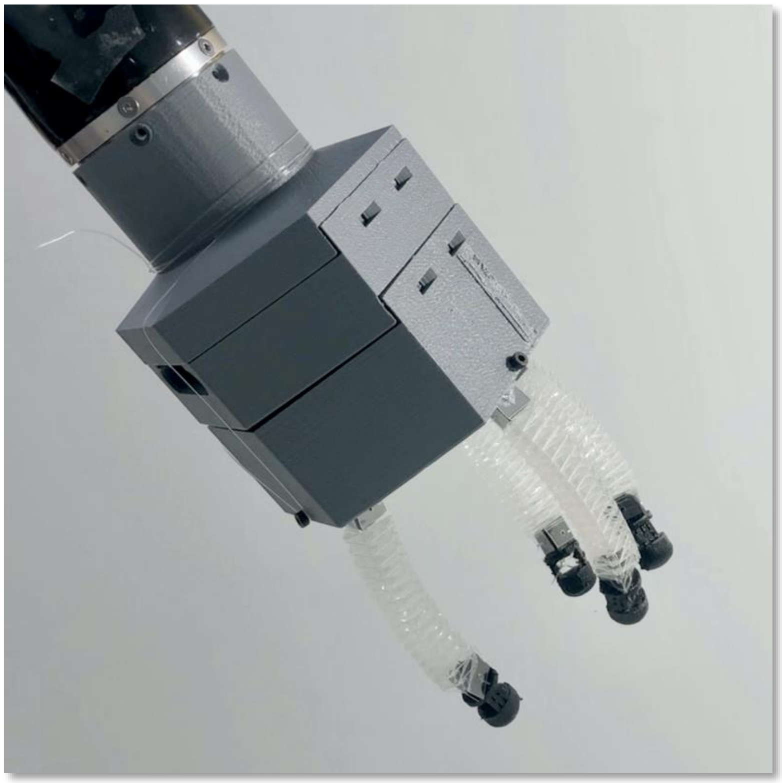
Flexible Robotic Origami Gripper (FROG)



A Major Qualifying Project Report
submitted to the Faculty of the WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

BY:
Anna Mederer (RBE/ME)
Maria Medina Martinez (CS)
Selina Spry (RBE/ECE)

ADVISORS:
Professor Çağdaş Önal
Professor Berk Çallı
Professor Jie Fu

DATE: May 6th, 2021

Abstract

Origami robots have been introduced as a new soft robotic technology that can be easily constructed from planar sheets of material to create rigid or deformable linkages. With this Major Qualifying Project, the team investigated the capabilities of a modular soft robotic gripper with triangular beam and Yoshimura origami finger designs folded from PET plastic. Due to its origami design, the gripper is lightweight, flexible, and durable. A vision system working in conjunction with an impedance controller aided the gripper to determine and provide three stable grasp patterns to pick up an assortment of objects. The implementation of a switching control system has demonstrated the need for more adaptive control for this compliant gripper. This project provides a foundation for future research into origami grippers.

Executive Summary

Origami-inspired flexible robots are a subset of soft robots, therefore they differ from rigid robots in aspects including material, control, and potential applications. By leveraging origami folds, flat substrates are able to form both rigid and flexible 3D structures, such as stiff linkages and actuated hinges [1]. Printable origami robots enable the creation of novel robotic structures that are adaptable and modular. The research of this project was greatly influenced by work completed by WPI Soft Robotics Lab, including the Yoshimura origami modules and the 3D-printed prosthetic hand [2].

The goal of this project was to create a flexible robotic origami gripper capable of recognizing and grasping different objects. Throughout this MQP, we aimed to analyze the benefits of having origami soft materials and mechanisms utilized within the gripper, investigate means to control the movement and force output of the gripper, and examine the potential of utilizing computer vision to augment the gripper's capabilities. Furthermore, we intend to study the capabilities and benefits of origami robots to create a low-cost gripper that is both lightweight and modular.

