	2	$8.00	$16.00
Small Sprocket	RevRobotics	REV-41-1338	2	$4.00	$8.00
ROTATING PLATE MECHANISM					
Servo Mount L-Beam	RevRobotics	REV-41-1707	1	$2.75	$2.75
Rubber Stopper	Amazon	-	1	$9.36	$9.36
SCREWS AND NUTS					
M3 T-Slot Screw: 8mm L	RevRobotics	REV-41-1167	1	$16.00	$16.00
M3 Nut	RevRobotics	REV-41-1126	1	$4.00	$4.00
M3.7 Button Head Screw: 10mm L	RevRobotics	REV-41-1364	2	$4.00	$8.00
M4 Hex Head Screw: 14mm L	McMaster-Carr	92095A193	1	$8.95	$8.95
M3 Hex Head Bolt: 20mm L	RevRobotics	REV-41-1124	1	$10.00	$10.00
Long Hex Screw	McMaster-Carr	92245A541	1	$6.53	$6.53
Servo Screw					
MOTORS AND SERVOS					
Standard Servo	RevRobotics	REV-41-1097	5	$24.00	$120.00
Servo Bracket Outside Channel	RevRobotics	REV-41-1682	1	$5.00	$5.00
Servo Bracket Inside Channel	RevRobotics	REV-41-1681	1	$5.00	$5.00
Aluminum Servo Horn	RevRobotics	REV-41-1363	3	$4.00	$12.00
Servo Gear Adapter	RevRobotics	REV-41-1364	2	$4.00	$8.00
Nema 17 Stepper Motor	Adafruit	324	2	$14	$28
REV Core Hex Motor	RevRobotics	REV-41-1300	1	$21.00	$21.00
ELECTRONICS					
Raspberry Pi Zero W	Adafruit	3400	1	$10	$10
STM32 Nucleo	Amazon	-	1	$24.95	$24.95
MATEKSYS FCHUB-6S	Amazon	-	1	$10.99	$10.99
MATERIALS					
Balsa Block	Specialized Balsa LLC	1x1x48in	4	$5.23	$20.92
Aluminum sheet	Midwest Steel and Aluminum	6061 Aluminum	2		$8.43
				TOTAL	$660.08

B

Part Drawings

B1 Lifting and Climbing Mechanism

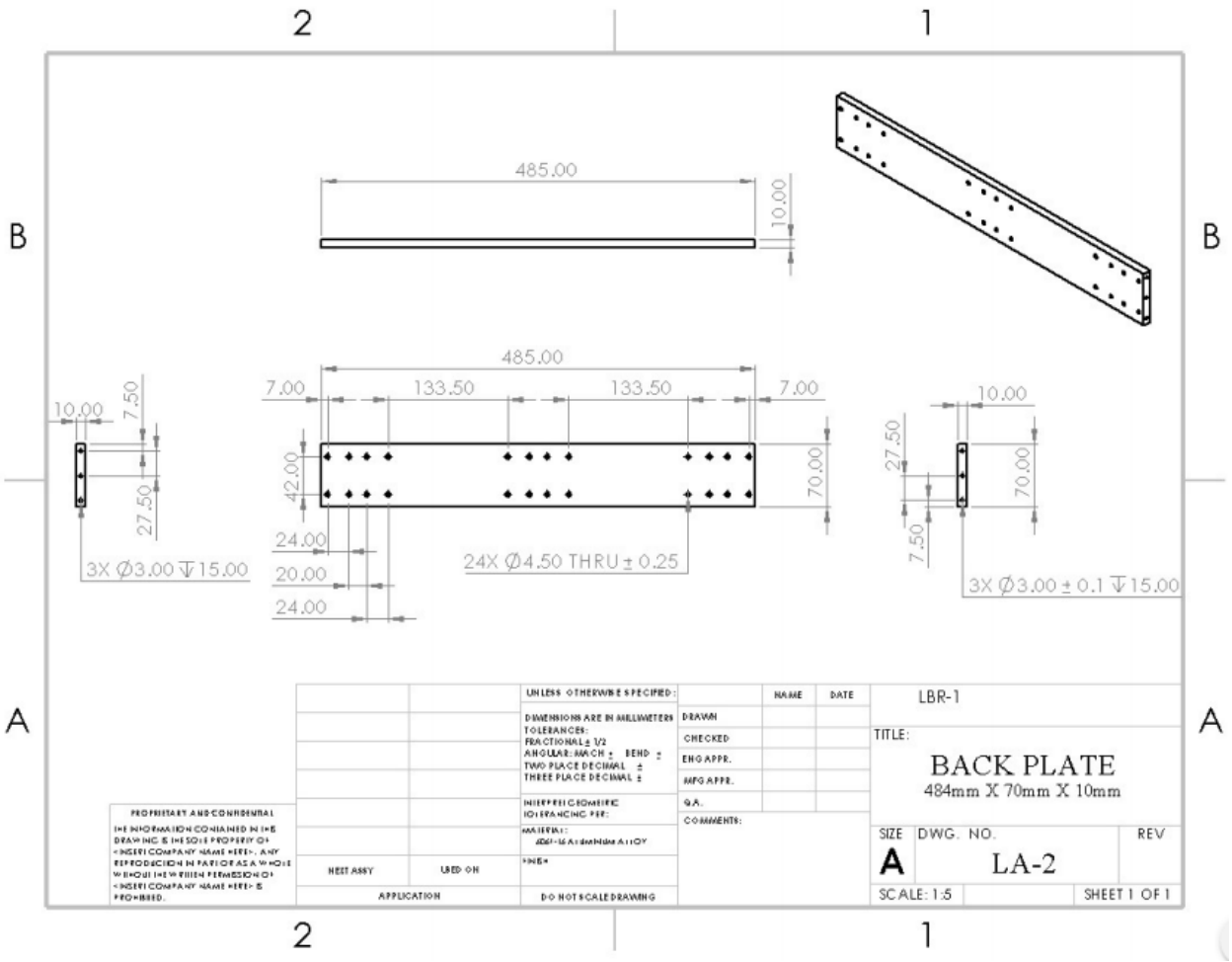


Figure B1.1: Part drawing of the back plate of the linear actuator

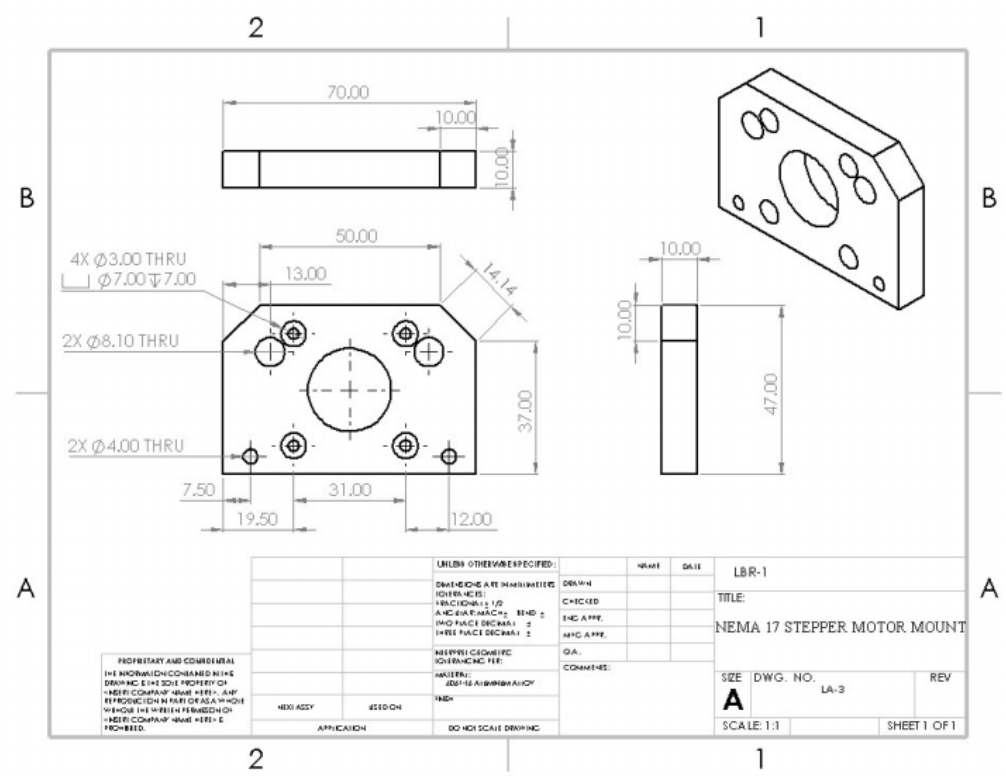


Figure B1.2: Part drawing of the Nema 17 stepper motor mount for the linear actuator.

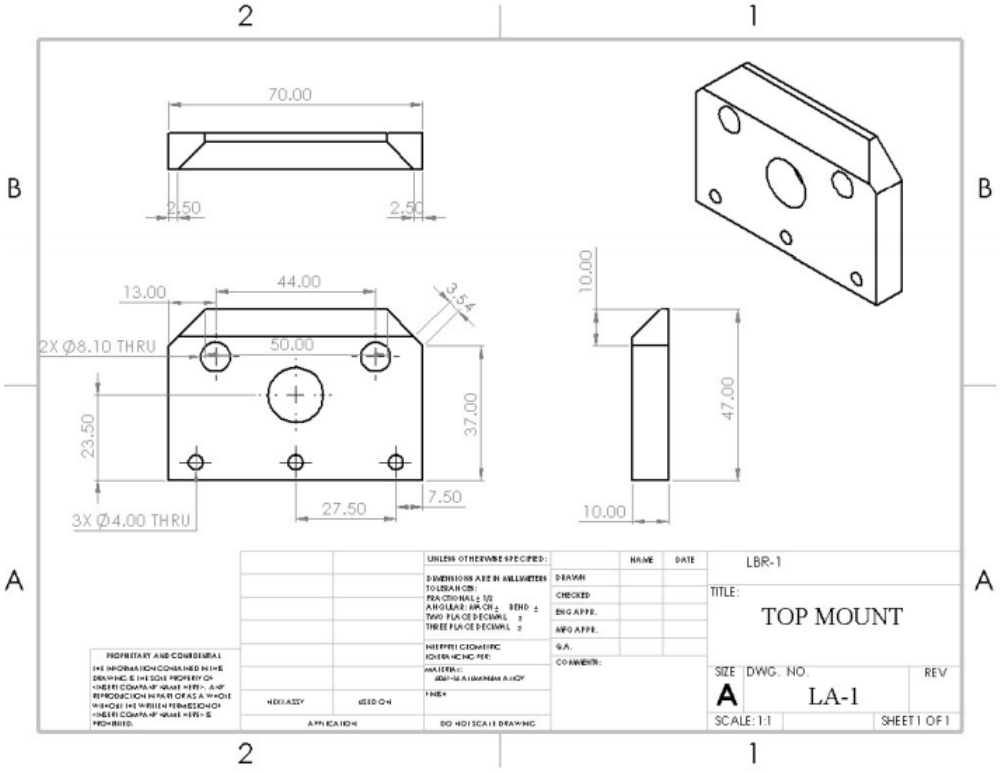


Figure B1.3 Part drawing of the top mount for the linear actuator.

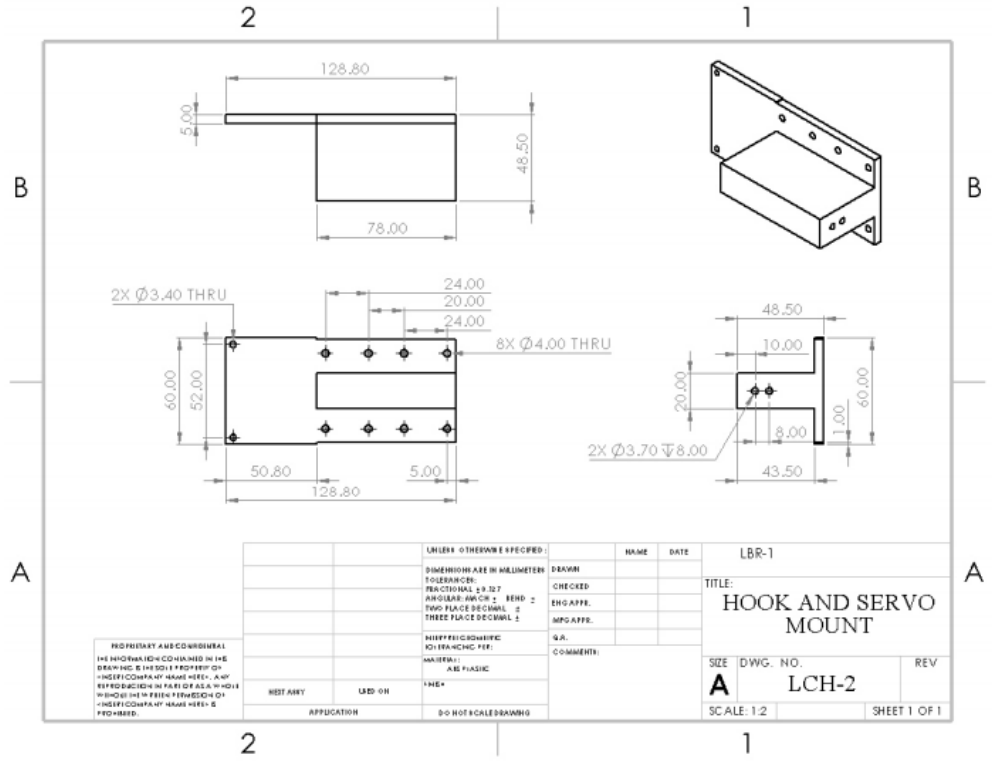


Figure B1.4 Part drawing of the hook and servo mount for the dual hook assembly.

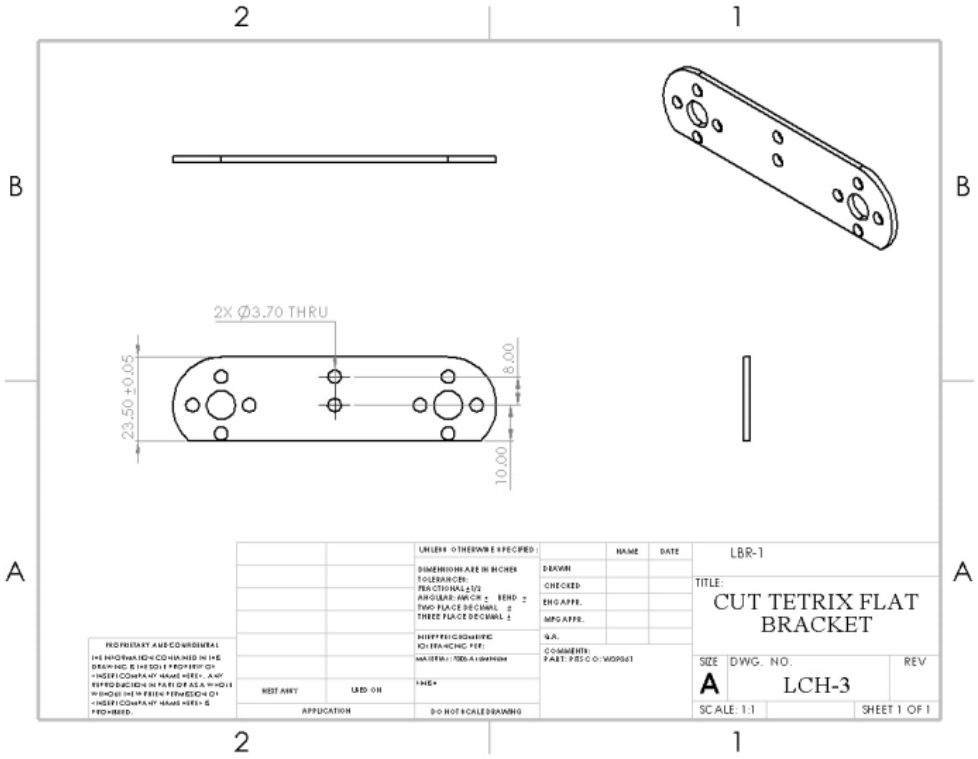


Figure B1.5: Part drawing of the modified TETRIX flat bracket for the dual hook assembly

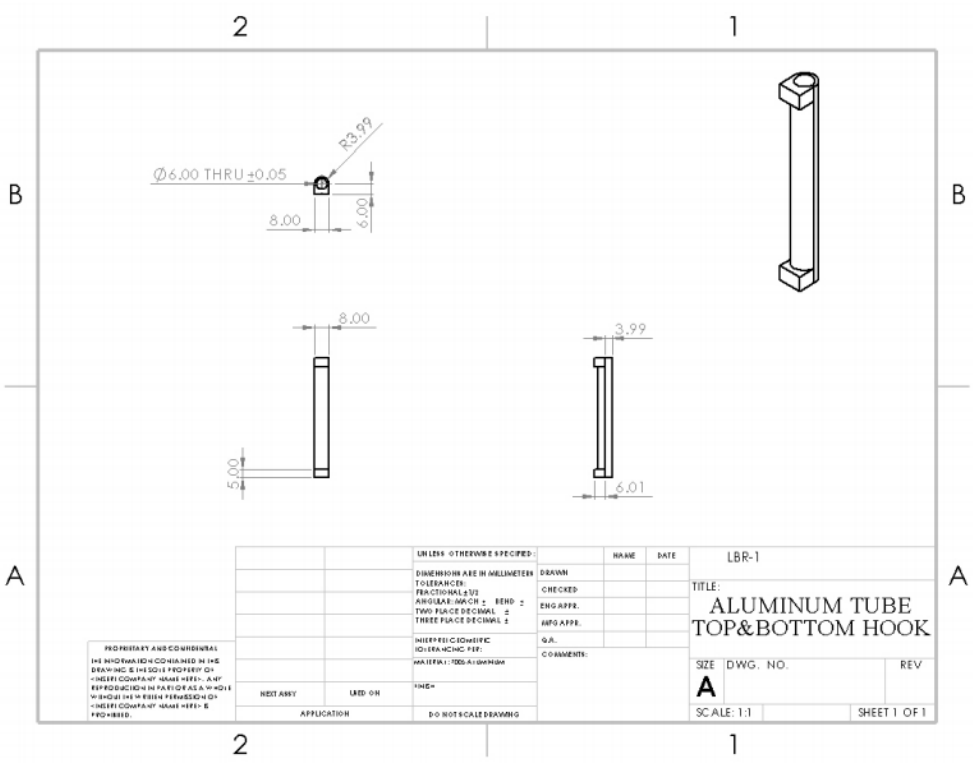


Figure B1.6: Part drawing of the aluminum tube for the lifting and climbing hooks

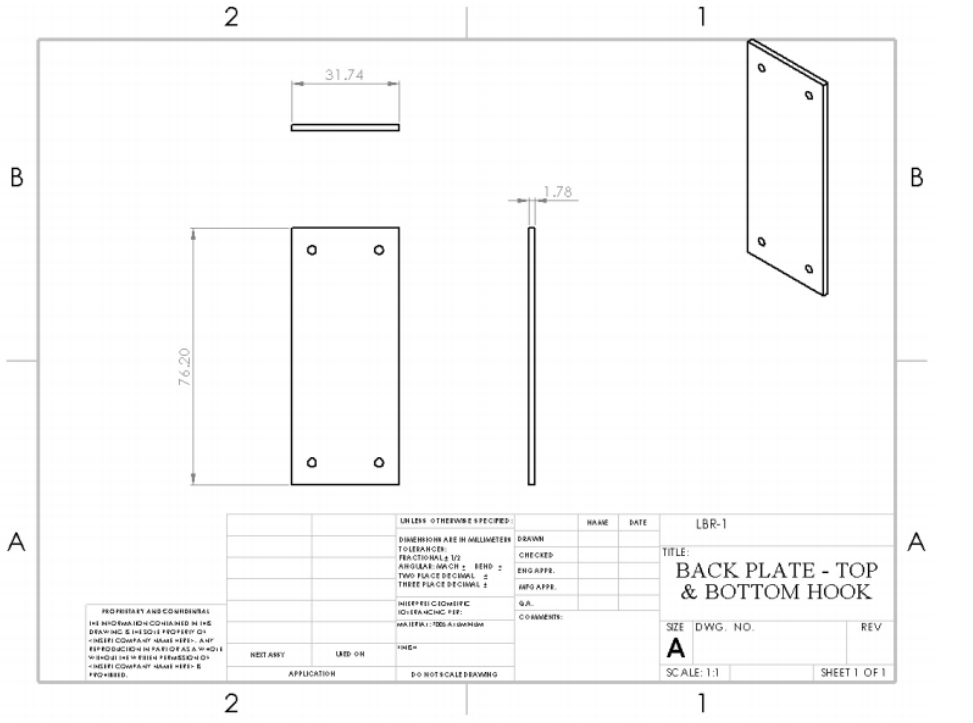


Figure B1.7: Part drawing of the back plate for the lifting and climbing hooks.

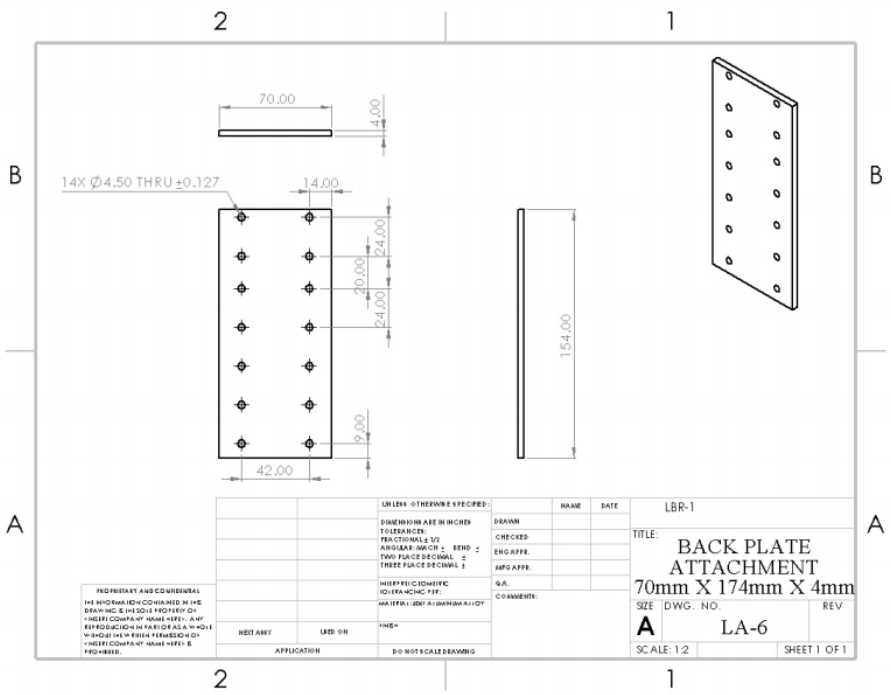


Figure B1.8: Part drawing of the back plate attachment that holds the linear actuator and extrusion together.

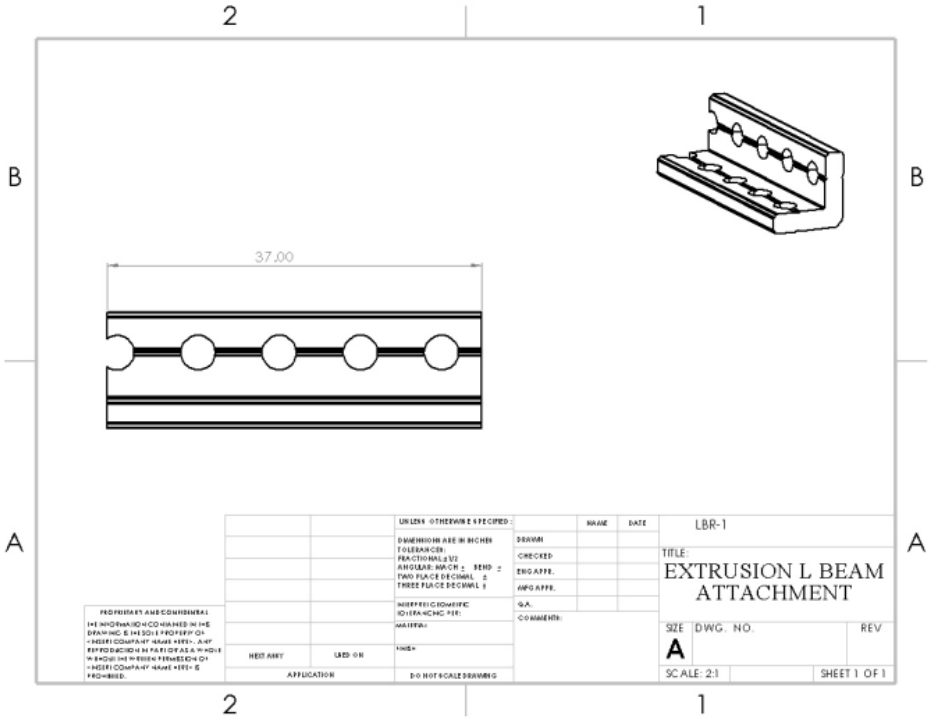


Figure B1.9: Part drawing of the modified L-beam connecting the lifting and climbing mechanism to the base plate.