The design of the gripper leveraged two origami finger designs, the triangular beam and Yoshimura. These fingers are operated to form four total grasp configurations: open, pinch, power, and tripod. The triangular beam design is inspired by the hexapod robot [3] and provides rigid links with flexible joints. In contrast, the Yoshimura pattern is inspired by traditional Yoshimura folds, which are highly flexible axially and very strong torsionally [4]. Due to the palm's modular design, the origami fingers can be easily interchanged. A switching hybrid controller was developed to provide a more adaptive control scheme. With a hybrid controller, the robot is able to utilize position control in free space until it interacts with an object. Once it senses an object, the gripper relies on force control to ensure a stable grasp. In addition, the team developed a simulation that demonstrates the trajectory of the joints in the triangular beam finger. For the vision system of the gripper, the team used Principal Component Analysis (PCA) to recognize an object and then the identified grasp is received and executed by the gripper. Overall, the system uses ROS to communicate between components.

A significant result of our research determined that the grasp patterns executed by the Yoshimura fingers better conformed to the objects as it leveraged a continuum design. We also identified PCA to be a feasible method of object detection and utilizing ROS further supported a modular system. Based on the results of our project, we determined a number of recommendations that could improve this project in the future. There is potential to further experiment with different joint stiffnesses along the triangular beam finger. This would allow for a better closing sequence of the triangular beam finger for grasping objects. Additionally, we recommend an exploration of Yoshimura and triangular beam finger combinations to better understand if certain origami fingers are better for specific finger types. Furthermore, there is potential to improve upon our control scheme as well as the detection of small and irregular objects.

As demonstrated through our MQP, origami modules are a viable structure for constructing practical and lightweight robotic grippers. The completion of this project provides a foundation for future research into robotic grippers that takes advantage of the benefits of origami soft materials.

Acknowledgments

The FROG MQP Team would like to thank the following people for guiding and supporting us in the completion of this project:

- WPI Soft Robotics Lab for graciously lending support and materials to the team as well as their lab space.
- Yinan Sun for his expertise and mentorship throughout the duration of this project.
- Ashley Chu and Shou-Shan Chiang for their advice and knowledge in relation to the computer vision and Jaco arm aspects of this project.
- Ann Marie Votta for her research into prosthetics and force sensing.
- Mason Mitchell for his assistance with the magnetic force sensors.
- Erik Skorina and Tyler Looney for 3D printing various components of our origami robotic gripper.
- The National Science Fund for the REU that provided a foundation for our research into robotic origami grippers.

Finally, thank you to our project advisors Professor Önal, Professor Çallı, and Professor Fu for their time and assistance. Their dedication to our team's success was greatly appreciated and their continual advice was invaluable.

Table of Contents

List of Figures

List of Tables

1. Introduction

Robots have impacted various industries as they have been used to increase efficiency, precision, and mobility in a number of applied settings. Throughout the 20th century, robots were used primarily in automation to perform monotonous labor [5]. As technology advances, robotics has expanded beyond manufacturing facilities and into areas such as exploration, medical devices, etc. Current robotics research has also grown to include developments in the area of soft robotics.

Soft robots differ from rigid robots in aspects including material, control, and application potential. Soft materials have been manipulated to create novel robotic structures such as printable origami robots. Printable robots can leverage origami folds to achieve either rigid or flexible 3D structures from a flat substrate [1]. These folded modules are then able to form shapes ranging from stiff linkages to actuated hinges. Origami robots have been used for medical device applications, morphable wheels, actuated solar panels, and origami exoskeletons [6].

The majority of research into the field of origami robotics has been limited to niche applications. However, this research has provided a foundational understanding of the properties of origami robots. For instance in Professor Çağdaş Önal's research with WPI Soft Robotics Lab, his printable origami robots utilize the Yoshimura fold pattern. This pattern provides high torsional stiffness while also being highly flexible axially [4]. These origami structures can also ensure a better payload to robot weight ratio because of the characteristics of the fold itself. Since the robots are created with origami folds, the structure can collapse when needed for easy storage and deployability. Furthermore, a key benefit of these origami robots is the quick and straightforward fabrication process.

Almost all commercial and prototyped grippers rely on rigid materials to grasp various objects. These robotic grippers tend to be heavier, more expensive, and less flexible than what certain grasping applications require. However, no gripper currently exists that leverages the benefits of origami mechanisms to effectively grasp an object. Soft robotic origami grippers provide flexible and deformable characteristics that are advantageous for grasping and conforming to the shape of desired objects.