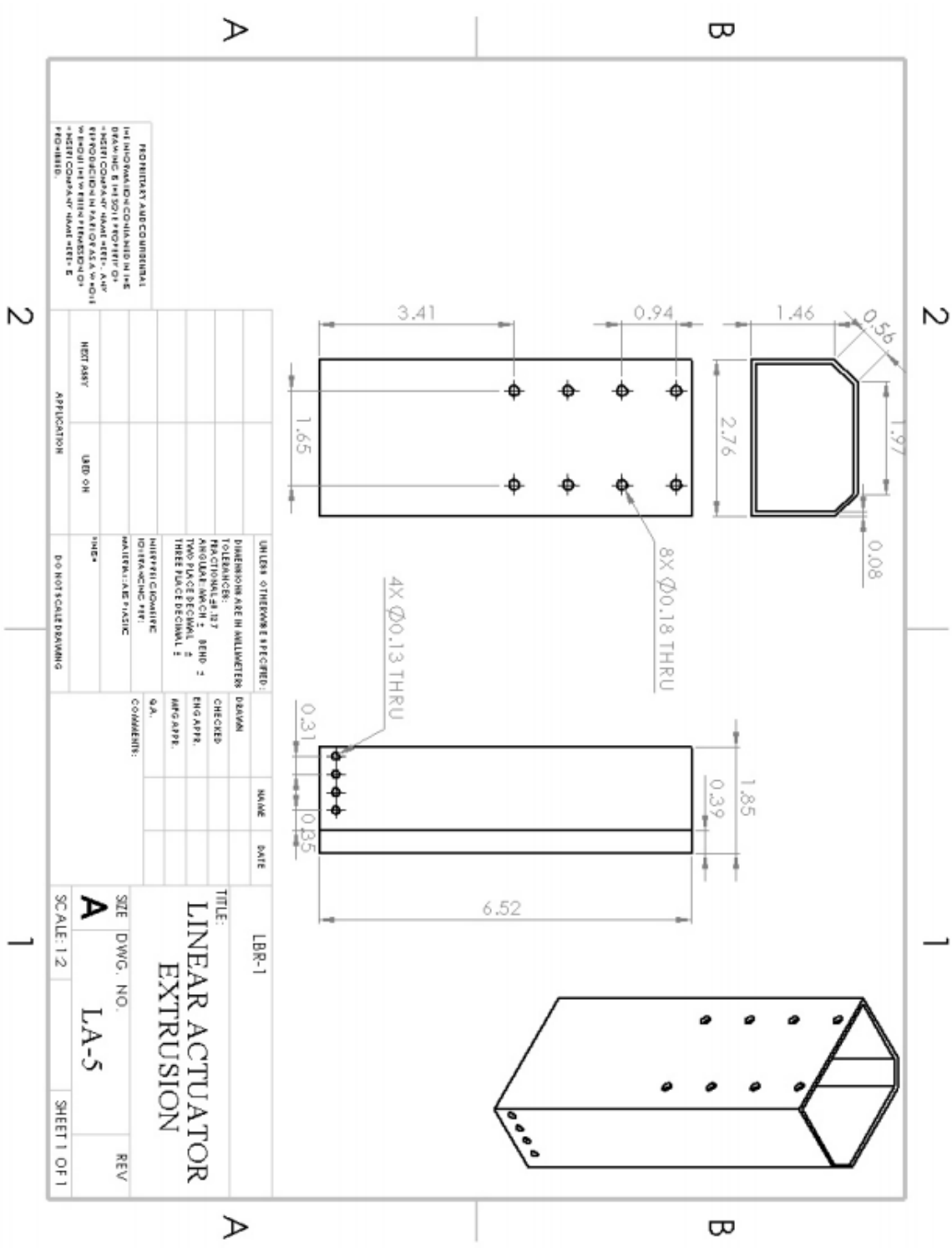


Figure B1.10: Part drawing of a custom extrusion to elevate linear actuator to desired height.

B2 Cube Latch Mechanism

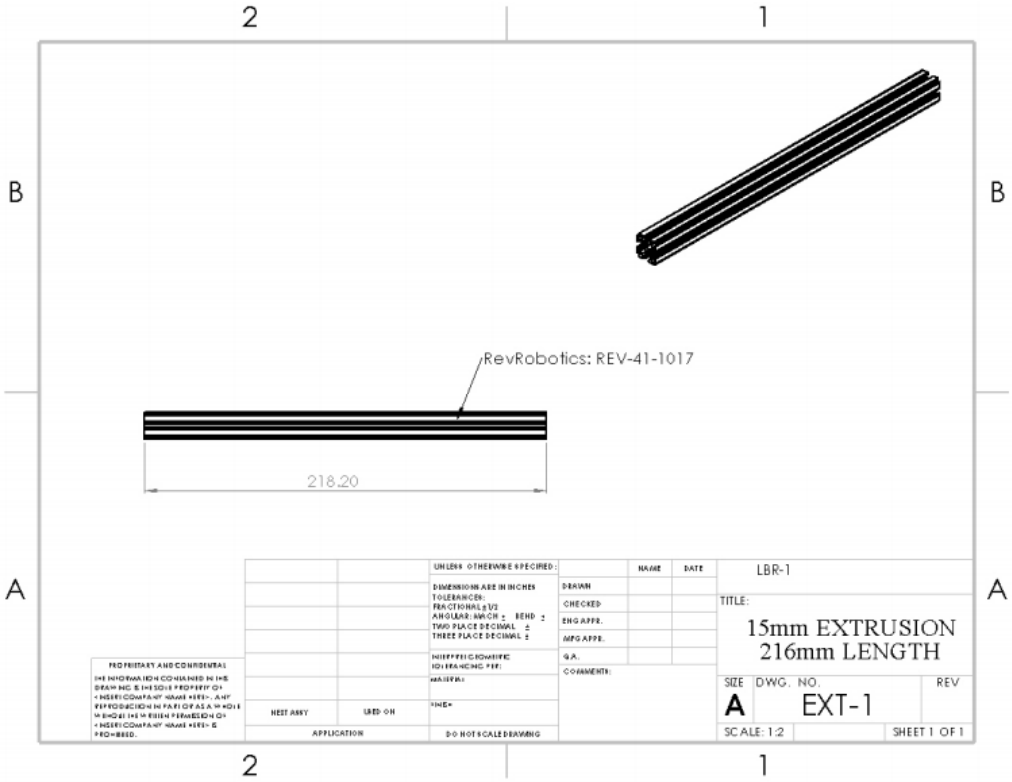


Figure B2.1: Part drawing of the 15mm extrusion to attach linear slide assembly to base plate.

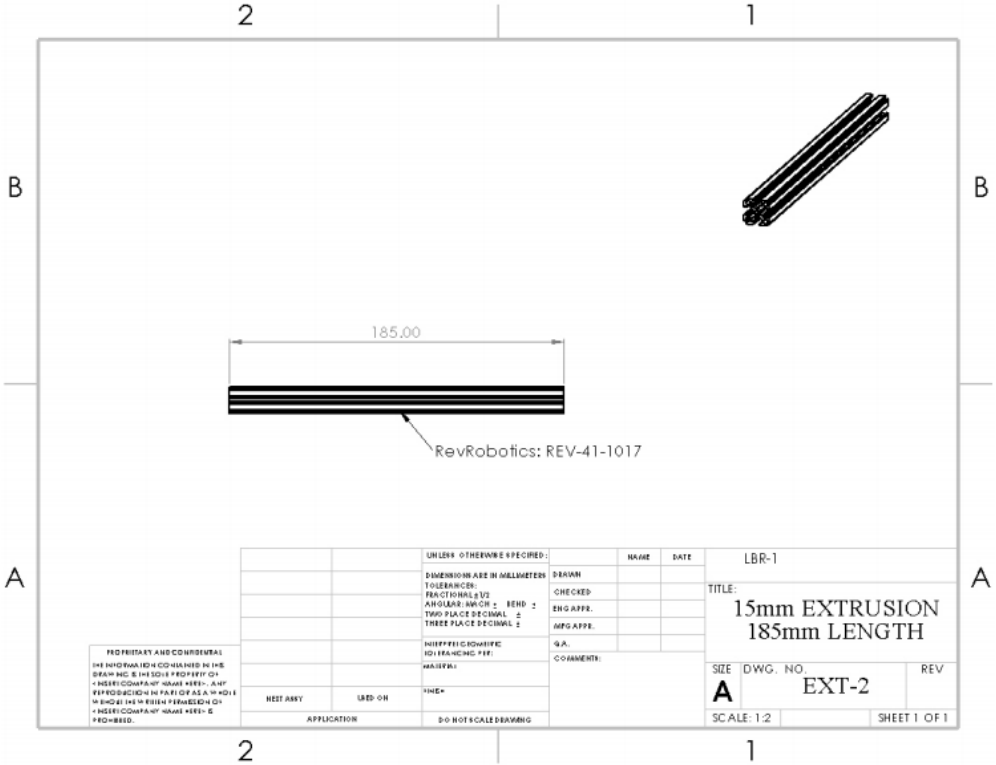


Figure B2.2: Part drawing of the 15mm extrusion for the linear slide assembly.

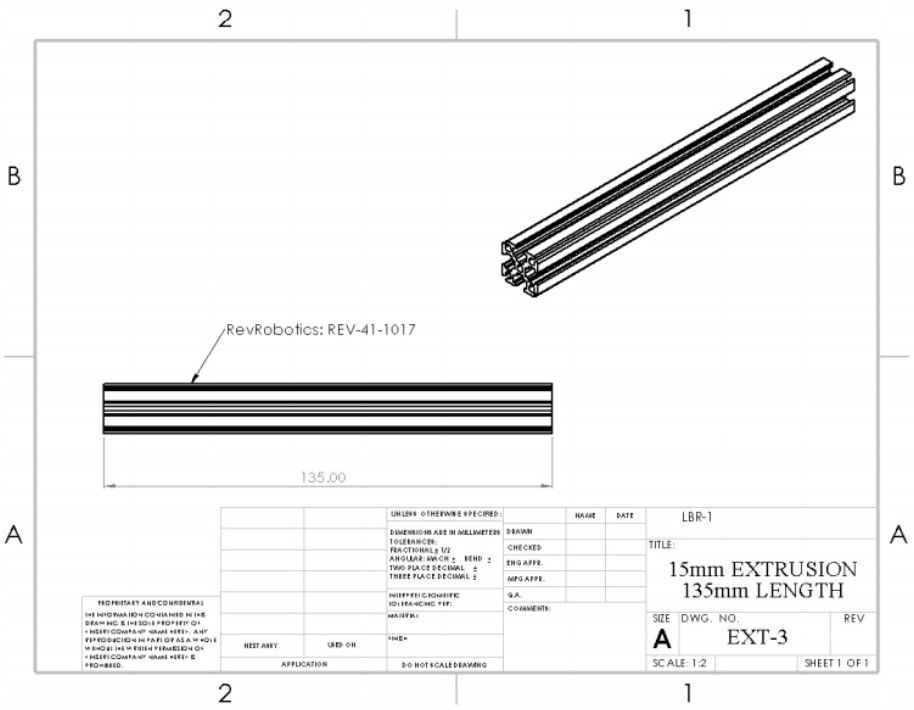


Figure B2.3: Part drawing of the 15mm extrusion to hold the motor and motor mount for the linear slide assembly.

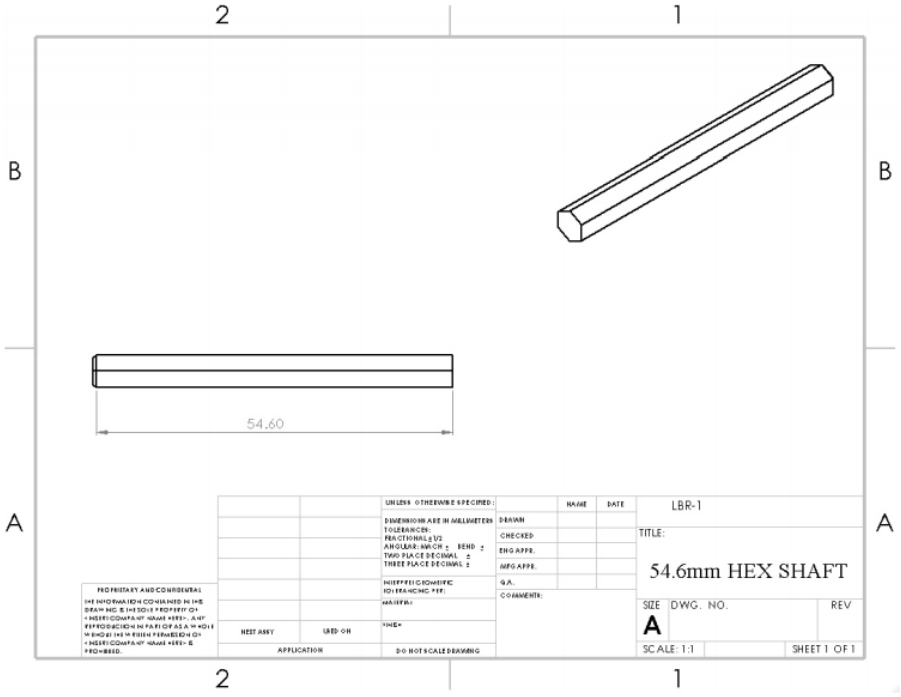


Figure B2.4: Part drawing of a hex shaft for the chain-and-sprocket system on the linear slide assembly

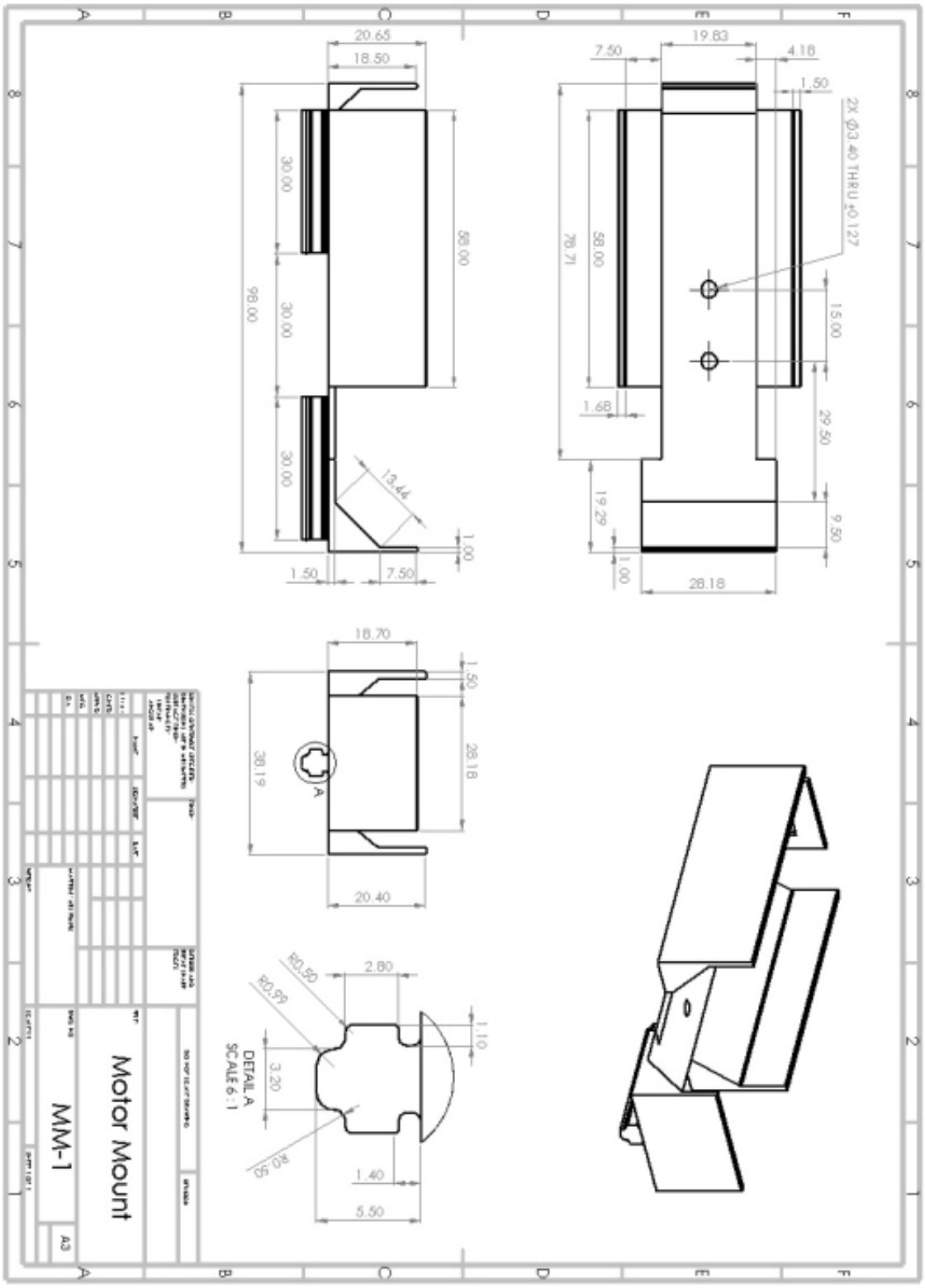


Figure B2.5: Part drawing of a custom motor mount for the linear slide assembly.

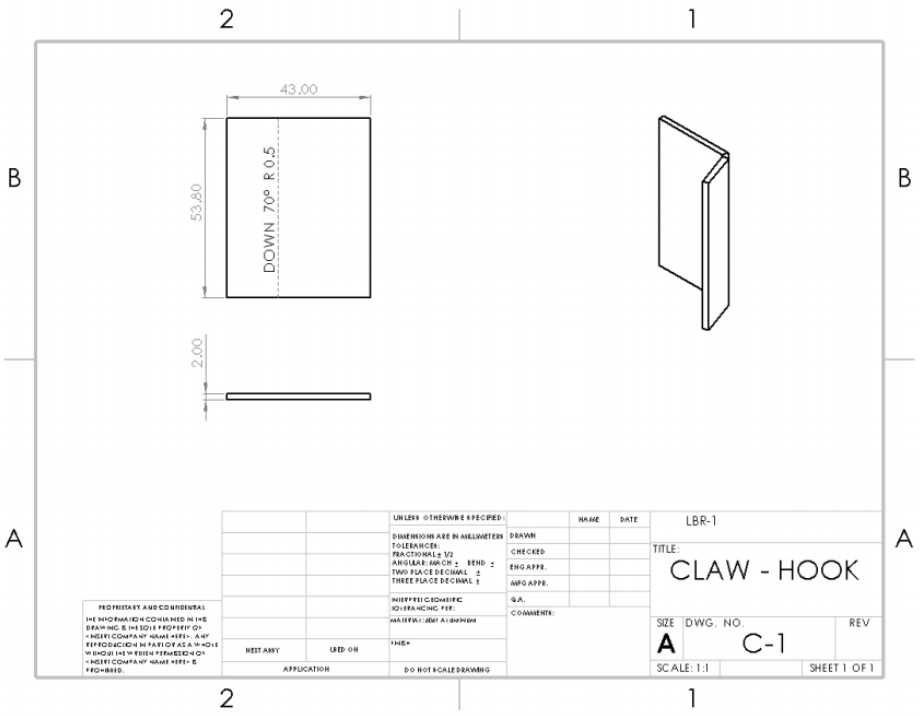


Figure B2.6: Part drawing of the claw's hook for the cube latch assembly

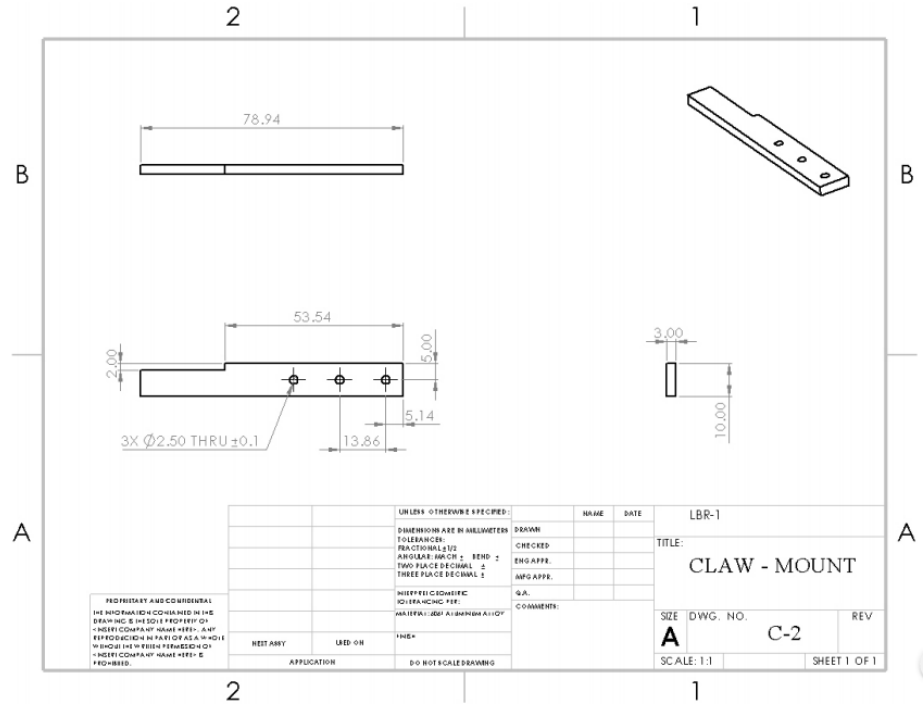


Figure B2.7: Part drawing of the mount of the claw's hook to be fastened onto the servo.

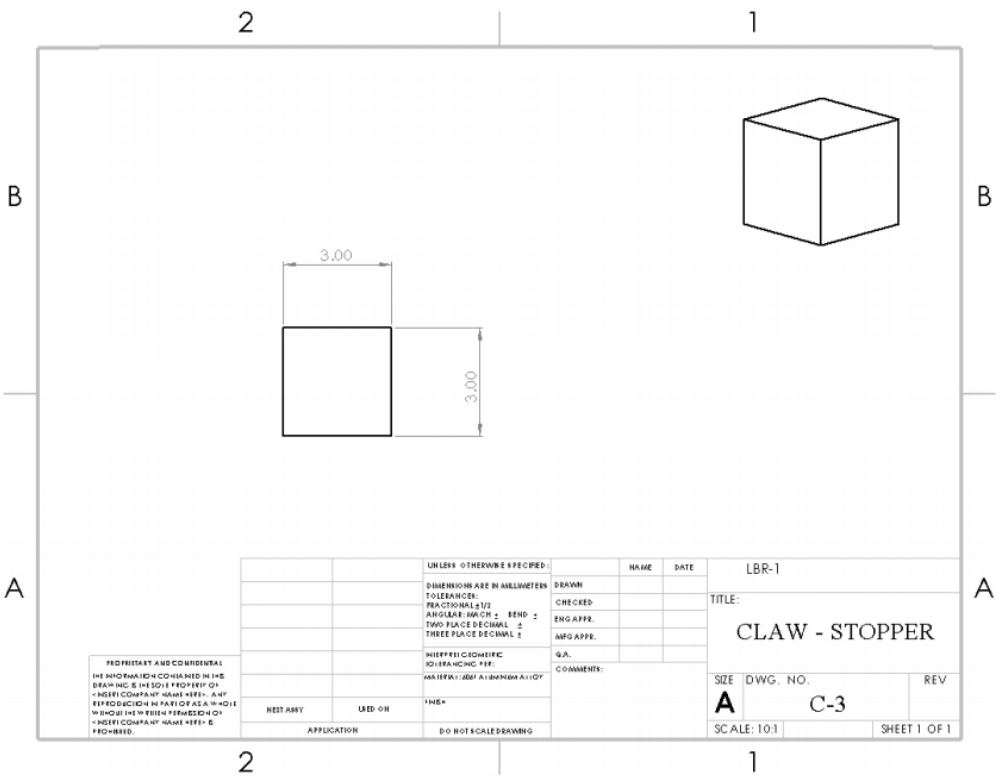


Figure B2.8: Part drawing of the stopper to constrain motion of cube element within the claw.

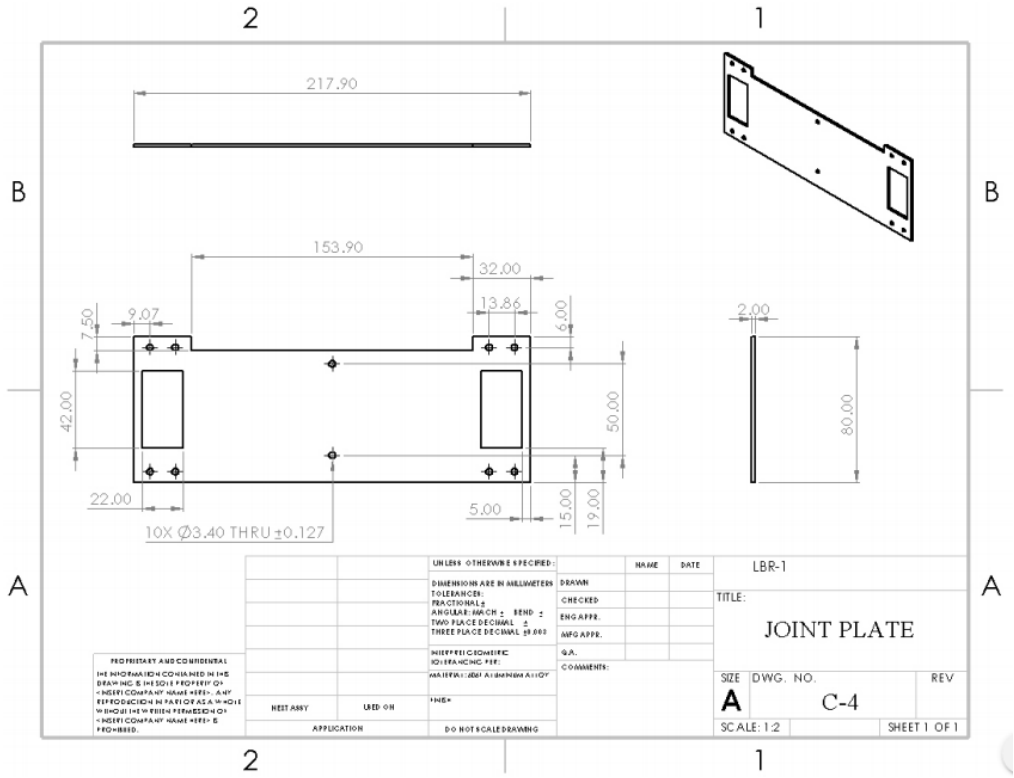


Figure B2.9: Part drawing of the joint plate of the cube latch assembly.