Flexible Robotic Origami Gripper (FROG) is a student proposed Major Qualifying Project (MQP) and is in line with the research plans of WPI Soft Robotics Lab. The goal of this project to explore the incorporation of origami folds within a gripper as well as the potential relationship with object recognition. Additionally, this project establishes a comparative analysis of two types of origami-inspired fingers utilizing a single modular palm. Each configuration has the capability of performing three types of grasp patterns determined by the robot vision system. The vision system, which leverages Principal Component Analysis (PCA) to recognize the objects, will be integrated with the gripper's switching controller to achieve the desired grasps. Additionally, a simulation of the gripper will be developed to visualize the output of the controller. This analysis will investigate the benefits of using origami robotics in the design of a gripper while addressing some of the difficulties of controlling compliant robots. Ultimately, this project serves as a foundation for future teams interested in exploring soft origami grippers.

2. Background

2.1 Origami Folding

Basic polyhedra, such as cubes and prisms, can be folded from flat sheets to act as rigid linkages in an origami robot [7]. By connecting two folded structures along a folded edge, these linkages can be combined to form simple joints [7]. In WPI's Soft Robotics Lab, a hexapod robot was designed from rigid triangular beams and flexural revolute joints [3]. With more complex origami patterns, components can attain compressed and bending configurations, which allows origami robots with these components to move more smoothly through the task space. Such origami patterns include the waterbomb base pattern which expands both radially and axially, the diagonal pattern which results in rotational motion, and the Yoshimura pattern which provides axial movement [1].

The Yoshimura pattern is created by making a series of horizontal valley folds and diagonal mountain folds. The angle of the diagonal in the Yoshimura pattern can be adjusted to obtain a stable structure of the desired axial length [8]. However, there can be plane deformation depending on the stiffness of the material, which results in the ability to compress the folded structure [8]. The Yoshimura pattern is employed in an origami continuum manipulator module designed in WPI Soft Robotics Lab [4]. The origami module has one axial degree of freedom and two bending degrees of freedom and is distinctive due to its high torsional stiffness in relation to its weight [4]. The desired properties for an origami gripper can be configured by making use of specific origami folding patterns.

2.2 Human Hand Biomechanics

Bioinspired grippers derive key aspects of their performance from the biomechanics of the human hand. The second through fifth fingers of the hand consist of proximal, intermediate, and distal phalanges with the metacarpophalangeal joint at the base of the finger and two interphalangeal joints, shown in Figure 1. The hand is generally modeled as an open kinematic chain from wrist to the fingertips with a total of over 20 degrees of freedom [9]. The interphalangeal joints are represented as one rotational degree of freedom and the metacarpophalangeal joints are represented as two rotational degrees of freedom [9]. The joints allow hands to execute a variety of grasps to perform tasks. Daily activities such as food preparation, personal care, and housekeeping utilize nine types of grasps [10]. These common grasps include the cylindrical grasp where fingers curl around a cylinder, the pinch grasp where the fingertips of the thumb and index squeeze a small object, and the hook grasp which uses the second through fifth fingers to pull a loop protrusion of an object [10]. Implementing a variety of grasps allows a gripper to grasp a larger assortment of objects.