B3 Rotating Plate Mechanism

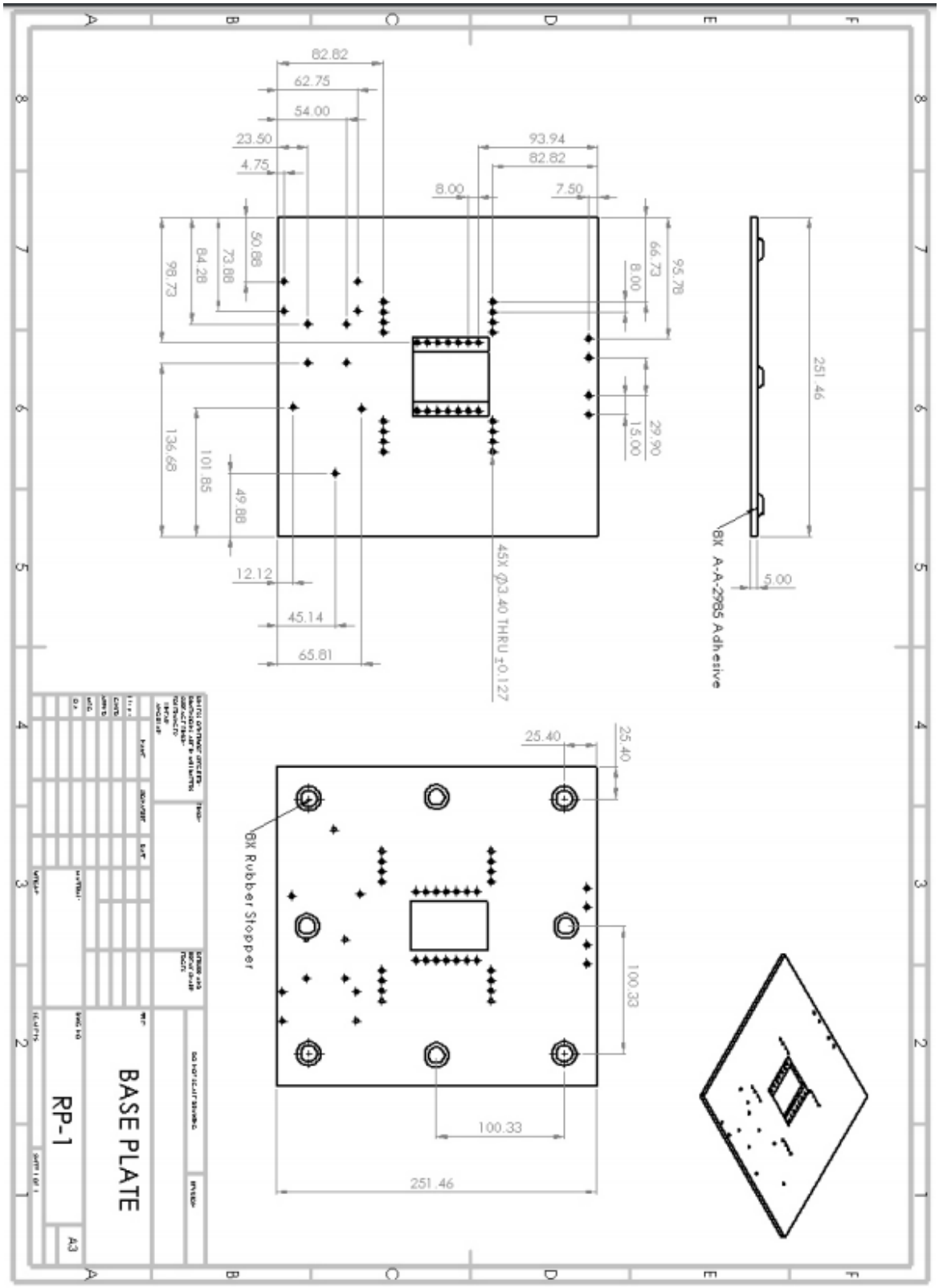


Figure B3.1: Part drawing of the base plate.

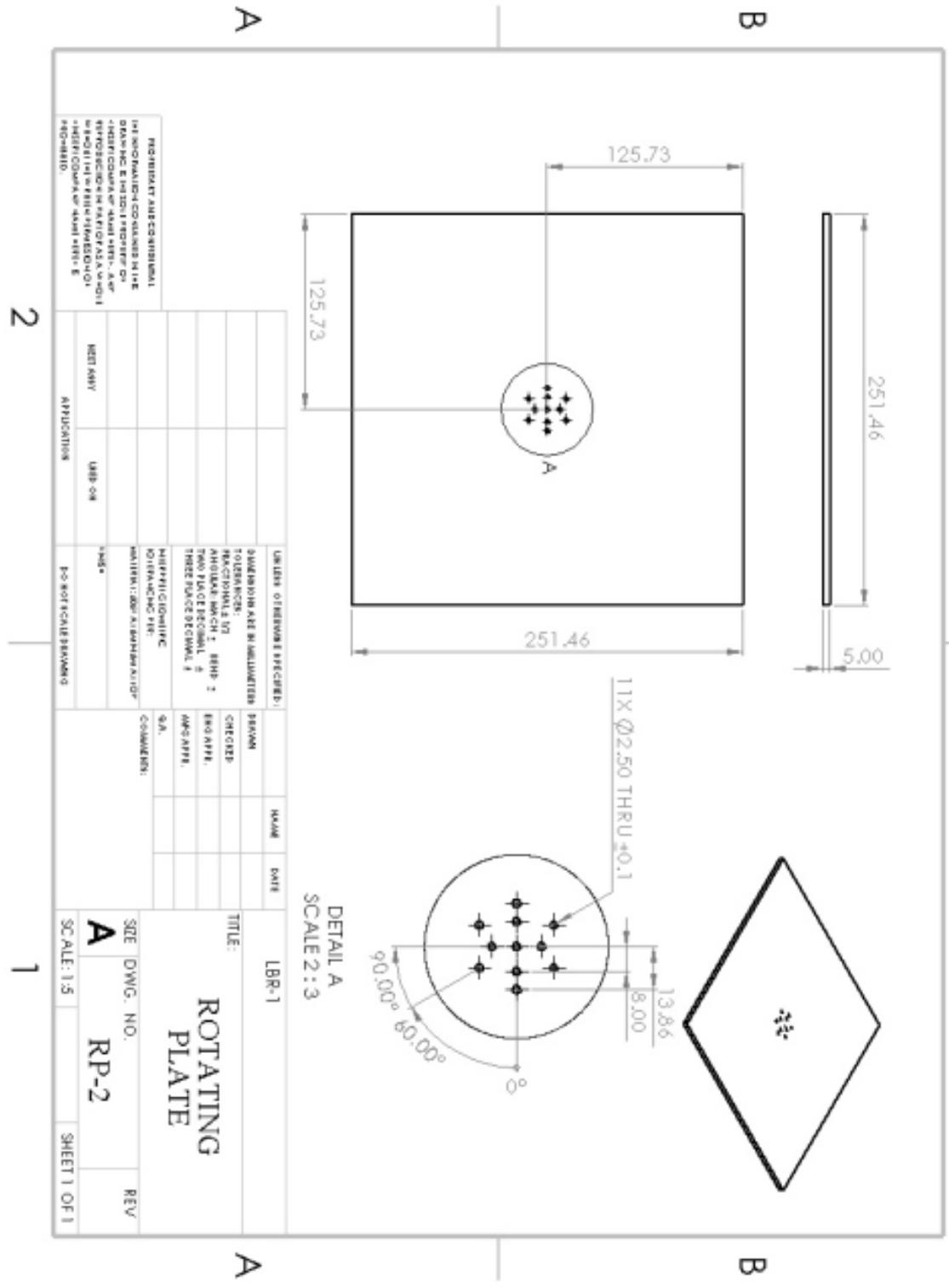


Figure B3.2: Part drawing of the rotating plate.

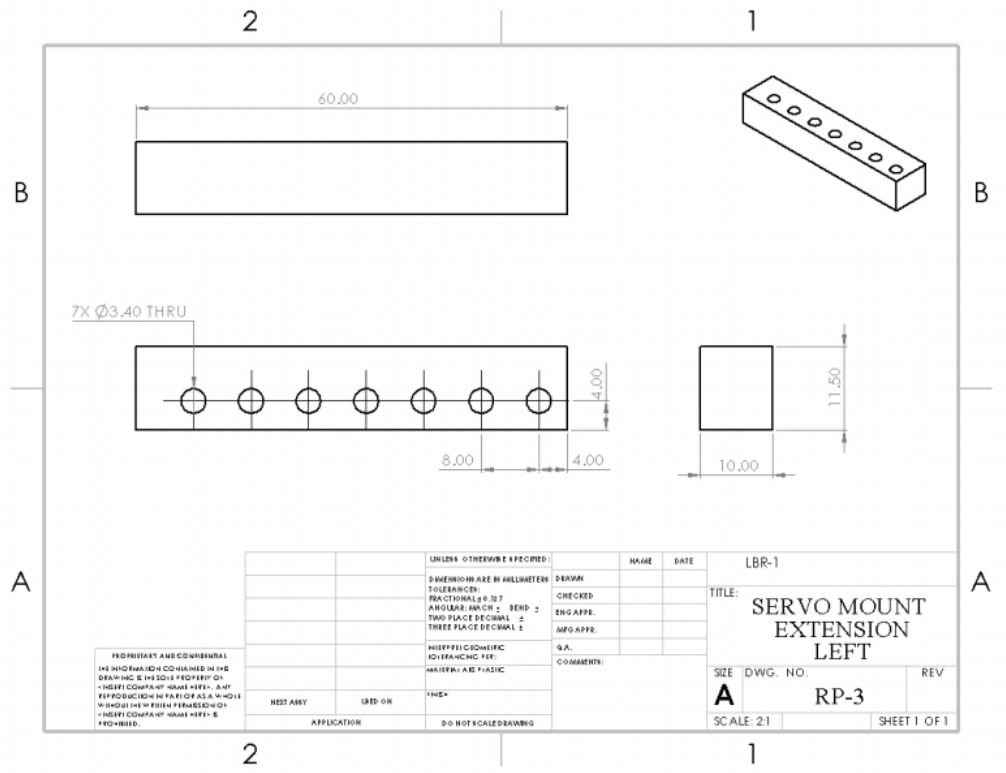


Figure B3.3: Part drawing of the servo mount extension (left) for the rotating plate mechanism.

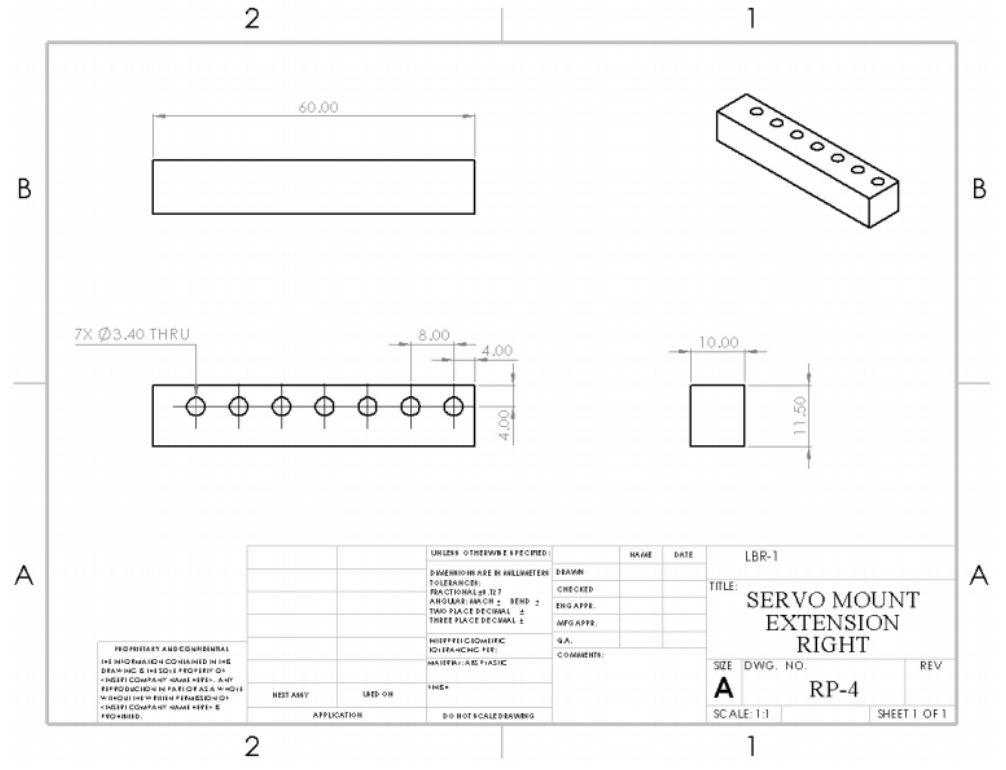


Figure B3.4: Part drawing of the servo mount extension (right) for the rotating plate mechanism.

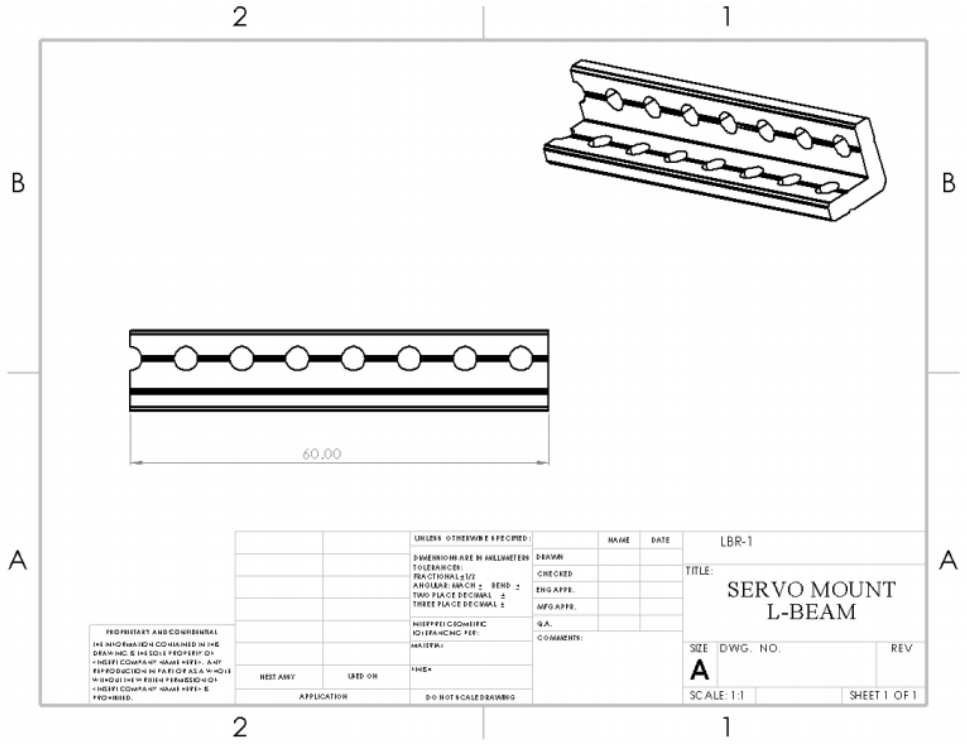


Figure B3.5: Part drawing of the modified L-beam to attach servo mount onto base plate.

B4 Cube

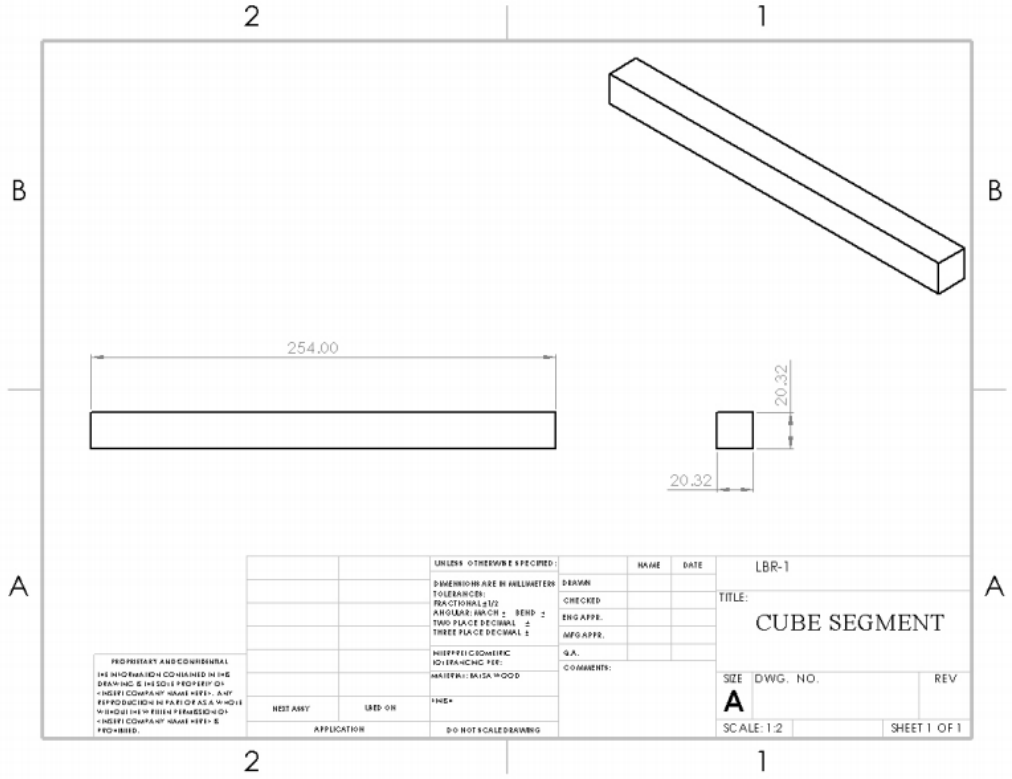


Figure B4.1: Part drawing of the horizontal and vertical segments of the cube element.

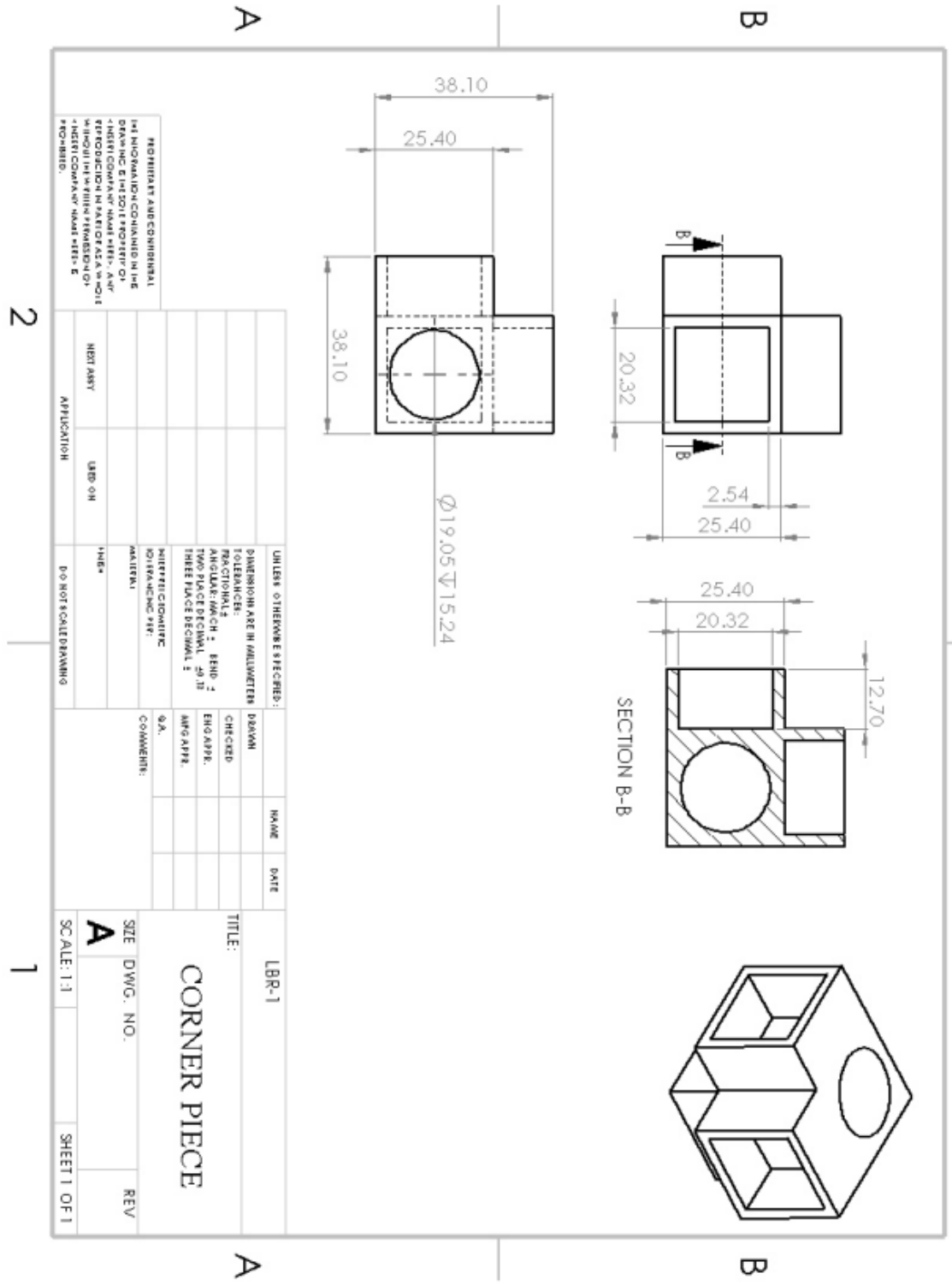


Figure B4.2: Part drawing of the corner piece of the cube element

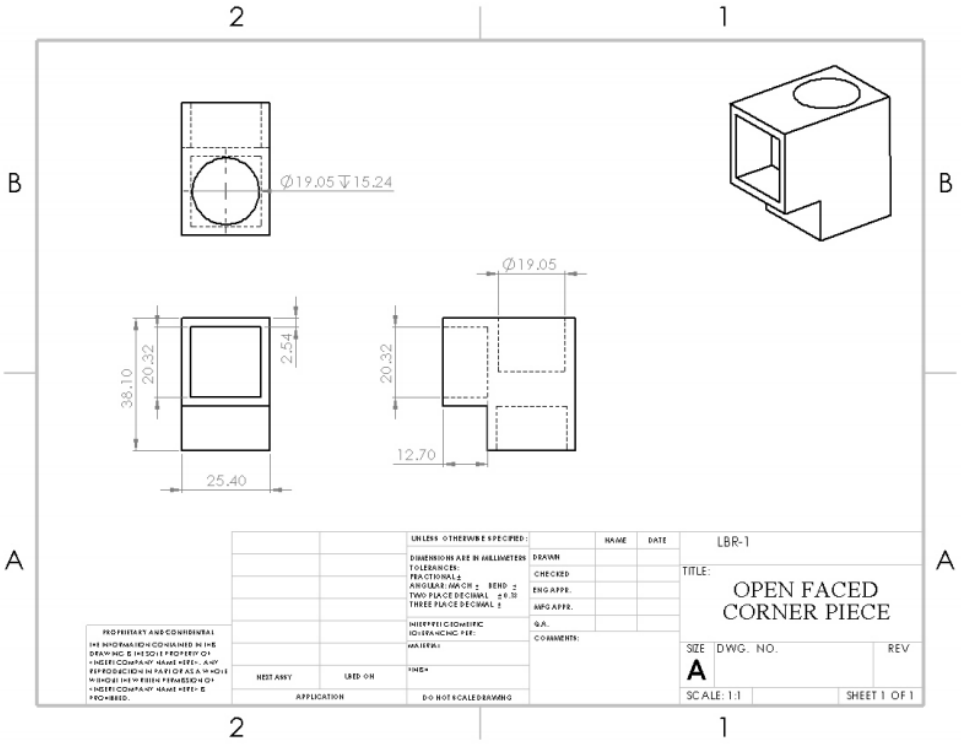


Figure B4.3: Part drawing of the open-faced corner piece of the cube element.