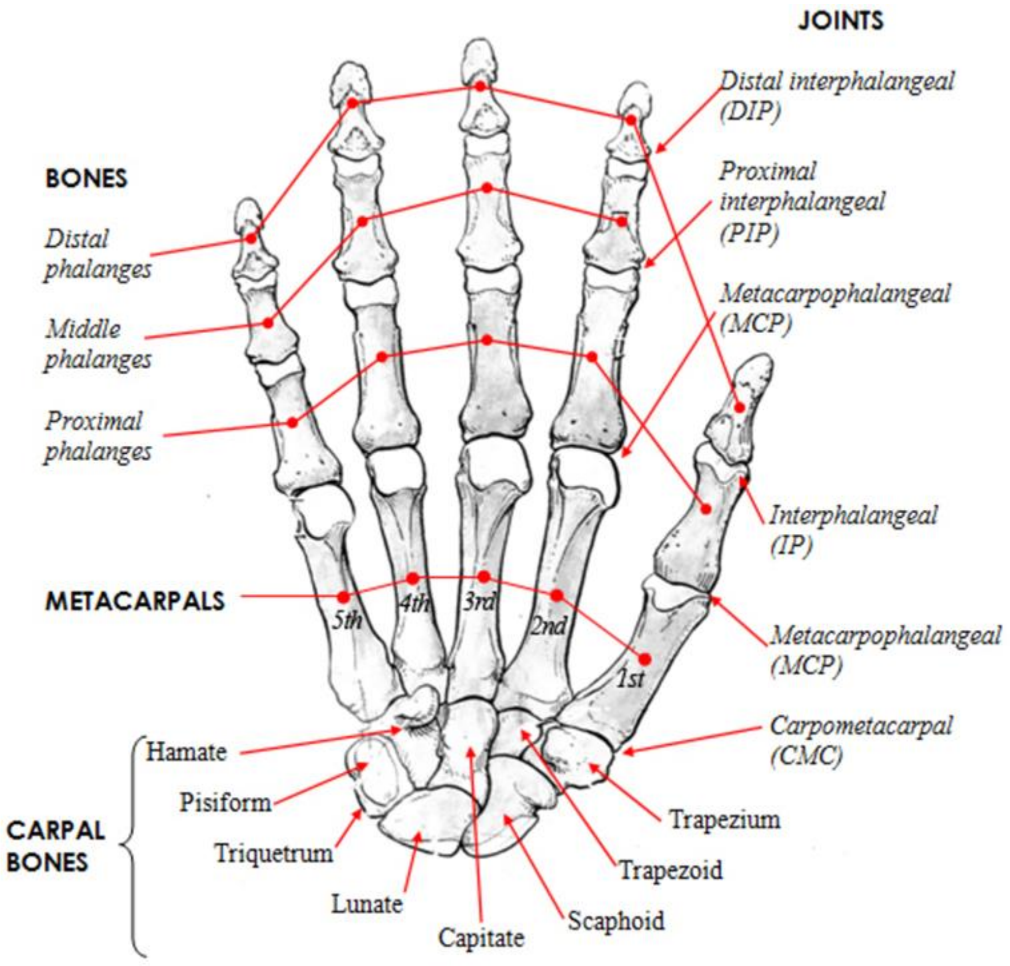


Figure 1: Bones and joints of the hand [11].

2.3 Underactuated Grippers

Underactuated grippers have grasping capabilities ranging from one simple grasp type to multiple complex grasp configurations. One type of basic 3-finger gripper is a gripper with soft monolithic fingers [12]. The fingers are a tendon-driven 3D printed structure with elliptical flexure hinges as joints and the gripper is operated by actuating all the tendons simultaneously [12]. The iRobot-Harvard-Yale (iHY) Hand is considered a moderately dexterous and medium complexity hand able to perform pinch and power grasps as well as in-hand repositioning [13]. While only having 3 fingers, the iHY Hand has 5 actuators to achieve flexion and abduction and results in the ability to execute more complex grasp patterns than most 3-finger grippers [13]. An anthropomorphic robot hand was designed with 4 tendon routing systems and 4 actuators to achieve power grasps and dexterous manipulation, including the complex in-hand manipulation task of equilibrium point manipulation [14].

In WPI's Soft Robotics Lab, Ann Marie Votta et al. [2] designed a 3D printed prosthetic hand with force control. The inclusion of 3D magnetic force sensors in the fingertips expands the grasping capabilities of the prosthetic hand by monitoring the slipping of the object and automating object release [2]. Additionally, the grasp pattern was detected from the electromyographic activity patterns of the user [2]. For a lightweight and inexpensive hand, the prosthetic hand reliably grasped objects from 1 mm to 12 cm and could lift objects greater than its weight [15].

2.4 Tendon Mechanics of Underactuated Fingers

One method of controlling soft grippers is through actuated tendons in the fingers. The number of actuators can be reduced to fewer than the number of tendons through the use of pulleys to create a differential mechanism. A differential system for the tendons allows one actuator to control the grasping force, while the force exerted by each finger on the grasped object is independent of the configuration of the other fingers [16]. Another aspect of tendon-driven fingers is the closing sequence of the phalanges, which is controlled by the placement of the tendon guides relative to the hinges [16]. With flexural joints, the relation of the joint stiffnesses between the joints also contributes to the closing sequence of the finger. To correctly grasp an object, the tendon action should result in the closing sequence of proximal to distal phalanges [16].

The use of multiple tendons per finger can result in more complex grasp configurations and anthropomorphic movements of the fingers. Generally, tendons perform finger flexion within the gripper. However, a second tendon that is offset from the center of the finger and is routed through a moment arm pulley can allow for finger abduction as well [14]. To avoid extra tendons to compensate for finger extension and adduction, spring stiffness in the joints and a torsional spring in the base of the finger allow for the gripper to passively return to its initial open state [14].

2.5 Object Detection

Robotic systems designed to interact with their environment often rely heavily on their vision system. Without this system, the robot is incapable of effectively interacting with its environment [17]. For this project, it is necessary to utilize a vision system that performs object detection. Object detection algorithms have the capacity of localizing multiple objects within a frame [18] as well as identifying them. The purpose of robotic systems is highly varied, but for a gripper intended to grasp objects in its environment, it is necessary to leverage a robot vision system.

2.5.1 Defining a Vision System

To develop an appropriate vision system, one must understand the different terminology used to describe these systems. Computer vision, machine vision, and robot vision are often used interchangeably, but all have specific implications of the intended use. Machine and robot vision can be seen as subcategories of the broad computer vision category. Computer vision takes images and extracts information from them to categorize them. However, machine vision takes this a step forward and uses this same technique for industrial applications. For example, a machine vision system may be used for automatic inspection of parts on a conveyor belt. Robot vision differentiates itself from machine vision in that the information processed by the vision system leads to a physical action carried out by the robot [17]. Per these definitions, this project falls under the scope of utilizing robot vision as the vision system will result in a grasping action by the gripper.

2.5.2 Existing Approaches

2.5.2.1 Augmented Reality (AR) Markers

A prevalent approach to object detection in previous MQP projects has been to use AR markers to mark objects of interest for a robotic hand. Most notably, the 2015 Vision-Based Intelligent Prosthetic Arm project used a palm embedded camera to determine object positions based on recognizing AR markers they attached to different objects [19]. Although relatively straightforward to implement, various teams emphasized that a significant drawback to this system was that all the objects in the environment needed to be tagged for their arm to detect the object [19], [20]. There is a multitude of libraries that may be used to implement these markers such as ARToolkit and ArUco that can also be integrated with ROS (Robot Operating System).

2.5.2.2 Image Processing-Based Methods

OpenCV is one of the most well-known open-source libraries aimed to integrate computer vision into real-time applications [21]. Due to its cross-platform support and documentation, it is frequently used as the foundation for various computer vision tasks. The OpenCV library provides simpler image processing-based object detection methods such as image segmentation and cascade classifiers. However, it also includes the deep neural networks (DNN) module that allows integration with machine and deep learning models that can be used for object detection [22].

2.5.2.3 Machine and Deep Learning

The object detection method used is highly dependent on a use case. If a machine or deep learning model is used, the user has the option to construct and train a model from scratch or utilize transfer learning on a pre-trained model [18]. However, constructing a model from scratch is mostly recommended for users with experience in creating convolutional neural networks (CNN) as it entails choosing the exact weights and convolutional layers that will yield optimal results. In the case of conventional machine learning methods, the features to be detected must also be identified by the developer as they cannot be automatically selected like in deep learning networks. Common machine learning object detection algorithms include HOG feature extraction, the Viola-Jones algorithm, and cascade classifiers [18], [23].

As mentioned previously, deep learning object detection methods leverage CNNs to detect and learn object features from the input data. This requires a large amount of raw labeled data so that the network can be trained to achieve the expected degree of accuracy. As a result, these models are only recommended if the user has access to a powerful GPU that can efficiently train the model. However, the majority of the impact felt by these large training sets is mostly seen when creating a model from the ground up. Often transfer learning will instead be used to take a pre-trained model, provide it a new set of labeled data, train it and then use it as expected. This approach takes advantage of the careful training done by deep learning specialists on a common dataset. The accuracy and efficiency of these pre-trained models also depend on the base network architecture and object detection frameworks they were built upon. For example, common base architectures include VGG, ResNet, as well as MobileNet [24]. Common frameworks and methods include Faster R-CNN, YOLO, and SSDs [25]. Pre-trained models will often select a base network then remove the last few layers that will classify an image and replace those layers with a specific object detection framework [18].