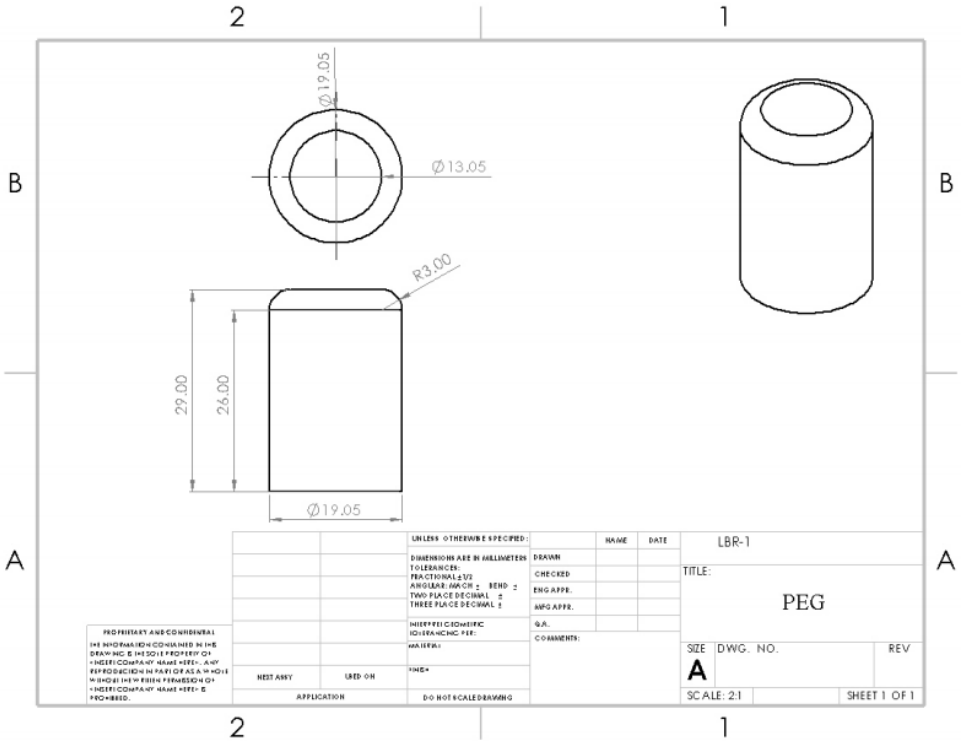


Figure B4.4: Part drawing of the peg of the cube element.

B5 Electronics Compartment

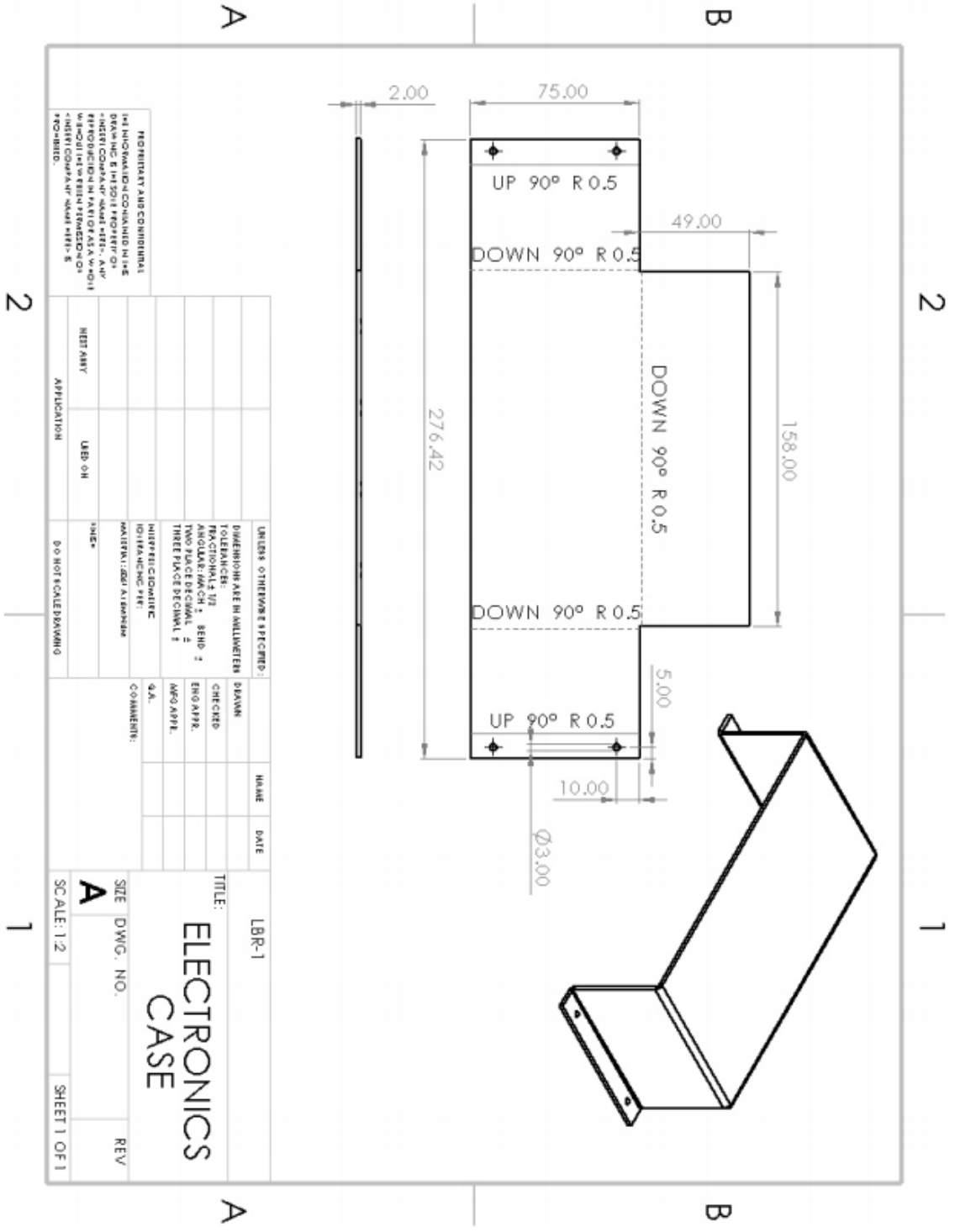


Figure B5.1: Part drawing of the electronics case.

C Assembly Drawings
C1 Sub-assemblies of Lifting and Climbing Mechanism

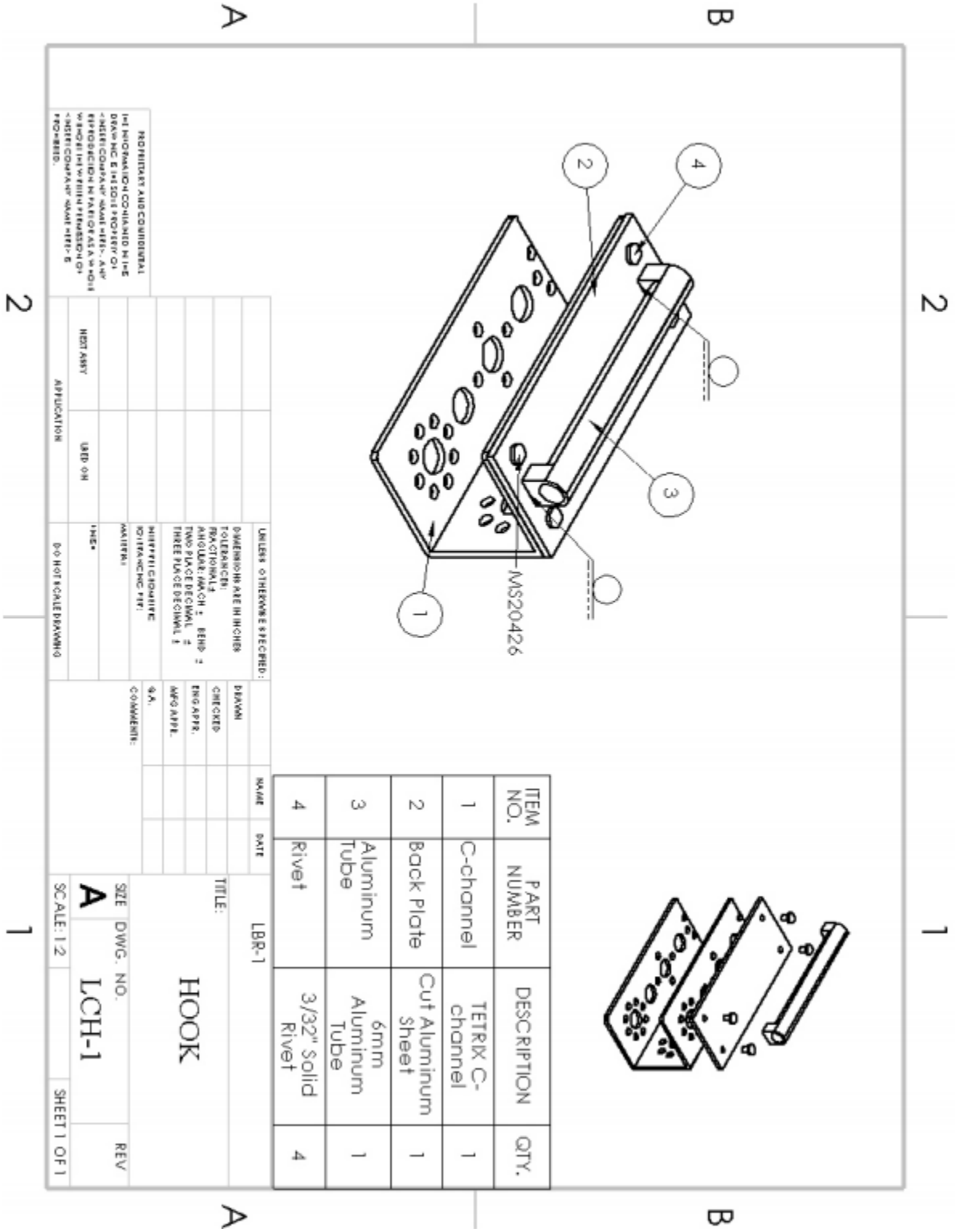


Figure C1.1: Assembly drawing of the lifting and climbing hook.

Figure C1.2: Assembly drawing of the servo assembly to control the lifting and climbing hook

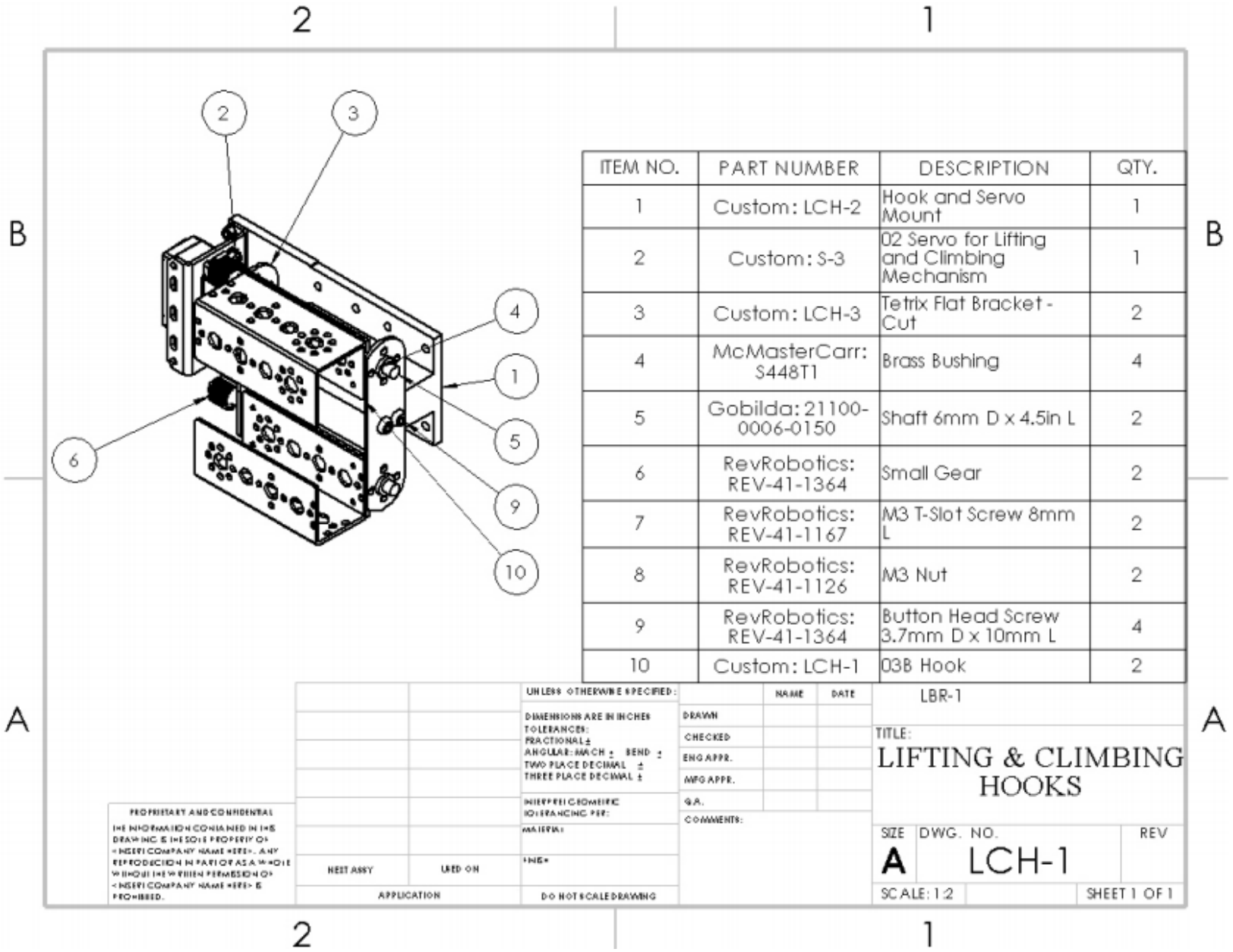
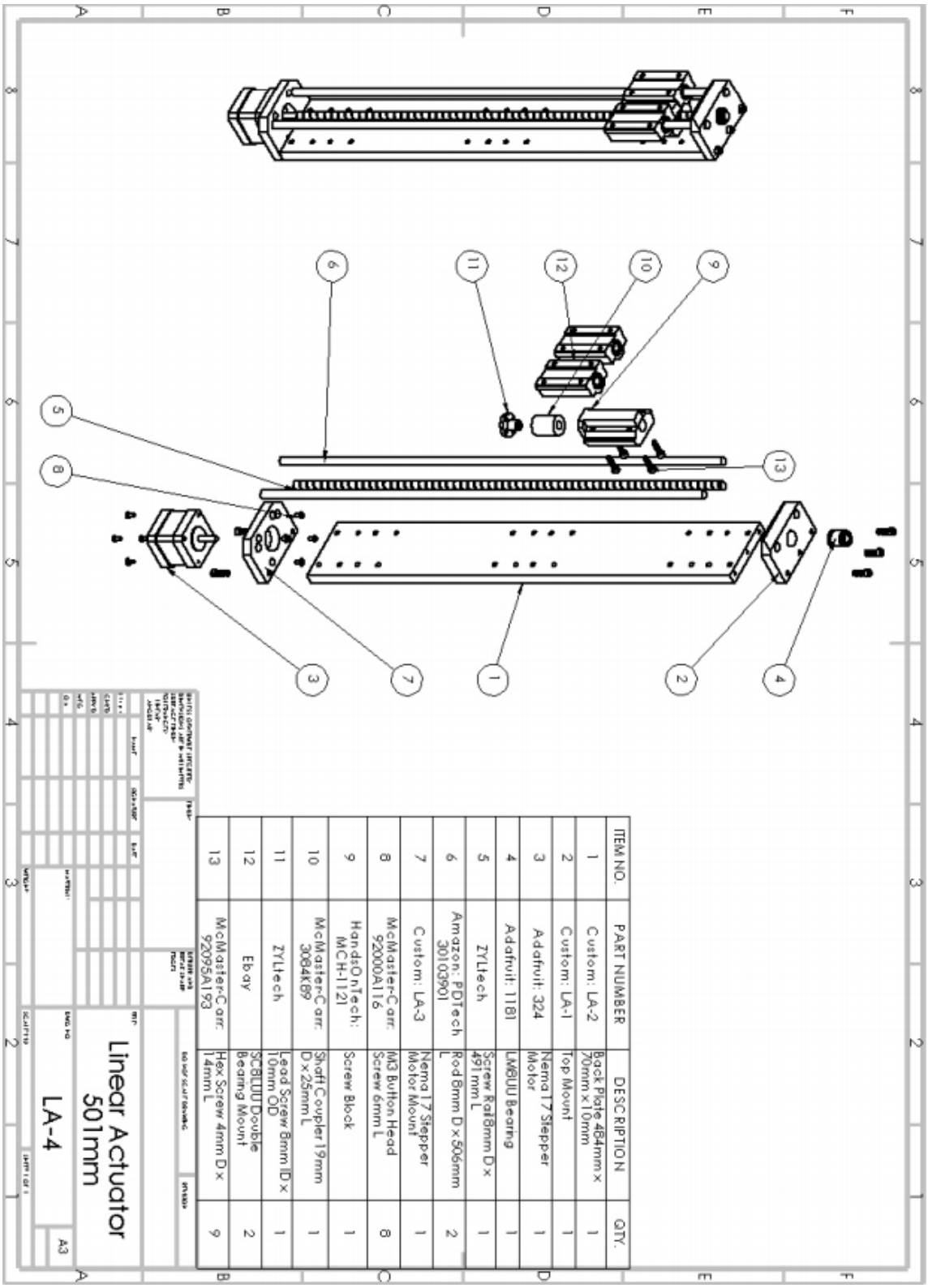


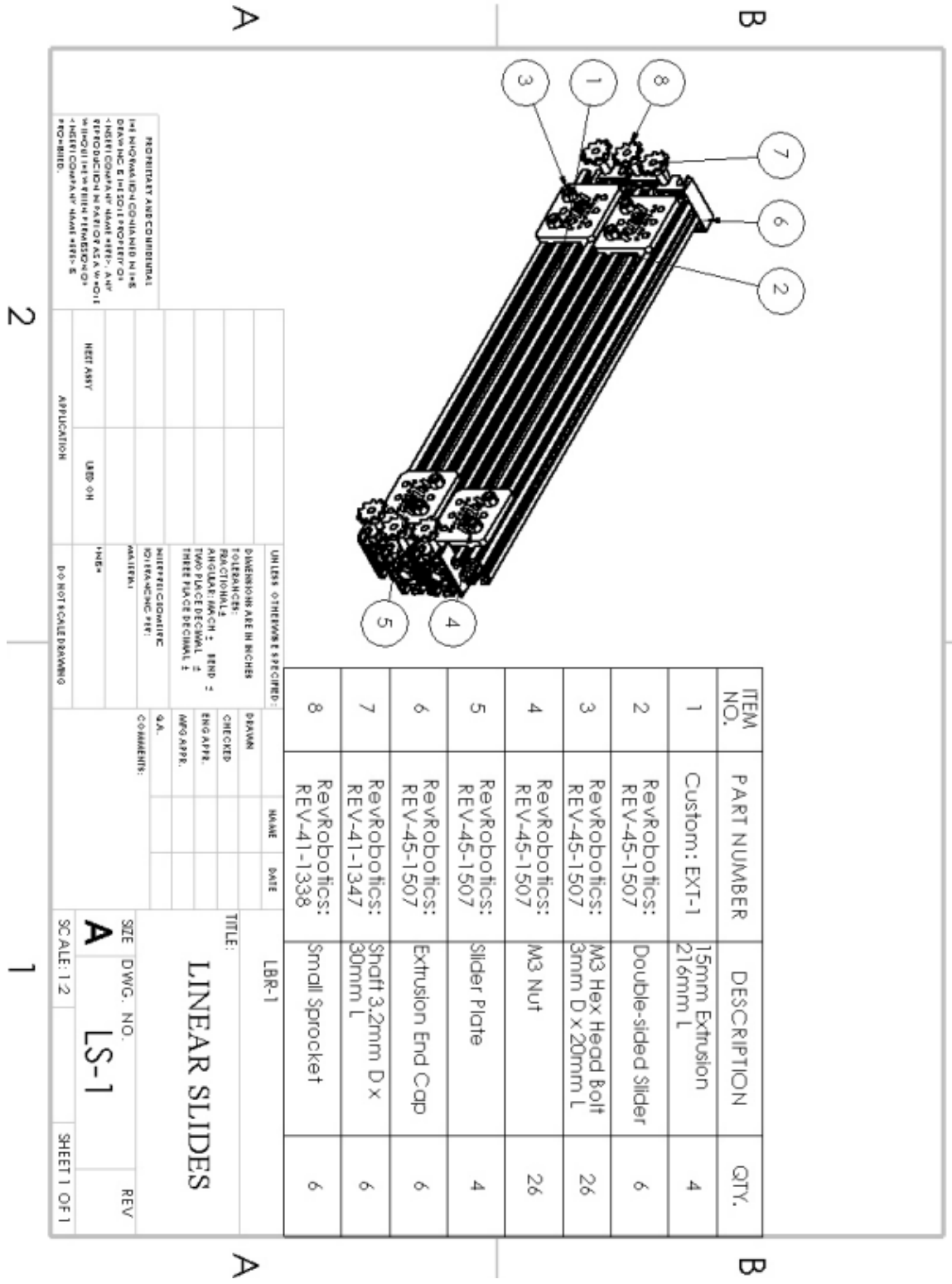
Figure C1.3: Assembly drawing of the dual hook assembly



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Custom: LA-2	Back Plate 484mm x 70mm x 10mm	1
2	Custom: LA-1	Top Mount	1
3	Adafruit: 324	Nema 17 Stepper Motor	1
4	Adafruit: 1181	LM8UU Bearing	1
5	ZYLtech	Screw Rod 8mm D x 491 mm L	1
6	Amazon: PDTech 30103901	Rod 8mm D x 506mm L	2
7	Custom: LA-3	Nema 17 Stepper Motor Mount	1
8	McMaster-Carr: 92000A116	M3 Button Head Screw 6mm L	8
9	HandsOnTech: MCH-1121	Screw Block	1
10	McMaster-Carr: 3084K89	Shaft Coupler 19mm D x 25mm L	1
11	ZYLtech	Lead Screw 8mm ID x 10mm OD	1
12	Ebay	SC8LUU Double Bearing Mount	2
13	McMaster-Carr: 92095A193	Hex Screw 4mm D x 14mm L	9

Figure C1.4: Assembly drawing of the linear actuator

C2 Sub-assemblies of Cube Latch Mechanism



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Custom: EXT-1	15mm Extrusion 216mm L	4
2	RevRobotics: REV-45-1507	Double-sided Slider	6
3	RevRobotics: REV-45-1507	M3 Hex Head Bolt 3mm D x 20mm L	26
4	RevRobotics: REV-45-1507	M3 Nut	26
5	RevRobotics: REV-45-1507	Slider Plate	4
6	RevRobotics: REV-45-1507	Extrusion End Cap	6
7	RevRobotics: REV-41-1347	Shaft 3.2mm D x 30mm L	6
8	RevRobotics: REV-41-1338	Small Sprocket	6

Figure C2.1: Assembly drawing of the linear slides.