2.5.2.4 Point Clouds

Point clouds are useful to leverage to perform object detection. Point clouds are data structures used to represent a collection of multi-dimensional points and are commonly used to represent three-dimensional data [26]. Acquired from sensors like stereo cameras, they represent the X, Y, and Z geometric coordinates of the underlying object. The retrieved point cloud could be used to determine the object's depth or distance from the camera while the filtered data could help classify the object [27]. Point clouds provide highly useful information that can be used to accurately identify the location of an object and enable a robot to move appropriately to its location. A previous MQP project utilized point clouds to classify objects within three categories: cylinders, spheres, and cubes and then used the camera input to signal to the prosthetic arm when to begin grasping an object [27]. The Point Cloud Library (PCL) is an open-source project for 2D/3D image and point cloud processing that can interface with OpenCV [26]. This library contains algorithms capable of various tasks such as filtering outliers from noisy data, stitching 3D point clouds together, and segmenting images. By leveraging this library it is possible to filter out the noise from a point cloud, and then use this information to more accurately classify the filtered object.

2.6 Impedance Control

2.6.1 Difficulties of Controlling Robots

Robotic manipulators can be classified into two categories, rigid robots and compliant robots. Stiff robots typically have very precise position control and can accurately determine their end effector position using sensor feedback [28]. However, these robots have trouble interacting with their environment. They tend to struggle to safely interact with objects and humans, especially in dynamic environments despite their joints having high levels of precision [28]. Stiff robots lack the ability to slow down when approaching an object, which helps to avoid high-speed collisions that may damage both the robot and surrounding objects or humans. Additionally, stiff robot systems are more likely to experience oscillations when they come into contact with objects [28]. This instability further deteriorates the potential for force control.

On the other hand, there are compliant robotic manipulators that are significantly better at managing their interactions with their environment [28]. However, the downfall of compliant systems due to their underactuated nature is the reduction in precise control [28]. In fact, soft robots are even more sensitive to external forces in the environment than rigid robots since they are inherently compliant. Since there is not as much control over the robot's trajectories, manipulation is significantly more difficult. Finding the balance between precise position control and robust grasp control is critical to the design approach.

2.6.2 Impedance Control in Soft Robotics

A robot's movements can be divided into three phases [29]. In the first phase, the robotic manipulator moves in free space clear of objects and potential obstacles. The second phase is initiated when the manipulator comes close to an object. This is known as the transition phase, rightfully named as it transitions from free motion to constrained motion. The third and last phase is the constrained motion phase where the robot is fully constrained and in contact with an object.

These different movements can be determined by control algorithms. Model-based control systems are typically more accurate since the nonlinearities in the dynamic model are canceled out by the feedback in the system [29]. However, as a result of its better performance, the model-based control algorithms have larger computing loads.

Impedance control, which is a type of indirect force control algorithm, has been used to guarantee safe and compliant interactions between a robot and its environment. With impedance control, the interaction force is regulated so that the dynamic relationship between the interaction force and its resulting motions are controlled [29]. With this controller, the robot would be required to be tuned to the specific task the robot would be carrying out as well as its dynamic properties, namely, mass, inertia, and damping. Impedance control is modeled as follows:

$$M_d(\ddot{X} - \ddot{X}_d) + B_d(\dot{X} - \dot{X}_d) + K_d(X - X_d) = F_e$$

Variables with the subscript "*d*" represented the desired values. The variable *X* is the actual position vector of the robot's end effector while *M*, *B*, *D*, and *X* represent inertia, damping, stiffness, and position respectively. The result of the operations is the interaction force, F_e. The interaction force is defined as the force between the robot end effector and the interacted object and can be monitored with a force sensor [29].

2.6.3 Impedance Control Design

Designing the controller by selecting the parameters to monitor the impedance of the robot is called impedance shaping [29]. This is achieved by using a robot's dynamic model and calculated based on its differential kinematic relationship between the joint-space velocity and the Cartesian-space velocity. Based on the implementation of the impedance control, it may be a motion control-based approach, which is often used in industrial robots. With a motion control-based approach, the impedance equation would incorporate a new position variable that would represent the reference trajectory [29]. This type of impedance control may also be referred to as admittance control [29].