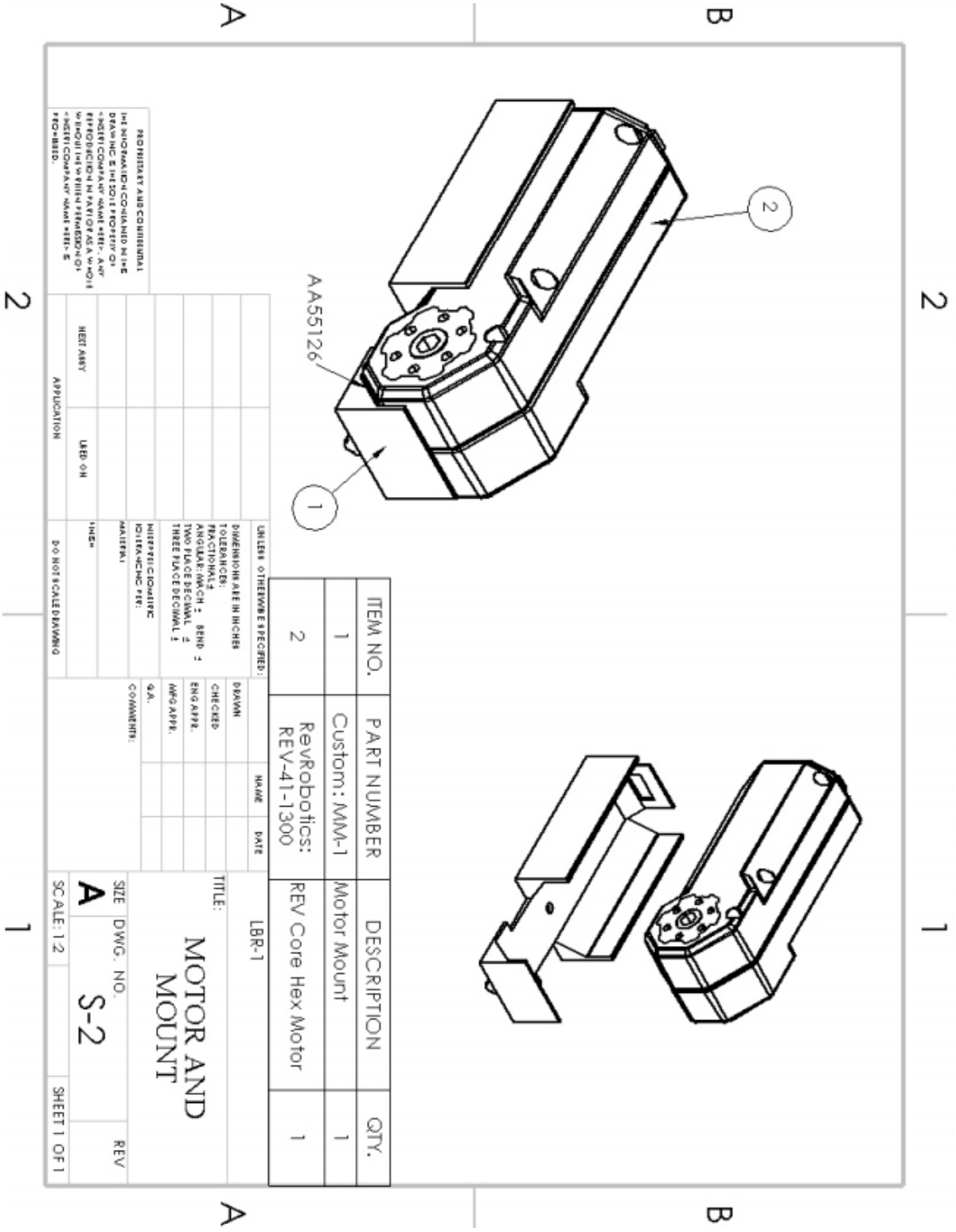
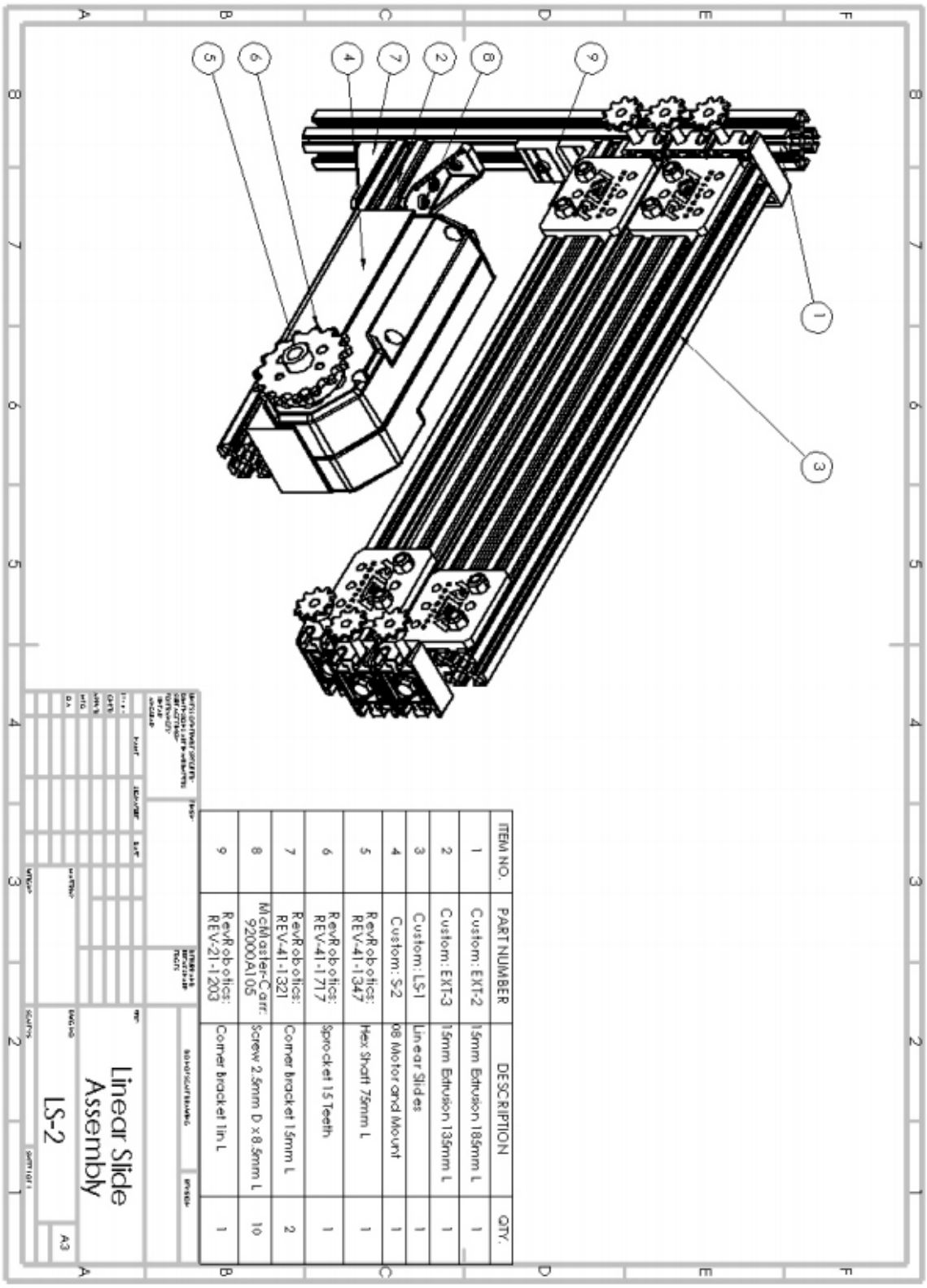


Figure C2.2: Assembly drawing of the motor and motor mount.



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Custom: EXT-2	15mm Extrusion 185mm L	1
2	Custom: EXT-3	15mm Extrusion 135mm L	1
3	Custom: LS-1	Linear Slides	1
4	Custom: S-2	08 Motor and Mount	1
5	RevRobotics: REV-41-1347	Hex Shaft 75mm L	1
6	RevRobotics: REV-41-1717	Sprocket 15 Teeth	1
7	RevRobotics: REV-41-1321	Corner bracket 15mm L	2
8	McMaster-Carr: 92000A105	Screw 2.5mm D x 8.5mm L	10
9	RevRobotics: REV-21-1203	Corner bracket 1in L	1

Figure C2.3: Assembly drawing of the linear slide assembly

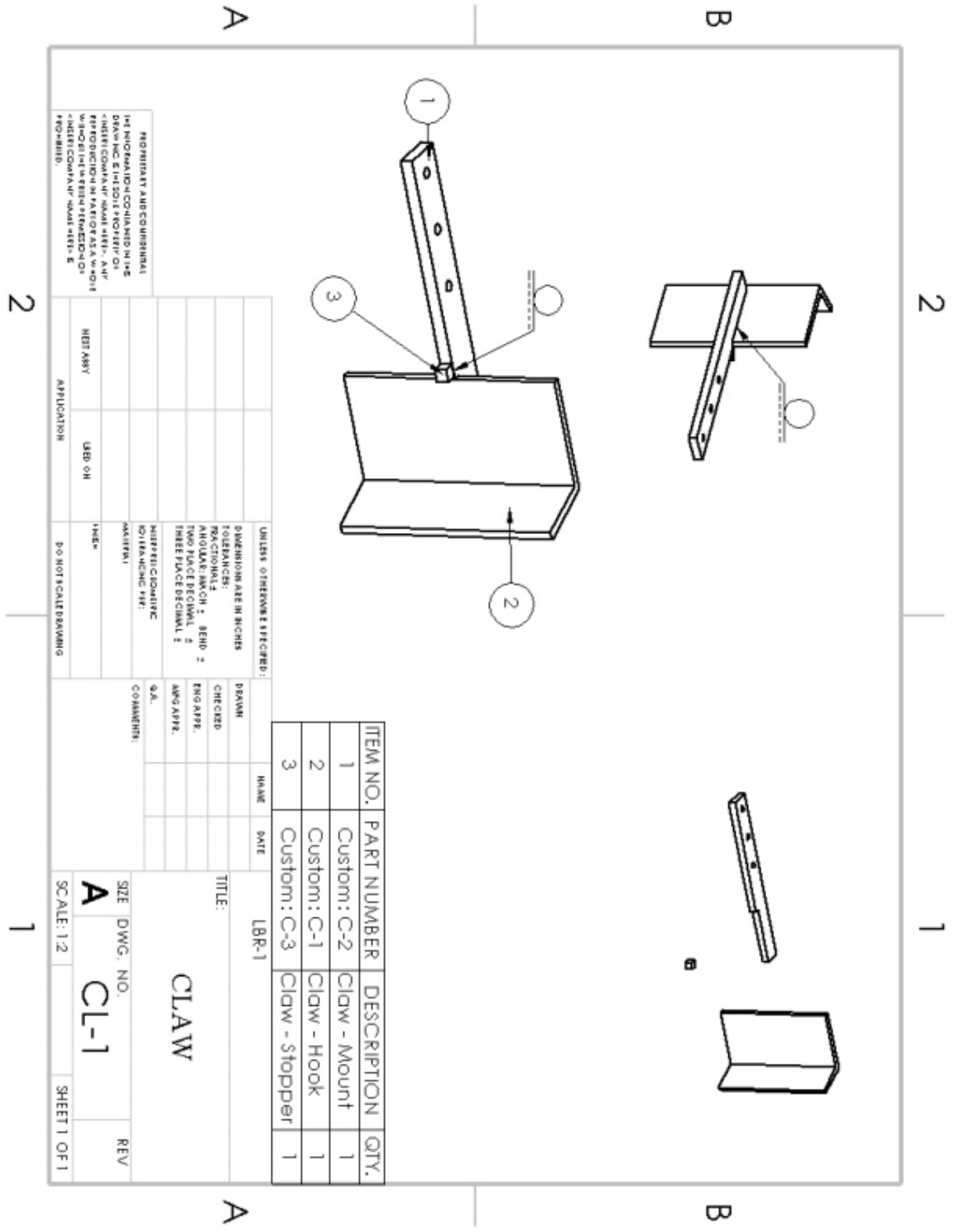


Figure C2.4: Assembly drawing of the claw for the cube latch assembly

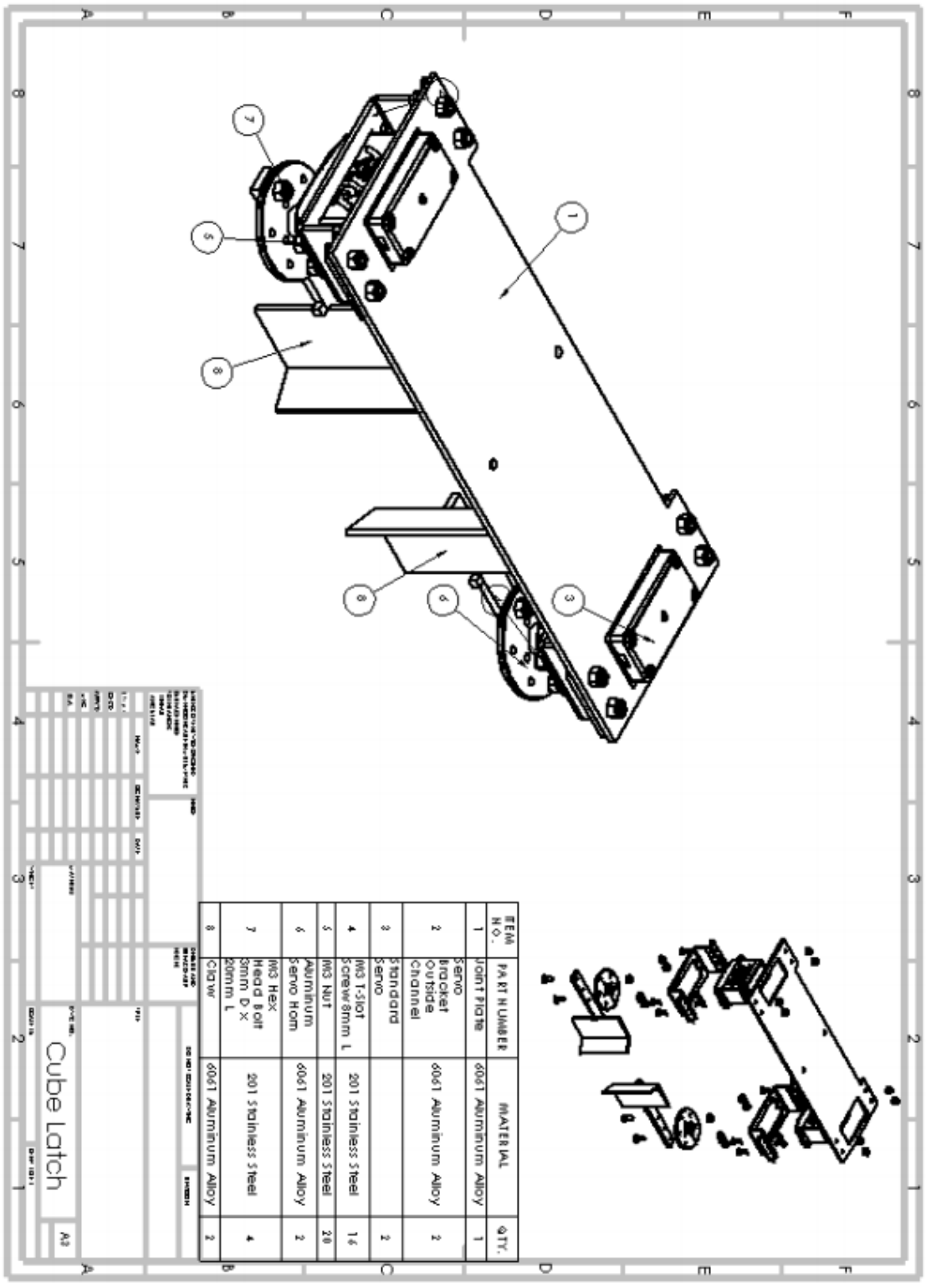


Figure C2.5: Assembly drawing of the cube latch assembly.

C3 Sub-assemblies of Rotating Plate Mechanism

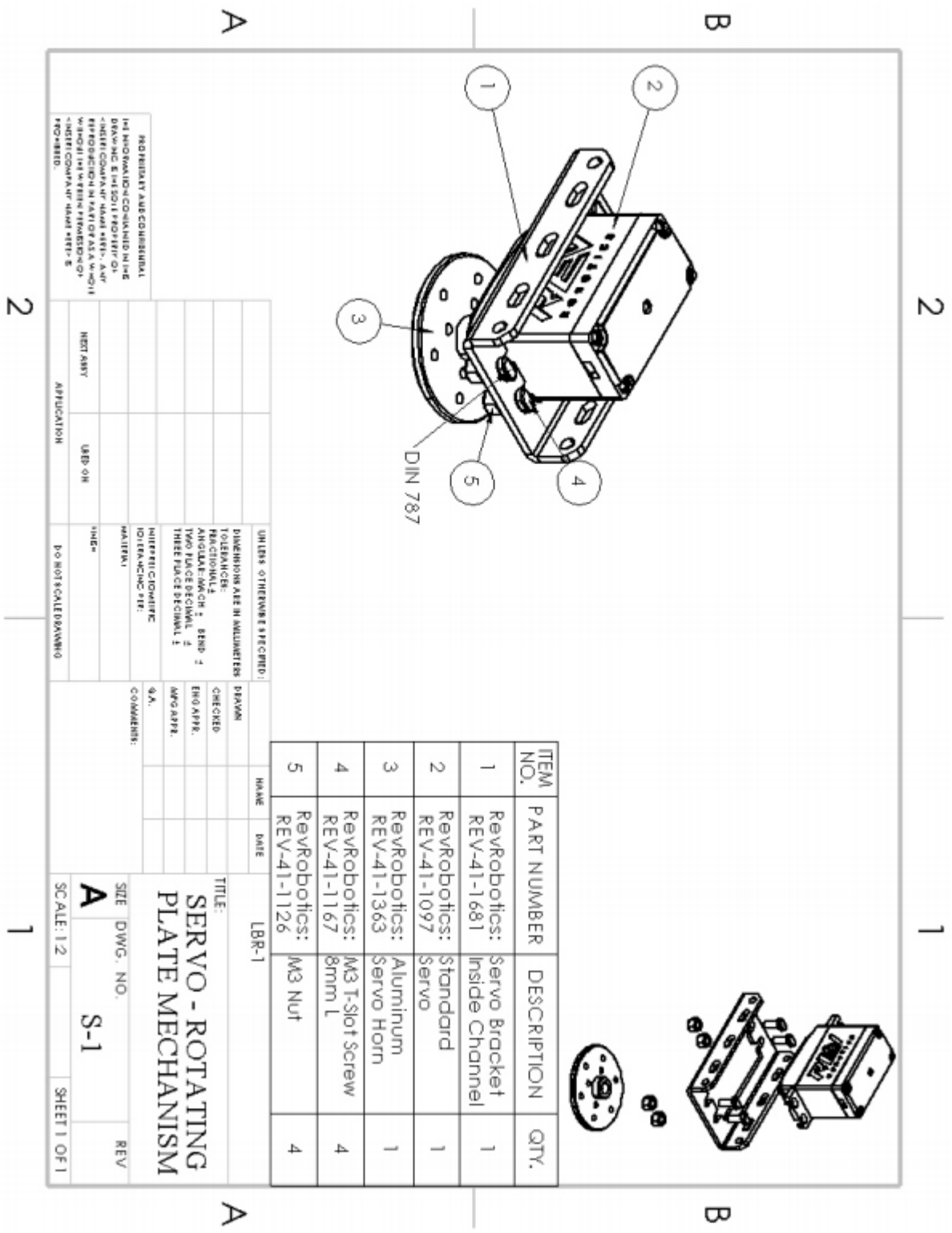
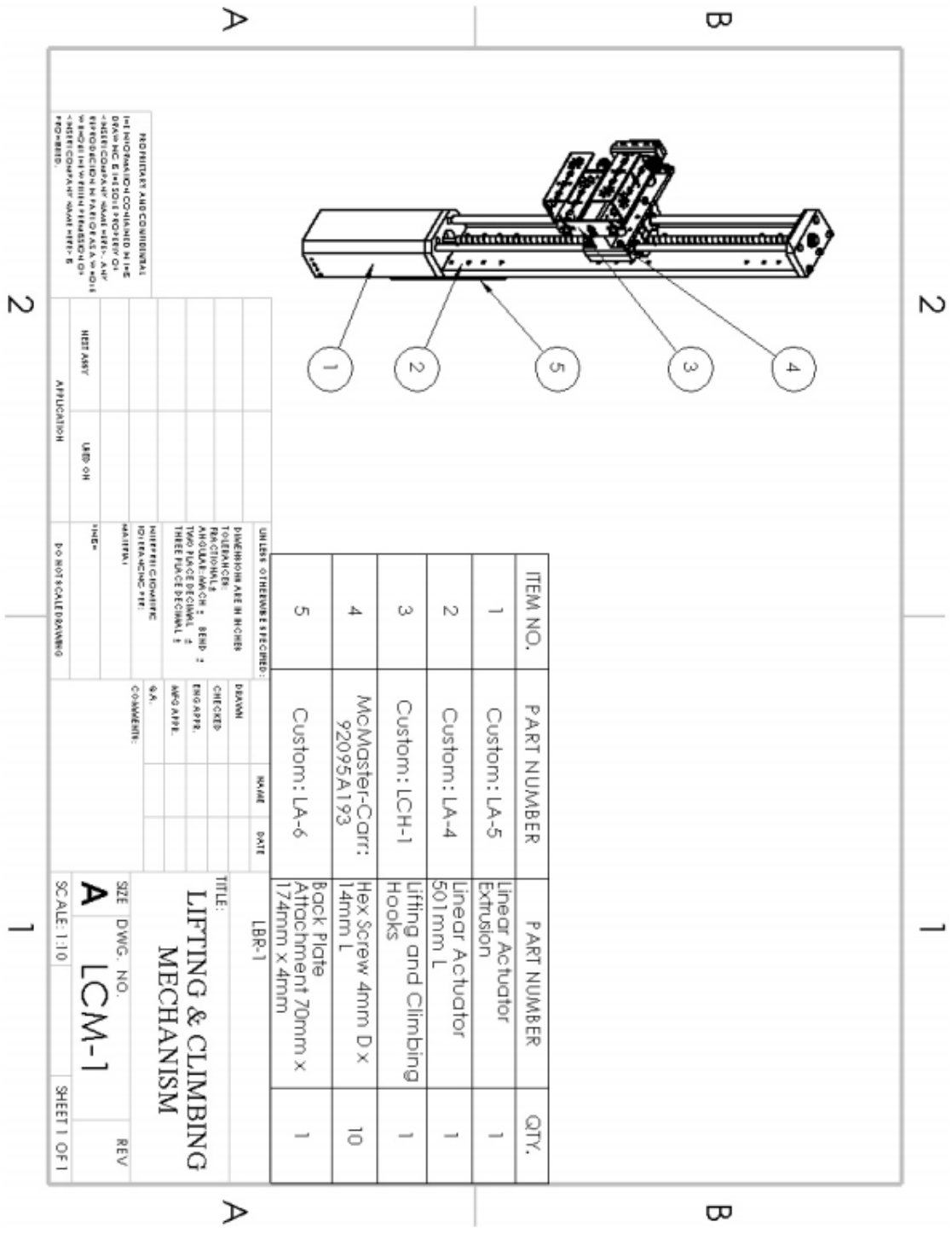


Figure C3.1: Assembly drawing of the servo assembly for the rotating plate mechanism.

C4 Main Assemblies



ITEM NO.	PART NUMBER	PART NUMBER	QTY.
1	Custom: LA-5	Linear Actuator Extrusion	1
2	Custom: LA-4	Linear Actuator 501mm L	1
3	Custom: LCH-1	Lifting and Climbing Hooks	1
4	McMaster-Carr: 92095A193	Hex Screw 4mm D x 14mm L	10
5	Custom: LA-6	Back Plate Attachment 70mm x 174mm x 4mm	1

PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

NEXT ASSY | USED ON | APPLICATION | DO NOT SCALE DRAWING

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH

	NAME	DATE
DRAWN		
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

TITLE:
LIFTING & CLIMBING MECHANISM
LBR-1

SIZE A | DWG. NO. LCM-1 | REV
SCALE: 1:10 | SHEET 1 OF 1

Figure C4.1: Assembly drawing of the lifting and climbing mechanism.

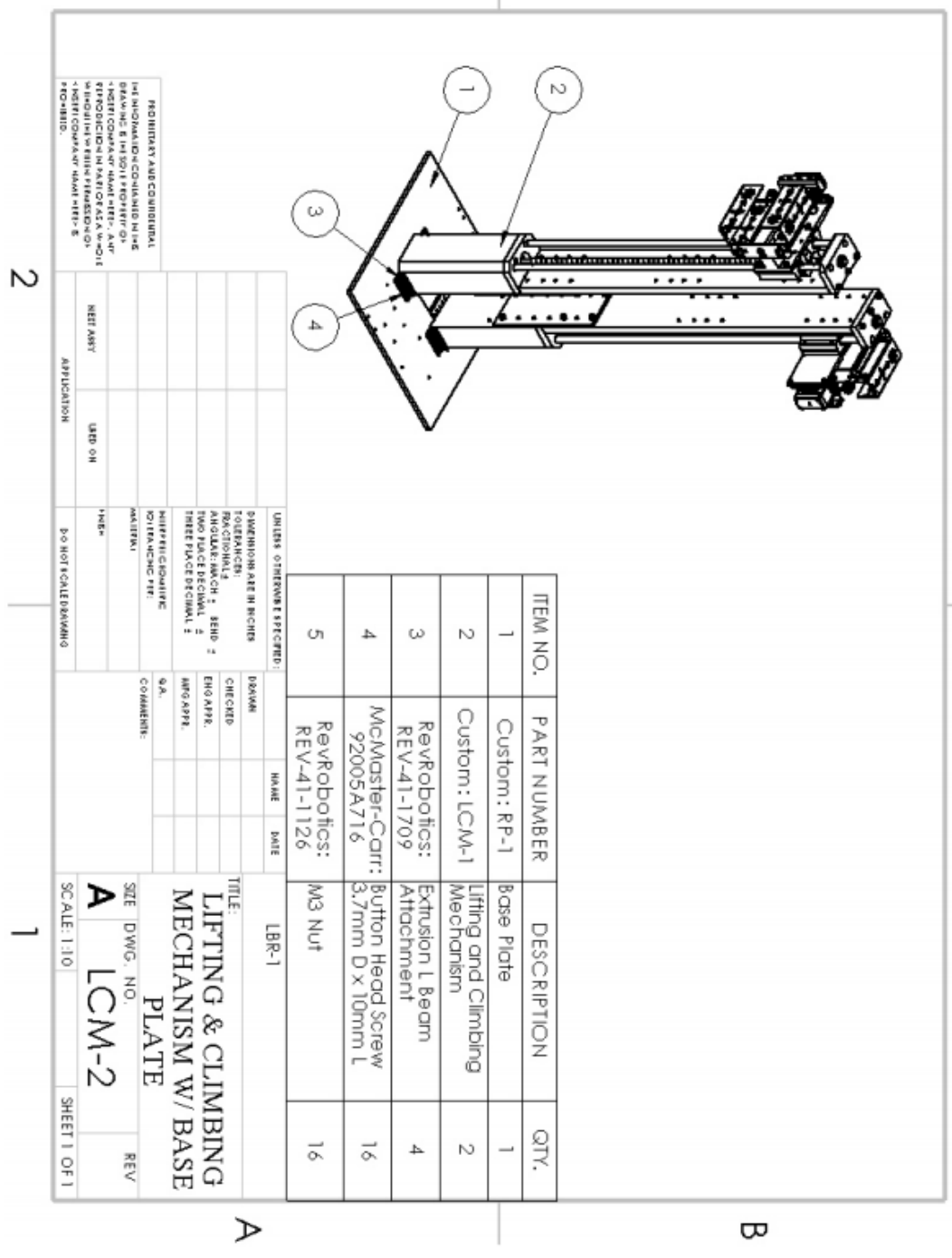
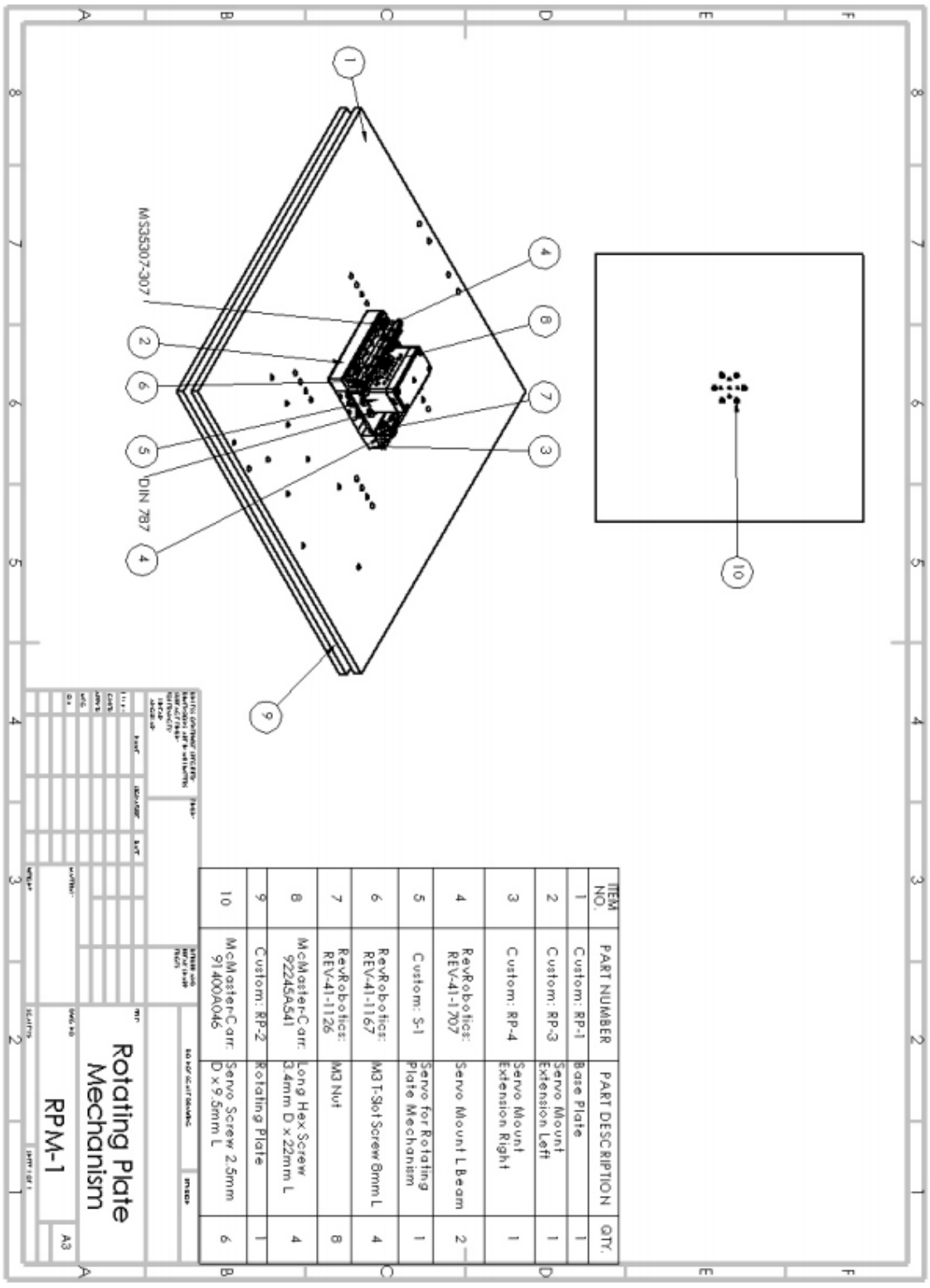


Figure C4.2: Assembly drawing of the lifting and climbing mechanism on base plate.

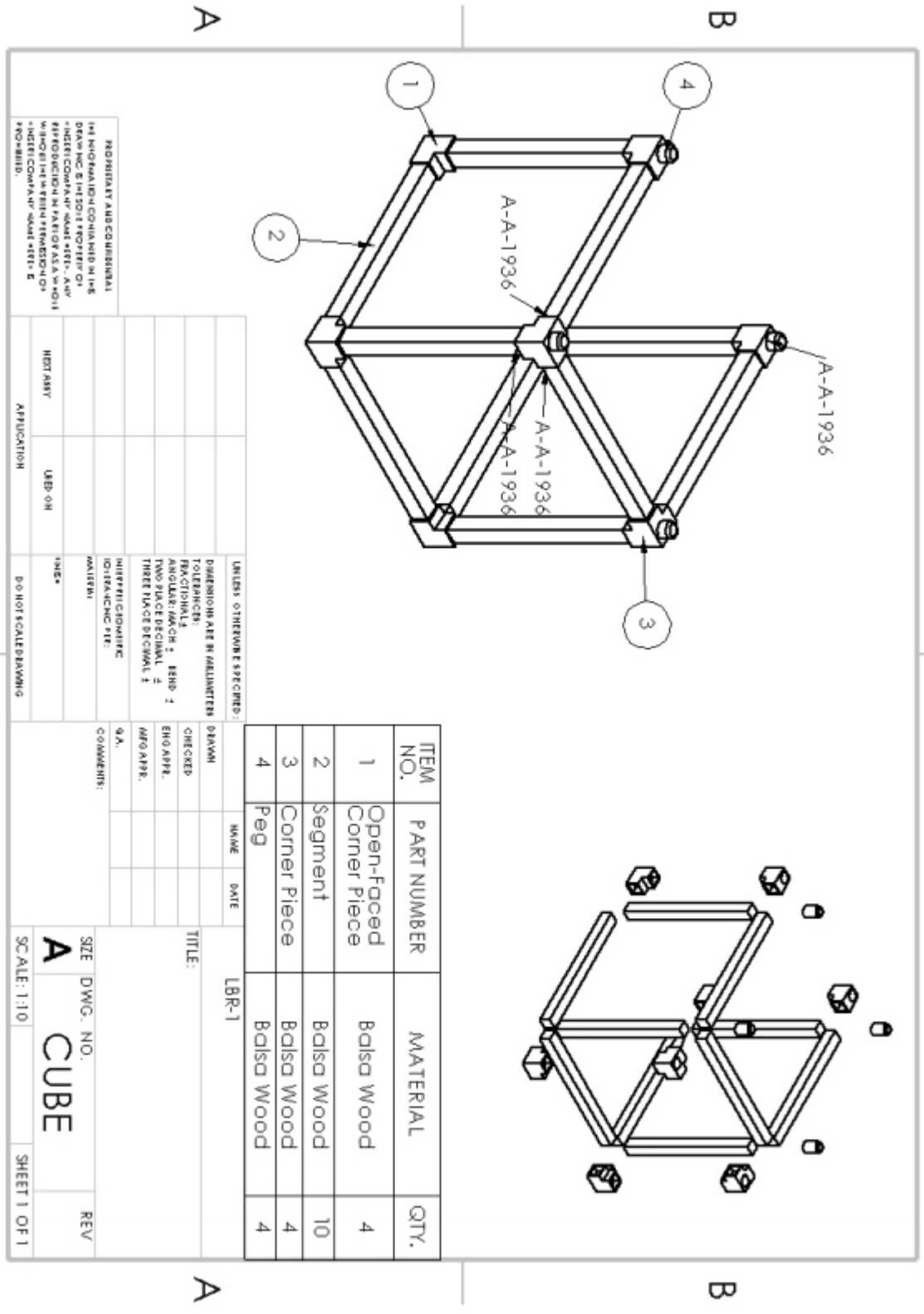
Figure C4.3: Assembly drawing of the cube latch mechanism.



ITEM NO.	PART NUMBER	PART DESCRIPTION	QTY.
1	Custom: RP-1	Base Plate	1
2	Custom: RP-3	Servo Mount Extension Left	1
3	Custom: RP-4	Servo Mount Extension Right	1
4	RevRobotics: REV-41-1707	Servo Mount L Beam	2
5	Custom: S-1	Servo for Rotating Plate Mechanism	1
6	RevRobotics: REV-41-1167	M3 T-Slot Screw 8mm L	4
7	RevRobotics: REV-41-1126	M3 Nut	8
8	McMaster-Carr: 92245A541	Long Hex Screw 3.4mm D x 22mm L	4
9	Custom: RP-2	Rotating Plate	1
10	McMaster-Carr: 91400A046	Servo Screw 2.5mm D x 9.5mm L	6

Figure C4.4: Assembly drawing of the rotating plate mechanism.

Figure C4.5: Assembly drawing of the final robot design



ITEM NO.	PART NUMBER	MATERIAL	QTY.
1	Open-Faced Corner Piece	Balsa Wood	4
2	Segment	Balsa Wood	10
3	Corner Piece	Balsa Wood	4
4	Peg	Balsa Wood	4

Figure C4.6: Assembly drawing of the cube element.

D Code

D1 Webserver (Python)

D1.1 webserver.py

```
from flask import Flask, render_template, request, jsonify,
make_response import rospy
from std_msgs.msg import Bool, Int16


app = Flask(__name__)

active_pub = rospy.Publisher( /active , Bool, queue_size=10) # Robot can
move reset_pub = rospy.Publisher( /reset , Bool, queue_size=10) # Robot
will
 ,! immediate descend to ground
num_blocks_pub = rospy.Publisher( /num_blocks , Int16, queue_size=10) #
 Blocks ,! the robot is working with


climb_enable_pub = rospy.Publisher( /climb_to_height_enable , Bool,
 ,! queue_size=10)  # Robot will climb to a target height
climb_height_pub = rospy.Publisher( /climb_to_height_target , Int16,
 ,! queue_size=10)  # Height the robot will climb to

# index route: just returns the index.html webpage @app.route("/")
def index():
    return render_template("index.html")

# update route: updates a ros topic, used by the web interface
@app.route("/update", methods=["GET"])
def update():
    topic                          =
    str(request.args.get("topic"))
    value                          =
    str(request.args.get("value"))

    if topic == "active":
        active_pub.publish(data = False if value == "false" else value ==
        "true")
    elif topic == "reset":
        reset_pub.publish(data = False if value == "false" else value ==
        "true")
    elif topic == "num_blocks":
        num_blocks_pub.publish(Int16(data=value))
    elif topic == "climb_to_height_enable":
```

```
            climb_enable_pub.publish(data = False if value == "false" else
             value == ,! "true")

        elif topic == "climb_to_height_target":
            climb_height_pub.publish(Int16(data=value))
        else:
            return make_response(jsonify({"result": "invalid topic"}))
        return make_response(jsonify({"result": "success"}))

@app.route("/test")
def test():
        active_pub.publish(Bool(data=False))
        return "test"

#  starts the webserver and initializes the ros nodes def start_webserver():
        rospy.init_node("webserver") # init ros node
        app.run(host="0.0.0.0")
```

D2 Robot Initialization Script (Python)

D2.1 main.py

```
import web.webserver
import states.overall
import threading
import time
import rospy

if __name__ == "__main__":
    rospy.init_node("mission_control")
    webserver_thread = threading.Thread(target=web.webserver.start_webserver)
    #webserver_thread.start()

    print("Starting overall state machine")
    states.overall.overall_state_machine()
    time.sleep(1)
```

D3 Robot Shared Variable Controller (Python)

D3.1 variables.py

```
# functions managing persistant variables

import rospy
from std_msgs.msg import Bool, Float64, Int16
```

```
variables = {
    "num_blocks": 0,
    "current_block": 0,

    "current_height": 0,
    "max_height": 0,
    "climb_height_enable": False,
    "climb_to_height": 0,

    "reset": False
}

def get_num_blocks():
    return variables["num_blocks"]

def set_num_blocks(value):
    variables["num_blocks"] = value


def get_current_block():
    return variables["current_block"]

def set_current_block(value):
    variables["current_block"] = value

def get_current_height():
    return variables["current_height"]

def set_current_height(value):
    variables["current_height"] = value

def get_max_height():
    return variables["max_height"]

def set_max_height(value):
    variables["max_height"] = value


def get_climb_height_enable():
    return variables["climb_height_enable"]


def get_climb_to_height():
    return variables["climb_to_height"]
```

```
def set_climb_to_height(value):
    variables["climb_to_height"] = value


def get_reset_enable():
    return variables["reset"]

def reset_sub_callback(msg):
    variables["reset"] = msg.data
reset_sub = rospy.Subscriber( /reset/ , Bool, reset_sub_callback)

def climb_to_height_target_sub_callback(msg):
    variables["climb_to_height"] = msg.data
reset_sub = rospy.Subscriber( /climb_to_height_target/ ,
 Int16, ,! climb_to_height_target_sub_callback)

def climb_to_height_enable_sub_callback(msg):
    variables["climb_height_enable"] = msg.data
reset_sub = rospy.Subscriber( /climb_to_height_enable/ ,
 Bool, ,! climb_to_height_enable_sub_callback)
```

D4 Robot Status Controller (Python)

D4.1 status.py

```
# contains functions that read topics on the status of robot parts

import rospy
from std_msgs.msg import Bool, Int16


robot_statuses = {
    "vertical_actuator": "low",
    "horizontal_actuator": "low",

    "left_grabber_servo": 0,
    "right_grabber_servo": 0,

    "left_forklift_servo": 0,
    "right_forklift_servo": 0,

    "bottom_servo": 0
}
```

```
def status_vertical_actuator():
    value = robot_statuses["vertical_actuator"]
    #print("The current value of the vertical actuator is " + str(value))

def status_horizontal_actuator(value="none"):
    value = robot_statuses["horizontal_actuator"]
    #print("The current value of the horizontal actuator is " + str(value))

def status_left_grabber_servo(value="none"):
    value = robot_statuses["left_grabber_servo"]
    #print("The current value of the left grabber servo is " + str(value))

def status_right_grabber_servo(value="none"):
    value = robot_statuses["right_grabber_servo"]
    #print("The current value of the right grabber servo is " + str(value))


def status_left_forklift_servo(value="none"):
    value = robot_statuses["left_forklift_servo"]
    #print("The current value of the left forklift servo is " + str(value))

def status_right_forklift_servo(value="none"):
    value = robot_statuses["right_forklift_servo"]
   #print("The current value of the right forklift servo is " + str(value))

def status_bottom_servo(value="none"):
    value = robot_statuses["bottom_servo"]
    #print("The current value of the bottom servo is" + str(value))

def vertical_actuator_sub_callback(msg):
    robot_statuses["vertical_actuator"] = msg.data
vertical_actuator_sub = rospy.Subscriber( /vertical_actuator_status/ ,
 Int16, vertical_actuator_sub_callback)

def horizontal_actuator_sub_callback(msg):
    robot_statuses["horizontal_actuator"] = msg.data
horizontal_actuator_sub = rospy.Subscriber( /horizontal_actuator_status/ ,
 Int16, horizontal_actuator_sub_callback)

def left_grabber_servo_sub_callback(msg):
    robot_statuses["left_grabber_servo"] = msg.data
```

```
left_grabber_servo_sub = rospy.Subscriber( /left_grabber_servo_status/ ,
 Int16, left_grabber_servo_sub_callback)

def right_grabber_servo_sub_callback(msg):
    robot_statuses["right_grabber_servo"] = msg.data
right_grabber_servo_sub = rospy.Subscriber( /right_grabber_servo_status/ ,
 Int16, right_grabber_servo_sub_callback)

def left_forklift_servo_sub_callback(msg):
    robot_statuses["left_forklift_servo"] = msg.data
left_forklift_servo_sub = rospy.Subscriber( /left_forklift_servo_status/ ,
 Int16,  left_forklift_servo_sub_callback)

def right_forklift_servo_sub_callback(msg):
    robot_statuses["right_forklift_servo"] = msg.data
right_forklift_servo_sub = rospy.Subscriber(
 /right_forklift_servo_status/ , Int16,
 right_forklift_servo_sub_callback)

def bottom_servo_sub_callback(msg):
    robot_statuses["bottom_servo"] = msg.data
bottom_servo_sub = rospy.Subscriber( /bottom_servo_status/ ,
 Int16,  bottom_servo_sub_callback)
```

D5 Finite State Machines (Python)

D5.1 Overall State Machine -- overall.py

```
# overall state machine

import states.construction
import states.ascension
import states.descension
import states.actuations
import storage.variables

def overall_state_machine():
    state = 0
    # runs until told to exit
    print("Running overall state machine")

 while state != -
               1:
   if state == 0:
            print("Running IDLE state machine")
```

```
            # idle
            states.actuations.force_neutral_posit
            ion()
            storage.variables.set_num_blocks(3)
            storage.variables.set_climb_to_height
            (1) state = 2
        elif state == 1:
            print("Running CONSTRUCTION state machine")
            status = states.construction.construction_state_machine()
            storage.variables.set_climb_to_height(3)
            if status: # check if completed
                successfully state = 2
        elif state == 2:
            print("Running ASCENSION state machine")
            status = states.ascension.ascension_state_machine()
            storage.variables.set_climb_to_height(1)
            if status: # check if completed
                successfully state = 3

        elif state == 3:
            print("Running DESCENSION state machine")
            status = states.descension.descension_state_machine()
            if status:  # check if completed successfully
                state = -1
    print("Exited state machines")
```

D5.2 Construction State Machine -- construction.py

```
# state machine for construction

import simulator.actuators
import simulator.status
import storage.variables
import states.actuations

def construction_state_machine():
    state = 0

    states = {
        0: neutral_position,
        1: elevate_tower,
        2: reach_cube,
        3: pull_cube,
        4: deelevate_tower
```

```
    }

    print("  Starting construction state machine")

    while state != -1:
        state = states[state]()
        #a = raw_input(">")

    print("  Finished construction state machine")
    return True

def neutral_position():
    print("    In construction::neutral position")
    states.actuations.neutral_position()
    num_blocks = storage.variables.get_num_blocks()

    if num_blocks > 0:
        print("      There are " + str(num_blocks) + " blocks to construct")
        storage.variables.set_num_blocks(num_blocks - 1)
        return 1
    elif storage.variables.get_reset_enable():
        print("  Asked to reset")
        return -1
    else:
        print("      There are no more blocks to construct")
        return -1

def elevate_tower():
    print(" In contruction::elevate tower")
    states.actuations.elevate_structure()
    return 2


def reach_cube():
    print("    In contruction::reach cube")
    states.actuations.reach_cube()
    return 3

def pull_cube():
    print("    In contruction::pull cube")
    states.actuations.pull_cube()
    return 4

def deelevate_tower():
```

```
    print(" In contruction::deelevate tower")
    states.actuations.deelevate_structure()
    return 0
```

D5.3 Ascension State Machine -- ascension.py

```
# state machine for ascension

import simulator.actuators
import simulator.status
import storage.variables
import states.actuations

def ascension_state_machine():
    state = 0
    states = {
           0: start_of_cycle,
           1: elevate_forklift,
              2: forklift_hooks_out,
        3: rotate_buttom_in,
        4: deelevate_forklift,
        5: rotate_buttom_out,
        6: forklift_hooks_in,
        7: end_of_cycle,
    }

    print("  Starting ascension state machine")

    while state != -1:
        state = states[state]()

    print("  Finished ascension state machine")
    return True

def start_of_cycle():
    print("    In ascension::start of cycle")
    current_height                                          =
    storage.variables.get_current_height()  target_height
    =             storage.variables.get_climb_to_height()
    height_difference = target_height - current_height
    if height_difference > 0:
        print(" There are " + str(height_difference) + " blocks left to ,!
         climb")

        return 1
```

```
    else:
        print("      There are no blocks left to climb")
        return -1

def elevate_forklift():
    print(" In ascension::elevate forklift")
    states.actuations.elevate_forklift()
    return 2


def forklift_hooks_out():
    print(" In ascension::forklift hooks out")
    states.actuations.forklift_forks_out()
    return 3


def rotate_buttom_in():
    print(" In ascension::rotate bottom in")
    states.actuations.rotate_bottom_in()
    return 4


def deelevate_forklift():
    print(" In ascension::deelevate forklift")
    states.actuations.deelevate_forklift()
    return 5

def rotate_buttom_out():
    print(" In ascension::rotate bottom out")
    states.actuations.rotate_bottom_out()
    return 6


def forklift_hooks_in():
    print(" In ascension::forklift hooks in")
    states.actuations.forklift_forks_in()
    return 7


def end_of_cycle():
    print("    In ascension::end of cycle")
    current_height = storage.variables.get_current_height() +
    1 storage.variables.set_current_height(current_height)
    return 0
```

D5.4 Descension State Machine -- descension.py

```
# state machine for descension

import simulator.actuators
import simulator.status
import storage.variables
import states.actuations

def descension_state_machine():
    state = 0

    states = {
        0: start_of_cycle,
        1: rotate_buttom_in,
        2: elevate_forklift,
        3: rotate_buttom_out,
        4: forklift_hooks_in,
        5: deelevate_forklift,
        6: forklift_hooks_out,
        7: end_of_cycle,
    }

    print("  Starting descension state machine")

    while state != -1:
        state = states[state]()

    print("  Finished descension state machine")
    return True

def start_of_cycle():
    print("    In descension::start of cycle")
    current_height                                          =
    storage.variables.get_current_height()
    target_height                                           =
    storage.variables.get_climb_to_height()
    height_difference = target_height - current_height
    if height_difference < 0:
        print(" There are " + str(-1 * height_difference) + " blocks left to
         ,! descend")

        return 1
    else:
```

```
        print("      There are no blocks left to climb")
        return -1

def rotate_buttom_in():
    print(" In descension::rotate bottom
    in")
    states.actuations.rotate_bottom_in()
    return 2


def elevate_forklift():
    print(" In descension::elevate
    forklift")
    states.actuations.elevate_forklift()
    return 3


def rotate_buttom_out():
    print(" In descension::rotate bottom
    out")
    states.actuations.rotate_bottom_out()
    return 4


def forklift_hooks_in():
    print(" In descension::forklift hooks
    in")
    states.actuations.forklift_forks_in()
    return 5


def deelevate_forklift():
    print(" In descension::deelevate
    forklift")
    states.actuations.deelevate_forklift()
    return 6


def forklift_hooks_out():
    print(" In descension::forklift hooks
    out")
    states.actuations.forklift_forks_out()
    return 7
```

```
def end_of_cycle():
    print("    In descension::end of cycle")
    current_height = storage.variables.get_current_height() - 1
    storage.variables.set_current_height(current_height)
    return 0
```

D5.5 State Machine Actuator Control -- actuations.py

```
# contains function for actuator sequences

import simulator.actuators
import simulator.status

# lifts the structure def elevate_structure():
#     rotate forklift servos out forklift_forks_out()

#     raise top actuator full
    simulator.actuators.vertical_actuator(2)

def deelevate_structure():
    # lower top actuator half
    simulator.actuators.vertical_actuator(1)

#     rotate forklift servos in forklift_forks_in()

#     low top actuator full
    simulator.actuators.vertical_actuator(0)

# only raises the forklift actuator def elevate_forklift():
    # raise top actuator full
    simulator.actuators.vertical_actuator(2)

def deelevate_forklift():
    # low top actuator full
    simulator.actuators.vertical_actuator(0)

def forklift_forks_out():
#     rotate cube grabber servos out simulator.actuators.forklift_servos(-1)

def forklift_forks_in():
#     rotate cube grabber servos in simulator.actuators.forklift_servos(1)

def reach_cube():
```

```
    # extend grabber full
    simulator.actuators.horizontal_actuator(2)

#     rotate cube grabber servos out simulator.actuators.left_grabber_servo(-
1) simulator.actuators.right_grabber_servo(-1)

def pull_cube():
    # retract grabber half
    simulator.actuators.horizontal_actuator(1)

#     rotate cube grabber servos in simulator.actuators.left_grabber_servo(1)
simulator.actuators.right_grabber_servo(1)

#     retract grabber full
    simulator.actuators.horizontal_actuator(-2)

def rotate_bottom_out():
#     rotate bottom servo out simulator.actuators.bottom_servo(1)

def rotate_bottom_in():
#     rotate bottom servo out simulator.actuators.bottom_servo(1)

# assumes a neutral position
def neutral_position():
    # check the status of the vertical actuator
    if simulator.status.status_vertical_actuator() != "low":
        simulator.actuators.vertical_actuator(0)
    simulator.actuators.vertical_actuator(-2)

    # check the status of the horizontal actuator
    if simulator.status.status_horizontal_actuator() != "low":
        simulator.actuators.horizontal_actuator(0)
    simulator.actuators.horizontal_actuator(-2)

    # check the status of the grabber servos
    if simulator.status.status_left_grabber_servo() != 1:
        simulator.actuators.left_grabber_servo(1)
    if simulator.status.status_right_grabber_servo() != 1:
        simulator.actuators.right_grabber_servo(1)

    # check the status of the forklift servos
    if simulator.status.status_left_forklift_servo() != 1:
        simulator.actuators.left_forklift_servo(1)
    if simulator.status.status_right_forklift_servo() != 1:
```

```
        simulator.actuators.right_forklift_servo(1)

    # check the status of the bottom servo
    if simulator.status.status_bottom_servo() != -1:
        simulator.actuators.bottom_servo(-1)

# forces a neutral position, used at the start to initialize servos def
force_neutral_position():
    deelevate_forklift()
    forklift_forks_in()
    simulator.actuators.left_grabber_servo(
    1)
    simulator.actuators.right_grabber_servo
    (1)
    simulator.actuators.left_forklift_servo
    (1)
    simulator.actuators.right_forklift_serv
    o(1) simulator.actuators.bottom_servo(-
    1)
```