2.7 Force Sensing

Force sensors can be incorporated into a gripper to enable intelligent grasping with force control [2]. To do so, the sensors utilize the information on shear and normal forces that is captured. The normal force would help to detect stable grasps, while the readings from the shear force would help identify when an object is lifted or slipping. This type of data would be crucial to understanding the stability of a grasp. For instance, if there is an increase in shear force and the gripper has not been moving, then most likely the item that it is holding is about to slip and the gripper may want to readjust its grasp.

3. Project Strategy

3.1 Project Goal

The FROG MQP is designed to explore and research the capabilities of a soft robotic gripper while utilizing the advantages of origami folds. With origami, soft materials can be manipulated to create a flexible yet durable structure based on the bending and torsional stiffness properties of the origami pattern. This project analyzes the benefits of having certain structural components of the gripper made with origami folded soft materials as well as investigates means to control the movement and strength of the robotic gripper. Being a soft robot, the system requires force sensors that help determine and track the gripper movement. In conjunction, this MQP examines the potential of utilizing computer vision to augment the gripper's capabilities through object recognition. Therefore, the goal of this project is to create a flexible robotic origami gripper that can recognize and grasp objects.

3.2 Task Specifications

The scope of this project is to develop a flexible robotic gripper that utilizes origami designs to stably grasp various objects detected by computer vision. By the conclusion of the MQP, the team plans to achieve the following features:

1. The total cost of the project should not exceed $600 (MQP student budget).
2. The maximum weight of the gripper including its onboarded motors must not exceed 333 grams (weight of Ann Marie's hand).
3. The payload weight should be a minimum of 50 grams (the weight of a small ball).
4. The gripper must be able to successfully grasp an object up to an 8cm diameter/width (or have a crevice or protrusion for the gripper to hold it).
5. In a closed gripper position, the gripper should be confined to a 25 cm cube.
6. The gripper will contain origami modules.
7. The gripper will contain a single modular palm structure with 2 sets of interchangeable origami fingers.
8. There must be at least 2 different origami finger designs.
9. The gripper will have at least 3 fingers (maximize grasp potential).
10. There should be at least one force sensor per gripper finger.
11. The gripper in coordination with the Jaco arm must reach a final grasp position within 30 seconds.
12. The gripper must close around the object within 1 second.
13. The gripper must be able to grasp objects that do not require a change in wrist orientation.
14. The gripper should have a maximum of 1 motor per finger.
15. The gripper should have a maximum of 4 motors per gripper.
16. Motors should fit within the palm of the gripper.
17. The gripper must not shake to the point of gripping failure.
18. The system must be able to respond to input from the vision system.
19. An object must be identified by the vision system within less than 120 seconds.

20. The vision system must be able to detect the shape of an object to indicate the appropriate grasp for the gripper.
21. The vision system must be able to accurately recognize an object within the Jaco arm's task space that is fully present in the camera’s field of view (i.e not expected to recognize occluded objects).
22. The object will be recognized using partial object shape data (i.e it will identify a soda can as a cylinder).
23. The vision system will determine the location of the object to a degree of accuracy where it will be able to successfully grasp the object.
24. The camera size should not exceed 90 mm x 25 mm x 25 mm.
25. The vision system and the gripper will use up to 2 total processing units.
26. System architecture should be designed in a modular fashion.
27. The vision system should utilize an RGB-D camera to capture depth information.
28. Wires should have enough slack so that they do not unplug from ports.
29. The gripper design should ensure the mechanical and electrical components are easily accessible if maintenance is required (should not require taking apart the entire module if one component is in disrepair).
30. The gripper should not endanger any humans.

3.3 Approach and Timeline

3.3.1 Gripper Design Process

In the first quarter, the project began with paper origami prototyping to explore different variations of gripper fingers. This included testing models of fingers that were either a continuous origami pattern or an origami structure with discrete folds that define joints. This rapid prototyping was done with paper to allow the team to understand the advantages and disadvantages of origami patterns and structure. Additionally, this allowed the team to physically test the structures of the gripper fingers before committing to a particular design. By the second quarter, the project team chose two finger designs and began developing a complete set of fingers for the gripper. Iterations of the two origami finger designs were modified throughout the second and third quarters to improve the grasping ability of the fingers. One finger design required extra support through the addition of endoskeleton pieces to reinforce the origami structure, which was implemented in the third quarter.