D6 Simulation Actuator Relay (Python)

D6.1 actuators.py

```
# contains functions that convert desired actuator values to ros topic controls

import rospy
from std_msgs.msg import Bool, Float32, Int16
import time

cube_grabber_actuator_middle_pub =
 ,! rospy.Publisher("/lbr_robot/cube_grabber_actuator_middle_position", Float32, ,!
 queue_size=10)
cube_grabber_actuator_top_pub =
 ,! rospy.Publisher("/lbr_robot/cube_grabber_actuator_top_position", Float32, ,!
 queue_size=10)
forklift_left_actuator_pub =
 ,! rospy.Publisher("/lbr_robot/forklift_left_actuator_position", Float32, ,!
 queue_size=10)

forklift_right_actuator_pub =
 ,! rospy.Publisher("/lbr_robot/forklift_right_actuator_position", Float32, ,!
 queue_size=10)


cube_grabber_servo_pub =
```

```
 ,! rospy.Publisher("/lbr_robot/cube_grabber_servos_position", Float32, ,!
 queue_size=10)
forklift_servos_pub = rospy.Publisher("/lbr_robot/forklift_servos_position",
 ,!    Float32, queue_size=10)
forklift_left_servo_pub =
 ,! rospy.Publisher("/lbr_robot/forklift_left_servo_position", Float32, ,!
 queue_size=10)
 forklift_right_servo_pub =
 ,! rospy.Publisher("/lbr_robot/forklift_right_servo_position", Float32,
 ,! queue_size=10)
bottom_plate_servo_pub =
 ,! rospy.Publisher("/lbr_robot/bottom_plate_servo_position",
 Float32, ,! queue_size=10)

def vertical_actuator(state):
#     will need to check the prior state for the time delay if state == 0:
        #print("vertical actuator rostopic to stop")
        forklift_left_actuator_pub.publish(float(0.0
        ))
        forklift_right_actuator_pub.publish(float(0.
        0)) time.sleep(4)
    elif state == 1:
        #print("vertical actuator rostopic to raise
        half")
        forklift_left_actuator_pub.publish(float(0.8))
        forklift_right_actuator_pub.publish(float(0.8))
        time.sleep(4)

    elif state == 2:
        #print("vertical actuator rostopic to raise
        full")
        forklift_left_actuator_pub.publish(float(1.6))
        forklift_right_actuator_pub.publish(float(1.6))
        time.sleep(4)

    elif state == -1:
        #print("vertical actuator rostopic to lower
        half")
        forklift_left_actuator_pub.publish(float(0.8))
        forklift_right_actuator_pub.publish(float(0.8))
        time.sleep(4)

    elif state == -2:
        #print("vertical actuator rostopic to lower
        full")
```

```
        forklift_left_actuator_pub.publish(float(0.1))
        forklift_right_actuator_pub.publish(float(0.1))
        time.sleep(4)

    return True

def horizontal_actuator(state):
    if state == 0:
        #print("horizontal actuator rostopic to stop")
        time.sleep(0)
    elif state == 1:
        #print("horizontal actuator rostopic to raise half")
        cube_grabber_actuator_middle_pub.publish(float(0.1))
        cube_grabber_actuator_top_pub.publish(float(0.1))
        time.sleep(3)
    elif state == 2:
        #print("horizontal actuator rostopic to raise full")
        cube_grabber_actuator_middle_pub.publish(float(0.6))
        cube_grabber_actuator_top_pub.publish(float(0.6))
        time.sleep(5)
    elif state == -1:
        #print("horizontal actuator rostopic to lower half")
        cube_grabber_actuator_middle_pub.publish(float(0.1))
        cube_grabber_actuator_top_pub.publish(float(0.1))
        time.sleep(3)
    elif state == -2:
        #print("horizontal actuator rostopic to lower full")
        cube_grabber_actuator_middle_pub.publish(float(0.01))
        cube_grabber_actuator_top_pub.publish(float(0.01))
        time.sleep(3)
    return True

def bottom_servo(state):
    #print("  turning bottom servo to PWM " + str(pwm))
    bottom_plate_servo_pub.publish(float(state * 3))
    time.sleep(3)

def left_grabber_servo(state):
    #print("  turning grabber left servo to PWM " + str(pwm))
    cube_grabber_servo_pub.publish(float(state))
    time.sleep(2)

def right_grabber_servo(state):
    #print("  turning grabber right servo to PWM " + str(pwm))
```

```
    cube_grabber_servo_pub.publish(float(state))
    time.sleep(2)

def forklift_servos(state):
    #print("  turning forklift left servo to PWM " + str(pwm))
    forklift_servos_pub.publish(float(state * 5))
    time.sleep(2)

def left_forklift_servo(state):
    #print("  turning forklift left servo to PWM " + str(pwm))
    forklift_left_servo_pub.publish(float(state * 5))
    time.sleep(2)

def right_forklift_servo(state):
    #print("  turning forklift right servo to PWM " + str(pwm))
    forklift_right_servo_pub.publish(float(state * 5))
    time.sleep(2)
```

D7 Gazebo Plugins (C++)

D7.1 Servo Controller -- plugin servo.cc

```
#ifndef _PLUGIN_SERVO_
#define _PLUGIN_SERVO_

#include  <gazebo/gazebo.hh>
#include  <gazebo/physics/physics.hh>

#include  <thread>
#include  "ros/ros.h"
#include  "ros/callback_queue.h"
#include  "ros/subscribe_options.h"
#include  "std_msgs/Float32.h"


namespace gazebo
{
  class ServoControl : public ModelPlugin
  {
    public: ServoControl() : ModelPlugin()
      {
        printf("Initializing Servo Plugin\n");
      }
```

```
    private: physics::ModelPtr model;                          // model
    pointer

    private: physics::JointPtr forklift_left_servo_top_joint;      //
    forklift
     ,! left servo joint pointer
    private: physics::JointPtr forklift_left_servo_bottom_joint;      //
    forklift
     ,! left servo joint pointer

    private: physics::JointPtr forklift_right_servo_top_joint;     //
    forklift
     ,! right servo joint pointer
    private: physics::JointPtr forklift_right_servo_bottom_joint;      //
    forklift
     ,! left servo joint pointer

    private: physics::JointPtr bottom_plate_servo_joint;       // bottom
    plate
     ,! servo joint pointer

    private: physics::JointPtr cube_grabber_left_servo_joint;       // cube
     ,! grabber servo joint pointer
    private: physics::JointPtr cube_grabber_right_servo_joint;      // cube
     ,! grabber servo joint pointer

    private: common::PID vel_pid;                                  // PID
     ,! controller

     private: float initialPosition;
     private: std::unique_ptr<ros::NodeHandle> rosNode;
     private: ros::Subscriber rosSubForkliftServos;
     private: ros::Subscriber rosSubForkliftLeftServo;
     private: ros::Subscriber rosSubForkliftRightServo;
     private: ros::Subscriber rosSubBottomPlateServo;
     private: ros::Subscriber rosSubCubeGrabberServos;
     private: ros::CallbackQueue rosQueue;
     private: std::thread rosQueueThread;

public: void OnUpdate(const common::UpdateInfo &_info) { this->model-
  >GetJointController()->Update(); //this->model-
  >GetJoint("lbr_forklift_left::forklift_left_hook_top")->
   ,! SetVelocity(1, -5.0);
}

public: void Load(physics::ModelPtr _model, sdf::ElementPtr _sdf)
```

```
   {
/              store pointer to the model this->model = _model;

/        check joint count
      if (_model->GetJointCount() == 0)
      {
        std::cout << "Plugin Servo: Joint Count is 0, likely the plugin is not
         ,! loaded" << std::endl;
        return;
      }

      std::vector<boost::shared_ptr<gazebo::physics::Joint>> joints
       = ,! _model->GetJoints();
      for (int i=0; i<_model->GetJointCount(); i++)
      {
        // assign the joint for the left servo forklift TOP

if (joints.at(i)->GetName() ==
         ,! "lbr_forklift_left::forklift_left_hook_top")
        { this->forklift_left_servo_top_joint = _model->GetJoints()[i]; }

/          assign the joint for the left servo forklift BOTTOM if
(joints.at(i)->GetName() ==
         ,!"lbr_forklift_left::forklift_left_hook_bottom")
        { this->forklift_left_servo_bottom_joint = _model->GetJoints()[i]; }

/          assign the joint for the right servo forklift TOP
        if (joints.at(i)->GetName() ==
         ,! "lbr_forklift_right::forklift_right_hook_top")
        { this->forklift_right_servo_top_joint = _model->GetJoints()[i]; }

/          assign the joint for the right servo forklift BOTTOM if
(joints.at(i)->GetName() ==
         ,!"lbr_forklift_right::forklift_right_hook_bottom")
        { this->forklift_right_servo_bottom_joint = _model->GetJoints()[i]; }

/          assign the joint for the bottom plate servo
        if (joints.at(i)->GetName() == "bottom_plate_servo")
        { this->bottom_plate_servo_joint = _model->GetJoints()[i]; }

/          assign the joint for the cube grabber left servo if (joints.at(i)-
>GetName() ==
         ,!"lbr_cube_grabber::cube_grabber_left_servo")
        { this->cube_grabber_left_servo_joint = _model->GetJoints()[i]; }
```

```
/          assign the joint for the cube grabber right servo
        if (joints.at(i)->GetName() ==
         ,! "lbr_cube_grabber::cube_grabber_right_servo")
        { this->cube_grabber_right_servo_joint = _model->GetJoints()[i]; }
      }
      this->vel_pid = common::PID(1, 0.1, .01); // set a PID ,!
       controller for the servo


      // add the PID controller to the joints
      const auto &jointController = this->model->GetJointController();
        jointController->Reset();
        jointController->AddJoint(model->GetJoint("lbr_forklift_left::forklift
         ,! _left_hook_top"));
        jointController->SetVelocityPID(model->GetJoint("lbr_
         ,! forklift_left::forklift_left_hook_top")->GetScopedName(),
         ,! common::PID(1.0, 0.1, 0.01));
        jointController->SetVelocityTarget(model-
        >GetJoint("lbr_forklift_left::
         ,! forklift_left_hook_top")->GetScopedName(), 0.0);

         !  jointController->AddJoint(model-
         >GetJoint("lbr_forklift_left::forklift_left
         ,! _hook_bottom"));
        jointController->SetVelocityPID(model->GetJoint("lbr_forklift_left::
         ,! forklift_left_hook_bottom")->GetScopedName(), common::PID(1.0,
         0.1,
         ,! 0.01));
        jointController->SetVelocityTarget(model-
        >GetJoint("lbr_forklift_left::
         ,! forklift_left_hook_bottom")->GetScopedName(), 0.0);

        jointController->AddJoint(model->GetJoint("lbr_forklift_right::
         ,! forklift_right_hook_top"));
        jointController->SetVelocityPID(model->GetJoint("lbr_forklift_right::
         ,! forklift_right_hook_top")->GetScopedName(), common::PID(1.0, 0.1,
         ,! 0.01));
              jointController->SetVelocityTarget(model-
>GetJoint("lbr_forklift_right::
         forklift_right_hook_top")->GetScopedName(), 0.0);

        jointController->AddJoint(model->GetJoint("lbr_cube_grabber::
         ,! cube_grabber_left_servo"));
        jointController->SetVelocityPID(model->GetJoint("lbr_cube_grabber::
         ,! cube_grabber_left_servo")->GetScopedName(), common::PID(1.0, 0.1,
```

```
        ,! 0.01));
        jointController->SetVelocityTarget(model->GetJoint("lbr_cube_grabber::
        ,! cube_grabber_left_servo")->GetScopedName(), 0.0);

        jointController->AddJoint(model->GetJoint("lbr_cube_grabber::
        ,! cube_grabber_right_servo"));
        jointController->SetVelocityPID(model->GetJoint("lbr_cube_grabber::
        ,! cube_grabber_right_servo")->GetScopedName(), common::PID(1.0, 0.1,
        ,! 0.01));
        jointController->SetVelocityTarget(model->GetJoint("lbr_cube_grabber::
        ,! cube_grabber_right_servo")->GetScopedName(), 0.0);

        jointController->AddJoint(model-
        >GetJoint("bottom_plate_servo")); jointController-
        >SetVelocityPID(model->GetJoint("bottom_plate_servo") ,! -
        >GetScopedName(), common::PID(1.0, 0.1, 0.01));

        ,! jointController->SetVelocityTarget(model-
        >GetJoint("bottom_plate_servo") ,! ->GetScopedName(), 0.0);

      // set up ROS stuff
      if (!ros::isInitialized())
      {
        int argc = 0;
        char **argv = NULL;

        ros::init(argc, argv, "gazebo_client",
        ,! ros::init_options::NoSigintHandler);
      }

      this->rosNode.reset(new ros::NodeHandle("gazebo_client")); //
       create the ,! ROS node

/       create  a  ROS  topic  and  subscribe  to  it  ros::SubscribeOptions
so_forklift_servos =
       ,!
       ros::SubscribeOptions::create<std_msgs::Float32
       >("/" + ,! this->model->GetName() +
       "/forklift_servos_position",
                                                          1,

                                                          ,!
                                                              boost::bind(&Ser
                                                          voControl
```

```
                                                                ,!
                                                                    ::OnRosMsgForkli
                                                                ftServos,
                                                                ,! this, _1),
                                                               ros::VoidPtr(),
                                                                ,! &this-
                                                                >rosQueue);

ros::SubscribeOptions so_forklift_left_servo =
 ,! ros::SubscribeOptions::create<std_msgs::Float32>("/" +
    this->model->GetName() +
    "/forklift_left_servo_position", 1,


                                                                ,!
                                                                    boost::bind(&Ser
                                                                voControl
                                                                ,!
                                                                    ::OnRosMsgForkli
                                                                ftLeftServo
                                                                ,! , this, _1),
                                                               ros::VoidPtr(),
                                                                ,! &this-
                                                                >rosQueue);

ros::SubscribeOptions so_forklift_right_servo =
 ,! ros::SubscribeOptions::create<std_msgs::Float32>("/" +
    this->model->GetName() +
    "/forklift_right_servo_position", 1,


                                                                ,!
                                                                    boost::bind(&Ser
                                                                voControl
                                                                ,!
                                                                    ::OnRosMsgForkli
                                                                ftRight
                                                                ,! Servo, this,
                                                                _1),
                                                               ros::VoidPtr(),
                                                                ,! &this-
                                                                >rosQueue);

ros::SubscribeOptions so_bottom_plate_servo =
 ,! ros::SubscribeOptions::create<std_msgs::Float32>("/" +
```

```
    this->model->GetName() +
    "/bottom_plate_servo_position", 1,
                                                    ,!
                                                        boost::bind(&Ser
                                                    voControl
                                                    ,!
                                                        ::OnRosMsgBottom
                                                    PlateServo,
                                                    ,! this, _1),
                                                   ros::VoidPtr(),
                                                    ,! &this-
                                                    >rosQueue);

ros::SubscribeOptions so_cube_grabber_servos =
 ,! ros::SubscribeOptions::create<std_msgs::Float32>("/" +
    this->model->GetName() +
    "/cube_grabber_servos_position", 1,
                                                    boost::bind(&ServoC
                                                    ontrol
                                                    ,!
                                                        ::OnRosMsgCubeGr
                                                    abberServos,
                                                    ,! this, _1),
                                                   ros::VoidPtr(),
                                                    ,! &this-
                                                    >rosQueue);

std::cout << "Initialized ROS nodes" << std::endl;

this->rosSubForkliftServos =
 ,! this->rosNode-
>subscribe(so_forklift_servos); this-
>rosSubForkliftLeftServo =
 ,! this->rosNode-
>subscribe(so_forklift_left_servo); this-
>rosSubForkliftRightServo =
 ,! this->rosNode-
>subscribe(so_forklift_right_servo); this-
>rosSubBottomPlateServo =
 ,! this->rosNode-
>subscribe(so_bottom_plate_servo); this-
>rosSubCubeGrabberServos =
 ,! this->rosNode->subscribe(so_cube_grabber_servos);

// start the queue helper thread
```

```
    this->rosQueueThread =
     std::thread(std::bind(&ServoControl::QueueThread, ,! this));


    std::cout << "Servo Plugin successfully loaded" << std::endl;

  }

 // handle incoming message from ROS
 public: void OnRosMsgForkliftServos(const std_msgs::Float32ConstPtr &_msg)
 {
/     sets the position to the value of the ROS topic
/     left servo top
   this->model->GetJointController()->SetVelocityTarget(model-
    >GetJoint("lbr_ ,! forklift_left::forklift_left_hook_top")-
    >GetScopedName(), (int) ,! _msg->data);
   // left servo bottom
   this->model->GetJointController()->SetVelocityTarget(model->GetJoint("lbr_
    ,! forklift_left::forklift_left_hook_bottom")->GetScopedName(), (int)
    ,! _msg->data * -1.0);
   // right servo top
   this->model->GetJointController()->SetVelocityTarget(model-
    >GetJoint("lbr_ ,! forklift_right::forklift_right_hook_top")-
    >GetScopedName(), (int) ,! _msg->data);

   // right servo bottom
   this->model->GetJointController()->SetVelocityTarget(model->GetJoint("lbr_
    ,! forklift_right::forklift_right_hook_bottom")->GetScopedName(), (int)
    ,! _msg->data * -1.0);
 }

 public: void OnRosMsgForkliftLeftServo(const std_msgs::Float32ConstPtr
 &_msg)
 {
/     sets the position to the value of the ROS topic
/     left servo top
   this->model->GetJointController()->SetVelocityTarget(model-
    >GetJoint("lbr_ ,! forklift_left::forklift_left_hook_top")-
    >GetScopedName(), (int) ,! _msg->data);

   // left servo bottom
   this->model->GetJointController()->SetVelocityTarget(model->GetJoint("lbr_
    ,! forklift_left::forklift_left_hook_bottom")->GetScopedName(), (int)
    ,! _msg->data * -1.0);
```

```
  }

  public: void OnRosMsgForkliftRightServo(const std_msgs::Float32ConstPtr
   ,! &_msg)
  {
    // right servo top
    this->model->GetJointController()->SetVelocityTarget(model->GetJoint("lbr_
     ,! forklift_right::forklift_right_hook_top")->GetScopedName(), (int)
     ,! _msg->data);
    // right servo bottom
    this->model->GetJointController()->SetVelocityTarget(model->GetJoint("lbr_
    forklift_right::forklift_right_hook_bottom")->GetScopedName(), (int)
     ,! _msg->data * -1.0);
  }

  public: void OnRosMsgBottomPlateServo(const std_msgs::Float32ConstPtr &_msg)
  {
    // sets the velocity to the value of the ROS topic

     ,! this->model->GetJointController()->SetVelocityTarget(model-
     >GetJoint("bottom ,! _plate_servo")->GetScopedName(), (int) _msg->data);
  }
  public: void OnRosMsgCubeGrabberServos(const std_msgs::Float32ConstPtr
  &_msg)
     {
       // sets the velocity to the value of the ROS topic this->model-
       >GetJointController()->SetVelocityTarget(model->GetJoint("lbr ,!
       _cube_grabber::cube_grabber_left_servo")->GetScopedName(), (int) ,!
       _msg->data * -1.0);

       // sets the velocity to the value of the ROS topic this->model-
       >GetJointController()->SetVelocityTarget(model->GetJoint("lbr ,!
       _cube_grabber::cube_grabber_right_servo")->GetScopedName(), (int) ,!
       _msg->data);

     }

/       helper function to process messages private: void QueueThread()
     {
       static const double timeout =
       0.01; while (this->rosNode-
       >ok())
       {
         this->rosQueue.callAvailable(ros::WallDuration(timeout));
       }
     }
```

```
/       Pointer to the update event connection
    private: event::ConnectionPtr updateConnection;

  };
  GZ_REGISTER_MODEL_PLUGIN(ServoControl)
}

#endif
```

D7.2 Actuator Controller -- plugin actuator.cc

```
#ifndef _PLUGIN_ACTUATOR_
#define _PLUGIN_ACTUATOR_

#include  <gazebo/gazebo.hh>
#include  <gazebo/physics/physics.hh>

#include  <thread>
#include  "ros/ros.h"
#include  "ros/callback_queue.h"
#include  "ros/subscribe_options.h"
#include  "std_msgs/Float32.h"
namespace gazebo
{
  class ActuatorControl : public ModelPlugin
  {
    public: ActuatorControl() : ModelPlugin()
      {
        printf("Initializing Actuator Plugin\n");
      }

    private: physics::ModelPtr model;                          // model
    pointer

    private: physics::JointPtr forklift_left_actuator_joint;      // forklift
    left
     ,! actuator joint pointer
    private: physics::JointPtr forklift_right_actuator_joint;     // forklift
     ,! right actuator joint pointer

    private: physics::JointPtr cube_grabber_actuator_middle_joint;       //
    cube
     ,! grabber actuator joint pointer
```

```
    private: physics::JointPtr cube_grabber_actuator_top_joint;       // cube
     ,! grabber actuator joint pointer

    private: common::PID cg_pid;                                      // PID
     ,! controller
    private: common::PID la_pid;                                      // PID
     ,! controller

    private: float initialPosition;                                  // target
    rotation

    private: std::unique_ptr<ros::NodeHandle> rosNode;               // ROS node

    private: ros::Subscriber rosSubForkliftLeftActuator;             // ROS
    topic
     ,! subscriber
    private: ros::Subscriber rosSubForkliftRightActuator;            // ROS
    topic
     ,! subscriber

    private: ros::Subscriber rosSubCubeGrabberActuatorMiddle;        // ROS
    topic
     ,! subscriber
    private: ros::Subscriber rosSubCubeGrabberActuatorTop;           // ROS topic
     ,! subscriber

    private: ros::CallbackQueue rosQueue;                            // ROS
    callback
     ,! queue
    private: std::thread rosQueueThread;                             // ROS queue
    thread

    public: void Load(physics::ModelPtr _model, sdf::ElementPtr _sdf)
      {
        std::cout << "Loading Actuator Plugin" << std::endl;
        //store pointer to the model
        this->model = _model;

/       check joint count
     if (_model->GetJointCount() == 0)
     {
       std::cout << "Plugin Actuator: Joint Count is 0, likely the plugin is
        ,! not loaded" << std::endl;
       return;
```

```
      }

      std::vector<boost::shared_ptr<gazebo::physics::Joint>> joints =
       ,! _model->GetJoints();
      for (int i=0; i<_model->GetJointCount(); i++)
      {
/          assign the joint for the left forklift actuator if (joints.at(i)-
>GetName() ==
        ,!"lbr_forklift_left::forklift_left_actuator")
        { this->forklift_left_actuator_joint = _model->GetJoints()[i]; }

/          assign the joint for the right forklift actuator
        if (joints.at(i)->GetName() ==
        ,! "lbr_forklift_right::forklift_right_actuator")
        { this->forklift_right_actuator_joint = _model->GetJoints()[i]; }

/          assign the joint for the middle cube grabber actuator if
(joints.at(i)->GetName() ==
        ,!"lbr_cube_grabber::cube_grabber_actuator_middle")
        { this->cube_grabber_actuator_middle_joint = _model->GetJoints()[i]; }

/          assign the joint for the top cube grabber actuator
        if (joints.at(i)->GetName() ==
        ,! "lbr_cube_grabber::cube_grabber_actuator_top")
        { this->cube_grabber_actuator_top_joint = _model->GetJoints()[i]; }
      }

      this->cg_pid = common::PID(10, 15.0, 1.0);            // set a PID
       ,! controller for the actuator
      this->la_pid = common::PID(1000, 15.0, 1.0);            // set a PID
       ,! controller for the actuator

      // add the PID controller to the joints this->model-
      >GetJointController()->SetPositionPID(this->forklift_left c ,!
      _actuator_joint->GetScopedName(),

       ,! this->la_pid);
       this->model->GetJointController()->SetPositionPID(this->forklift_right
       c ,! _actuator_joint->GetScopedName(),
       ,! this->la_pid);
      this->model->GetJointController()->SetPositionPID(this->cube_grabber c
       ,! _actuator_middle_joint->GetScopedName(),
       ,! this->cg_pid);
      this->model->GetJointController()->SetPositionPID(this->cube_grabber c
       ,! _actuator_top_joint->GetScopedName(),
       ,! this->cg_pid);
```

```
// set up ROS stuff
if (!ros::isInitialized())
{
  int argc = 0;
  char **argv = NULL;
  ros::init(argc, argv, "gazebo_client",
   ,! ros::init_options::NoSigintHandler);
}

this->rosNode.reset(new ros::NodeHandle("gazebo_client")); //
 create the ,! ROS node

// create a ROS topic and subscribe to it
ros::SubscribeOptions so_forklift_left_actuator =
 ,! ros::SubscribeOptions::create<std_msgs::Float32>("/" +
    this->model->GetName() +
    "/forklift_left_actuator_position", 1,

                                                        ,!
                                                           boost::bind(&Act
                                                        uatorControl
                                                        ,!
                                                           ::OnRosMsgForkli
                                                        ftLeft
                                                        ,! Actuator, this,
                                                        ,! _1),
                                                       ros::VoidPtr(),
                                                        ,! &this-
                                                        >rosQueue);

ros::SubscribeOptions so_forklift_right_actuator =
 ,! ros::SubscribeOptions::create<std_msgs::Float32>("/" +
    this->model->GetName() +
    "/forklift_right_actuator_position", 1,

                                                        ,!
                                                           boost::bind(&Act
                                                        uatorControl
                                                        ,!
                                                           ::OnRosMsgForkli
                                                        ftRight
                                                        ,! Actuator, this,
```

```
                                                                    ,! _1),
                                                                   ros::VoidPtr(),
                                                                    ,! &this-
                                                                    >rosQueue);
                                                                    ros::SubscribeOptio
                                                                    ns
                                                                    so_cube_grabber_act
                                                                    uator_middle =
 ,! ros::SubscribeOptions::create<std_msgs::Float32>("/" +
    this->model->GetName() +
    "/cube_grabber_actuator_middle_position", 1,

                                                                    ,!
                                                                        boost::bind(&Act
                                                                    uatorControl
                                                                    ,!
                                                                        ::OnRosMsgCubeGr
                                                                    abber
                                                                    ,! ActuatorMiddle,
                                                                    ,! this, _1),
                                                                   ros::VoidPtr(),
                                                                    ,! &this-
                                                                    >rosQueue);

ros::SubscribeOptions so_cube_grabber_actuator_top =
 ,! ros::SubscribeOptions::create<std_msgs::Float32>("/" +
    this->model->GetName() +
    "/cube_grabber_actuator_top_position", 1,

                                                                    ,!
                                                                        boost::bind(&Act
                                                                    uatorControl
                                                                    ,!
                                                                        ::OnRosMsgCubeGr
                                                                    abber
                                                                    ,! ActuatorTop,
                                                                    this,
                                                                    ,! _1),
                                                                   ros::VoidPtr(),
                                                                    ,! &this-
                                                                    >rosQueue);

this->rosSubForkliftLeftActuator =
```

```
      ,! this->rosNode-
     >subscribe(so_forklift_left_actuator); this-
     >rosSubForkliftRightActuator =
      ,! this->rosNode-
     >subscribe(so_forklift_right_actuator); this-
     >rosSubCubeGrabberActuatorMiddle =
      ,! this->rosNode-
     >subscribe(so_cube_grabber_actuator_middle); this-
     >rosSubCubeGrabberActuatorTop =
      ,! this->rosNode->subscribe(so_cube_grabber_actuator_top);

/       start the queue helper thread this->rosQueueThread =
     ,!std::thread(std::bind(&ActuatorControl::QueueThread, this));

     std::cout << "Actuator Plugin successfully loaded" << std::endl;
   }

 // handle incoming message from ROS
 public: void OnRosMsgForkliftLeftActuator(const std_msgs::Float32ConstPtr
  ,! &_msg)
 {
/     sets the position to the value of the ROS topic this->model-
>GetJointController()->SetPositionTarget(
          this->forklift_left_actuator_joint->GetScopedName(), _msg->data);
          }

    public: void OnRosMsgForkliftRightActuator(const
    std_msgs::Float32ConstPtr
     ,! &_msg)
    {
/        sets the position to the value of the ROS topic this->model-
>GetJointController()->SetPositionTarget(
             this->forklift_right_actuator_joint->GetScopedName(), _msg-
             >data);
    }

    public: void OnRosMsgCubeGrabberActuatorMiddle(const
     ,! std_msgs::Float32ConstPtr &_msg)
    {
/        sets the position to the value of the ROS topic this->model-
>GetJointController()->SetPositionTarget(
             this->cube_grabber_actuator_middle_joint->GetScopedName(),
              ,! _msg->data);
    }
```

```
    public: void OnRosMsgCubeGrabberActuatorTop(const
    std_msgs::Float32ConstPtr
     ,! &_msg)
    {
/         sets the position to the value of the ROS topic this->model-
>GetJointController()->SetPositionTarget(
              this->cube_grabber_actuator_top_joint->GetScopedName(),
               ,! _msg->data);
    }

/       helper function to process messages private: void QueueThread()
    {
      static const double timeout =
      0.01; while (this->rosNode-
      >ok())
      {
        this->rosQueue.callAvailable(ros::WallDuration(timeout));
      }
    }

/       Pointer to the update event connection
    private: event::ConnectionPtr updateConnection;

  };
  GZ_REGISTER_MODEL_PLUGIN(ActuatorControl)
}

#endif
```

D7.3 Robot Spawner -- plugin spawner.cc

```
#include  <ignition/math/Pose3.hh>
#include  "gazebo/physics/physics.hh"
#include  "gazebo/common/common.hh"
#include  <gazebo/gazebo.hh>

namespace gazebo
{
  class ObjectSpawner : public WorldPlugin
  {
    public: ObjectSpawner() : WorldPlugin()
      {
        printf("Loaded Spawner Plugin\n");
      }
```

```
    public: void Load(physics::WorldPtr _parent, sdf::ElementPtr)
      {
        // initially spawn the LBR Robot
        _parent ->
        InsertModelFile("model://lbr_robot");
        printf("Spawned LBR\n");
      }
  };
  GZ_REGISTER_WORLD_PLUGIN(ObjectSpawner)
}
```

D8 Simulation Description Format (SDF) Files

D8.1 Robot (Overall) -- lbr robot.sdf

```
<?xml version= 1.0 ?>
<sdf version= 1.6 >
  <model name= lbr_robot
                        >
  <static>false</static>
  <!-- MAIN CHASSIS -->
  <link name= chassis >
       <pose>0 0 .05 0 0 0</pose>
       <collision name= collision ><geometry><box><size>.8
       .8 ,! .02</size></box></geometry></collision>
       <visual name= visual >
         <pose>-.79 -.5 -.54 1.57 0 1.57</pose>
         <geometry>
           <mesh>
             <uri>file://lbr_robot/lbr_chassis.stl</uri>
             <scale>0.004 0.004 0.004</scale>
          </mesh>
         </geometry>
       </visual>
       <inertial><mass>20</mass></inertial>
 </link>

 <!-- LEFT FORKLIFT -->
 <include>
   <uri>model://lbr_forklift_left</uri>
   <pose>-.45 0 .3 0 0 0</pose>
 </include>

 <!-- RIGHT FORKLIFT    -->
```

```
 <include>
   <uri>model://lbr_forklift_right</uri>
   <pose>.45 0 .3 0 0 0</pose>
 </include>

 <!-- CUBE GRABBER -->
 <include>
   <uri>model://lbr_cube_grabber</uri>
   <pose>0 .3 .4 -1.57 0 0</pose>
 </include>
 <joint name="cube_grabber" type="fixed">
   <parent>chassis</parent>
   <child>lbr_cube_grabber::cube_grabber_base</child>
 </joint>

 <!-- BOTTOM PLATE -->
 <include>
   <uri>model://lbr_bottom_plate</uri>
   <pose>0 0 0 0 0 0</pose>
 </include>
 <joint name="bottom_plate_servo" type="revolute">
   <pose>0 0 .05 0 0 0</pose>
   <parent>chassis</parent>
   <child>lbr_bottom_plate::bottom_plate</child>
   <axis>
     <limit>
       <lower>0.0</lower><upper>0.785</upper>
     </limit>
     <xyz>0 0 1</xyz>
   </axis>
 </joint>

 <!-- add the servo control plugin -->
 <plugin name="plugin_servo" filename="libplugin_servo.so"/>

 <!-- add the actuator control plugin -->
 <plugin name="plugin_actuator" filename="libplugin_actuator.so"/>
 </model>
</sdf>
```

D8.2 Robot (Lifting and Climbing Mechanism) -- lbr forklift left.sdf

```
<?xml version= 1.0 ?>
<sdf version= 1.6 >
```

```
<model name= lbr_forklift_left >
  <static>false</static>

  <!-- Base Lift Plate -->
  <link name= forklift_plate >
    <pose>.041 .109 .6 0 0 -3.14159</pose>
    <inertial>
      <pose>0 0 0 0 0 0</pose>
      <inertia>
        <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iyy
         >0< c ,! /iyy><izz>0</izz>
      </inertia>
      <mass>1.0</mass>
    </inertial>
    <collision name= collision
    ><geometry><box><size>.2 .6 ,!
    .35</size></box></geometry></collision> <visual
    name= visual >
      <pose>-1.178 -.308 -2.65 1.57 0
      1.57</pose> <geometry>
        <mesh>
          <uri>file://lbr_forklift_left/lbr_lifter_slide.stl</uri>
          <scale>0.004 0.004 0.004</scale>
      </mesh>
      </geometry>
    </visual>
  </link>

  <!-- Forklift Hook TOP -->
  <link name= forklift_hook_top >
    <pose>-2.795 .3648 -.5130 0 0 -1.57</pose>
    <inertial>
      <pose>0 0 0 0 0 0</pose>
      <inertia>
        <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iyy
         >0< c ,! /iyy><izz>0</izz>
      </inertia>
      <mass>.01</mass>
    </inertial>
    <collision name= collision >
    <geometry>
     <mesh>
       <uri>file://lbr_forklift_left/lbr_hook.stl</uri>
       <scale>0.004 0.004 0.004</scale>
     </mesh>
```

```
    </geometry>
  </collision>
  <visual name= visual >
    <geometry>
      <mesh>
        <uri>file://lbr_forklift_left/lbr_hook.stl</uri>
        <scale>0.004 0.004 0.004</scale>
      </mesh>
    </geometry>
  </visual>
</link>

<!-- Forklift Hook BOTTOM -->
<link name= forklift_hook_bottom >
  <pose>-2.797 -.3648 1.7070 3.14 0 1.57</pose>
  <inertial>
    <pose>0 0 0 0 0 0</pose>
    <inertia>
      <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iyy>0< c
       ,! /iyy><izz>0</izz>
    </inertia>
    <mass>.01</mass>
  </inertial>
  <collision name= collision >
    <geometry>
      <mesh>
        <uri>file://lbr_forklift_left/lbr_hook.stl</uri>
        <scale>0.004 0.004 0.004</scale>
      </mesh>
    </geometry>
  </collision>
  <visual name= visual >
    <geometry>
      <mesh>
        <uri>file://lbr_forklift_left/lbr_hook.stl</uri>
        <scale>0.004 0.004 0.004</scale>
      </mesh>
    </geometry>
  </visual>
</link>

<!-- JOINTS -->
   <!-- forklift plate to forklift fork, servo rotator TOP --
   > <joint name="forklift_left_hook_top" type="revolute">
     <pose>.5 2.777 1.242 0 0 0</pose>
```

```
      <parent>forklift_plate</parent>
      <child>forklift_hook_top</child>
      <axis>
        <limit>
          <lower>0</lower><upper>1.57</upper>
        </limit>
        <xyz>1 0 0</xyz>
      </axis>
    </joint>

    <!-- forklift plate to forklift fork, servo rotator BOTTOM --
    > <joint name="forklift_left_hook_bottom" type="revolute">
      <pose>.22 2.777 1.238 3.14 0 0</pose>
      <parent>forklift_plate</parent>
      <child>forklift_hook_bottom</child>
      <axis>
        <limit>
          <lower>0</lower><upper>1.57</upper>
        </limit>
        <xyz>1 0 0</xyz>
      </axis>
    </joint>

    <!-- forklift tower to forklift plate, linear actuator -->
    <joint name="forklift_left_actuator" type="prismatic">
      <pose>-.15 .1 .2 0 0 0</pose>
      <parent>chassis</parent>
      <child>forklift_plate</child>
      <axis>
        <limit>
          <lower>0</lower>
          <upper>1.8</upper>
          <velocity>.0000001</velocity>
        </limit>
        <dynamics><damping>10</damping></dynami
        cs> <xyz>0 0 5</xyz>
      </axis>
    </joint>


  </model>
</sdf>
```

D8.3 Robot (Lifting and Climbing Mechanism) -- lbr forklift right.sdf

```
<?xml version= 1.0 ?>
<sdf version= 1.6 >
  <model name= lbr_forklift_right >
    <static>false</static>

    <!-- Base Lift Plate -->
    <link name= forklift_plate >
      <pose>-.061 -.103 .6 0 0 3.14159</pose>
      <inertial>
        <pose>0 0 0 0 0 0</pose>
        <inertia>
          <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iy
           y>0< c ,! /iyy><izz>0</izz>
        </inertia>
        <mass>1.0</mass>
      </inertial>
      <collision name= collision
      ><geometry><box><size>.2 .6 ,!
      .35</size></box></geometry></collision> <visual
      name= visual >
        <pose>1.178 .308 -2.65 1.57 0 -
        1.57</pose> <geometry>
          <mesh>
            <uri>file://lbr_forklift_right/lbr_lifter_slide.stl</uri>
            <scale>0.004 0.004 0.004</scale>
       </mesh>
        </geometry>
      </visual>
    </link>

    <!-- Forklift Hook TOP -->
    <link name= forklift_hook_top >
      <pose>2.774 -.360 -.513 0 0 1.57</pose>
      <inertial>
        <pose>0 0 0 0 0 0</pose>
        <inertia>
          <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iy
           y>0< c ,! /iyy><izz>0</izz>
        </inertia>
        <mass>.01</mass>
      </inertial>
      <collision name= collision >
```

```
      <geometry>
        <mesh>
          <uri>file://lbr_forklift_right/lbr_hook.stl</uri>
          <scale>0.004 0.004 0.004</scale>
          </mesh>
    </geometry>
  </collision>
  <visual name= visual >
    <geometry>
      <mesh>
        <uri>file://lbr_forklift_right/lbr_hook.stl</uri>
        <scale>0.004 0.004 0.004</scale>
      </mesh>
    </geometry>
  </visual>
</link>

<!-- Forklift Hook BOTTOM -->
<link name= forklift_hook_bottom >
  <pose>2.775 .3684 1.7070 3.14 0 -1.57</pose>
  <inertial>
    <pose>0 0 0 0 0 0</pose>
    <inertia>
      <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iyy>0< c
       ,! /iyy><izz>0</izz>
    </inertia>
    <mass>.01</mass>
  </inertial>
  <collision name= collision >
    <geometry>
      <mesh>
        <uri>file://lbr_forklift_right/lbr_hook.stl</uri>
        <scale>0.004 0.004 0.004</scale>
      </mesh>
    </geometry>
  </collision>
  <visual name= visual >
    <geometry>
      <mesh>
        <uri>file://lbr_forklift_right/lbr_hook.stl</uri>
        <scale>0.004 0.004 0.004</scale>
      </mesh>
    </geometry>
  </visual>
```

```
</link>

<!-- JOINTS -->

<!-- forklift plate to forklift fork, servo rotator TOP -->
<joint name="forklift_right_hook_top" type="revolute">
  <pose>.5 2.777 1.242 0 0 0</pose>
  <parent>forklift_plate</parent>
     <child>forklift_hook_top</child>
     <axis>
       <limit>
         <lower>0</lower><upper>1.57</upper>
       </limit>
       <xyz>1 0 0</xyz>
     </axis>
   </joint>


   <!-- forklift plate to forklift fork, servo rotator BOTTOM --
   > <joint name="forklift_right_hook_bottom" type="revolute">
     <pose>.22 2.777 1.238 3.14 0 0</pose>
     <parent>forklift_plate</parent>
     <child>forklift_hook_bottom</child>
     <axis>
       <limit>
         <lower>0</lower><upper>1.57</upper>
       </limit>
       <xyz>1 0 0</xyz>
     </axis>
   </joint>



   <!-- forklift tower to forklift plate, linear actuator -->
   <joint name="forklift_right_actuator" type="prismatic">
     <pose>.15 -.1 .2 0 0 0</pose>
     <parent>chassis</parent>
     <child>forklift_plate</child>
     <axis>
       <limit>
         <lower>0</lower>
         <upper>1.8</upper>
         <velocity>.0000001</velocity>
       </limit>
       <dynamics><damping>10</damping></dynami
       cs> <xyz>0 0 1</xyz>
     </axis>
```

```
    </joint>

  </model>
</sdf>
```

D8.4 Robot (Cube Latch Mechanism) -- lbr cube grabber.sdf

```
<?xml version= 1.0 ?>
<sdf version= 1.6 >
<model name= lbr_cube_grabber >
  <static>false</static>
  <!-- Cube Grabber Actuator Base -->
  <link name= cube_grabber_base >
    <pose>0 -.20 0 0 0 0</pose>
    <inertial>
      <pose>0 0 0 0 0 0</pose>
      <inertia>
        <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iyy>0<
         c ,! /iyy><izz>0</izz>
      </inertia>
      <mass>0.5</mass>
    </inertial>
    <visual name= visual ><geometry><box><size>.01
     .01 ,! .01</size></box></geometry></visual>
  </link>

  <!-- Cube Grabber Actuator Extension MIDDLE -->
  <link name= cube_grabber_extender_middle >
    <pose>-.01 -.2 -.27 0 0 0</pose>
    <inertial>
      <pose>0 0 0 0 0 0</pose>
      <inertia>
        <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iyy>0<
         c ,! /iyy><izz>0</izz>
      </inertia>
      <mass>.5</mass>
    </inertial>
    <collision name= collision ><geometry><box><size>.06 .06
    ,! .88</size></box></geometry></collision> <visual name=
    visual >
      <pose>-.778 1.095 -.495 1.57 -1.57
      1.57</pose> <geometry>
        <mesh>
          <uri>file://lbr_cube_grabber/lbr_cascade_shaft.stl</uri>
          <scale>0.004 0.004 0.004</scale>
```

```
      </mesh>
     </geometry>
   </visual>
 </link>

 <!-- Cube Grabber Actuator Extension TOP -->
 <link name= cube_grabber_extender_top >
   <pose>-.01 -.262 -.24 0 0 0</pose>
   <inertial>
     <pose>0 0 0 0 0 0</pose>
     <inertia>
     <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iyy>0< c ,!
     /iyy><izz>0</izz>
    </inertia>
    <mass>.5</mass>
  </inertial>
  <collision name= collision ><geometry><box><size>.06 .06
  ,! .88</size></box></geometry></collision> <visual name=
  visual >
    <pose>-.778 1.095 -.495 1.57 -1.57 1.57</pose>
    <geometry>
      <mesh>
        <uri>file://lbr_cube_grabber/lbr_cascade_shaft.stl</uri>
        <scale>0.004 0.004 0.004</scale>
      </mesh>
    </geometry>
  </visual>
</link>


<!-- Cube Grabber Endfixture -->
<link name= cube_grabber_endfixture >
  <pose>-.01 -.30 .05 0 0 0</pose>
  <inertial>
    <pose>0 0 0 0 0 0</pose>
    <inertia>
      <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iyy>0< c
       ,! /iyy><izz>0</izz>
    </inertia>
    <mass>.5</mass>
  </inertial>
  <collision name= collision ><geometry><box><size>.875 .02
  ,! .3</size></box></geometry></collision> <visual name=
  visual >
```

```
    <pose>-.7780 1.265 -.685 3.14 -1.57</pose>
    <geometry>
      <mesh>
        <uri>file://lbr_cube_grabber/lbr_cube_grabber_endpiece.stl</uri>
        <scale>0.004 0.004 0.004</scale>
      </mesh>
    </geometry>
  </visual>
</link>

<!-- Cube Grabber Claw -->
<link name= cube_grabber_claw_left >
  <pose> .025 .95 .788 3.14 1.57 0</pose>
  <inertial>
  <inertia>
      <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iyy>0< c
       ,! /iyy><izz>0</izz>
    </inertia>
    <mass>.01</mass>
  </inertial>
  <collision name= collision >
    <geometry>
      <mesh>
        <uri>file://lbr_cube_grabber/lbr_cube_grabber_claw_left.stl</uri>
        <scale>0.004 0.004 0.004</scale>
      </mesh>
    </geometry>
  </collision>
  <visual name= visual >
    <pose>0 0 0 0 0 0</pose>
    <geometry>
      <mesh>
        <uri>file://lbr_cube_grabber/lbr_cube_grabber_claw_left.stl</uri>
        <scale>0.004 0.004 0.004</scale>
      </mesh>
    </geometry>
  </visual>
</link>

<!-- Cube Grabber Claw -->
<link name= cube_grabber_claw_right >
  <pose> 1.52 .95 .788 -3.14 1.57 0</pose>
  <inertial>
    <inertia>
      <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iyy>0< c
```

```
      ,! /iyy><izz>0</izz>
    </inertia>
    <mass>.01</mass>
  </inertial>
  <collision name= collision >
    <geometry>
      <mesh>
        <uri>file://lbr_cube_grabber/lbr_cube_grabber_claw_right.stl</uri>
        <scale>0.004 0.004 0.004</scale>
      </mesh>
    </geometry>
  </collision>
  <visual name= visual >
    <pose>0 0 0 0 0 0</pose>
    <geometry>
      <mesh>
      <uri>file://lbr_cube_grabber/lbr_cube_grabber_claw_right.stl</uri>
      <scale>0.004 0.004 0.004</scale>
      </mesh>
    </geometry>
  </visual>
</link>

<!-- JOINTS -->

<!-- forklift plate to forklift fork, servo rotator -->
<joint name="cube_grabber_left_servo" type="revolute">
  <pose>.713 1.05 .4077 0 0 0</pose>
  <parent>cube_grabber_endfixture</parent>
  <child>cube_grabber_claw_left</child>
  <axis>
    <limit>
      <lower>-3.142</lower><upper>0</upper>
    </limit>
    <xyz>0 0 1</xyz>
  </axis>
</joint>

<!-- forklift plate to forklift fork, servo rotator -->
<joint name="cube_grabber_right_servo" type="revolute">
  <pose>.713 1.05 1.156 0 0 0</pose>
  <parent>cube_grabber_endfixture</parent>
  <child>cube_grabber_claw_right</child>
  <axis>
    <limit>
```

```
      <lower>0</lower><upper>3.142</upper>
    </limit>
    <xyz>0 0 1</xyz>
  </axis>
</joint>

<!-- cube grabber base to cube grabber extender, linear actuator -->
<joint name="cube_grabber_actuator_middle" type="prismatic">
  <pose>0 0 -.5 0 0 0</pose>
  <parent>chassis</parent>
  <child>cube_grabber_extender_middle</child>
  <axis>
    <limit>
      <lower>0</lower>
      <upper>.6</upper>
    </limit>
    <dynamics><damping>10</damping></dynamics>
    <xyz>0 -1 0</xyz>
    </axis>
   </joint>

   <!-- cube grabber base to cube grabber extender, linear actuator -->
   <joint name="cube_grabber_actuator_top" type="prismatic">
     <pose>0 0 -.5 0 0 0</pose>
     <parent>cube_grabber_extender_middle</pare
     nt>
     <child>cube_grabber_extender_top</child>
     <axis>

       <limit>
         <lower>0</lower>
         <upper>.6</upper>
       </limit>
       <dynamics><damping>10</damping></dynami
       cs> <xyz>0 -1 0</xyz>
     </axis>
   </joint>

   <!-- cube grabber actuator to cube grabber end fixture, fixed -->
   <joint name="cube_grabber_endfixture_joint" type="fixed">
     <pose>0 0 .2 0 0 0</pose>
     <parent>cube_grabber_extender_top</pare
     nt>
     <child>cube_grabber_endfixture</child>
   </joint>
```

```
  </model>
</sdf>
```

D8.5 Robot (Rotating Plate Mechanism) -- lbr bottom plate.sdf

```
<?xml version= 1.0 ?>
<sdf version='1.6'>
  <model name='lbr_bottom_plate'>
    <static>false</static>

    <!-- Bottom Plate -->
    <link name='bottom_plate'>
      <pose>-.005 0 0 0 0 0</pose>
      <inertial>
        <pose>0 0 0 0 0 0</pose>
        <inertia>

<ixx>.51</ixx><iyy>.51</iyy><izz>.51</izz><ixx>0</ixx><iyy>0</iyy><izz>0<
/izz>
        </inertia>
        <mass>2</mass>
      </inertial>
      <collision name='collision'><geometry><box><size>.8 .8
.02</size></box></geometry></collision>
      <visual name='visual'>
      <pose>-.78 -.5 -.487 1.57 0 1.57</pose>
        <geometry>
          <mesh>
            <uri>file://lbr_bottom_plate/lbr_rotating_plate.stl</uri>
            <scale>0.004 0.004 0.004</scale>
          </mesh>
        </geometry></visual>
    </link>

  </model>
</sdf>
```

D8.6 Cube -- lbr cube.sdf

```
<?xml version= 1.0 ?>
<sdf version= 1.6 >
  <model name= lbr_cube >
    <static>false</static>

    <!-- CUBE -->
    <link name= cube >
      <pose>0 0 0 1.57 0 1.57</pose>
      <inertial>
        <pose>.6 .6 .6 0 0 0</pose>
        <inertia>
          <ixx>.01</ixx><iyy>.01</iyy><izz>.01</izz><ixx>0</ixx><iyy>
           0< c ,! /iyy><izz>0</izz>
        </inertia>
        <mass>.05</mass>
      </inertial>
      <collision name= collision >
        <geometry>
          <mesh>
            <uri>file://lbr_cube/cube.stl</uri>
            <scale>0.004 0.004 0.004</scale>
         </mesh>
        </geometry>
      </collision>
      <visual name= visual >
       <geometry>
          <mesh>
            <uri>file://lbr_cube/cube.stl</uri>
            <scale>0.004 0.004 0.004</scale>
         </mesh>
        </geometry>
      </visual>
    </link>
  </model>
</sdf>
```