In addition, the team created a gripper palm design. A factor that contributed to this design included the grasp patterns that the final gripper needed to accomplish. An initial palm design was developed in the second quarter and was redesigned at the beginning of the fourth quarter to adjust the angle of the fingers to perform the desired grasps as well as to include an attachment to the Jaco arm. Additionally, connector pieces to attach the origami fingers to the gripper palm were created at the beginning of the fourth quarter. Finally, during the fourth quarter, fingertips were designed to attach to the origami fingers as well as secure the magnetic force sensor.

3.3.2 Gripper Controller Development Process

The control portion of the project began in the first quarter by designing a circuit that was capable of controlling multiple motors. These motors would be small DC motors that were small enough to fit within the palm of the gripper yet strong enough to compress the Yoshimura origami modules. The motors in the gripper were programmed to receive feedback from its motor encoders to determine its desired position. Using breadboards to control four separate motors proved to be quite space consuming. Therefore, throughout the second quarter, the team experimented with designing a printed circuit board (PCB) to minimize the footprint and reduce the total number of electrical wires. Concurrently, the team also developed a program that began to coordinate the fingers of the gripper to form four different grasping configurations.

In the third quarter, the team worked together to develop a dynamic model of the triangular beam finger by determining the inertial, damping, and spring parameters. This dynamic model was then used as the foundation for the controller. In the fourth quarter, this controller was then implemented on the robotics gripper as well as used to develop a simulation. The control system was applied to all fingers of the gripper and resulted in coordinated finger control. By working in parallel with the computer vision system, the gripper was able to perform smart grasp patterns based upon the specific object identified.

3.3.3 Vision System Development Process

In the first quarter of this project, the team researched and tested various open-source object recognition models. This included deep learning models as well as popular image segmentation techniques. Through this research, the team understood the current limitations in existing computer vision systems for similar projects. The second quarter continued to explore image segmentation techniques that utilized point cloud data gathered using an RGB-D camera. Segmenting the object of interest from a captured scene subsequently led to this object being matched to templates corresponding to the grasp of interest. This research heavily leveraged the use of the Point Cloud Library (PCL). In the final two quarters, research stepped away from utilizing template alignment to match an object to a corresponding template. This was because of the limited reliability of template matching caused by segmentation limitations discussed in Section 5.1.4. The vision system switched to using PCA to match the appropriate template to an object. The vision system was then integrated with the existing gripper.

3.4 Project Timeline

With this project elapsing over an entire academic year, four quarters, the team has denoted below a number of major milestones completed throughout the year.

Table 1: MQP Milestones

Milestones:	***Deadline:***
Final Proposal	Week of Oct. 12th
Final Testing of 2D Image Segmentation	Week of Oct. 12th
Finalized Gripper Design	Week of Dec. 7th
Finalized Finger Design	Week of Dec. 7th
Finalized 3D Image Segmentation	Week of Dec. 7th
Completion of Grasp Patterns	Week of Feb. 22nd
Implementation of Object Recognition with Camera Unit	Week of Mar. 15th
Implementation of Controller	Week of Mar. 15th
Final Integration of Vision System with Gripper	Week of Apr. 5th
Final Presentation	Apr. 30th
Final Paper Completed	May 6th

4. Design

The design of the gripper and its overarching system are documented in this section. Each subsection summarizes the iterations of work required to reach the final system design.

4.1 Gripper Structure

4.1.1 Palm

Task specification #7 of this project emphasized designing a gripper structure with interchangeable gripper fingers. A modular structure such as this supported a beneficial prototyping environment for the gripper's fingers. Because of the ease of interchanging fingers, the gripper was capable of supporting a variety of finger combinations. For instance, the "thumb" finger could be a triangular beam finger while the other three fingers could be Yoshimura fingers. An additional benefit to this modular design was that it provides ease of access to different components of the gripper for any maintenance that may be required.

The hand structure is designed to be capable of incorporating up to four origami fingers. With four fingers, the gripper is capable of executing three different grasp patterns: pinch, tripod, and power. Human hands are capable of performing the common grasps shown in Figure 2. In the first iteration of the design, the fingers were positioned as depicted in Figure 3. The grasp configurations are listed in Table 2.

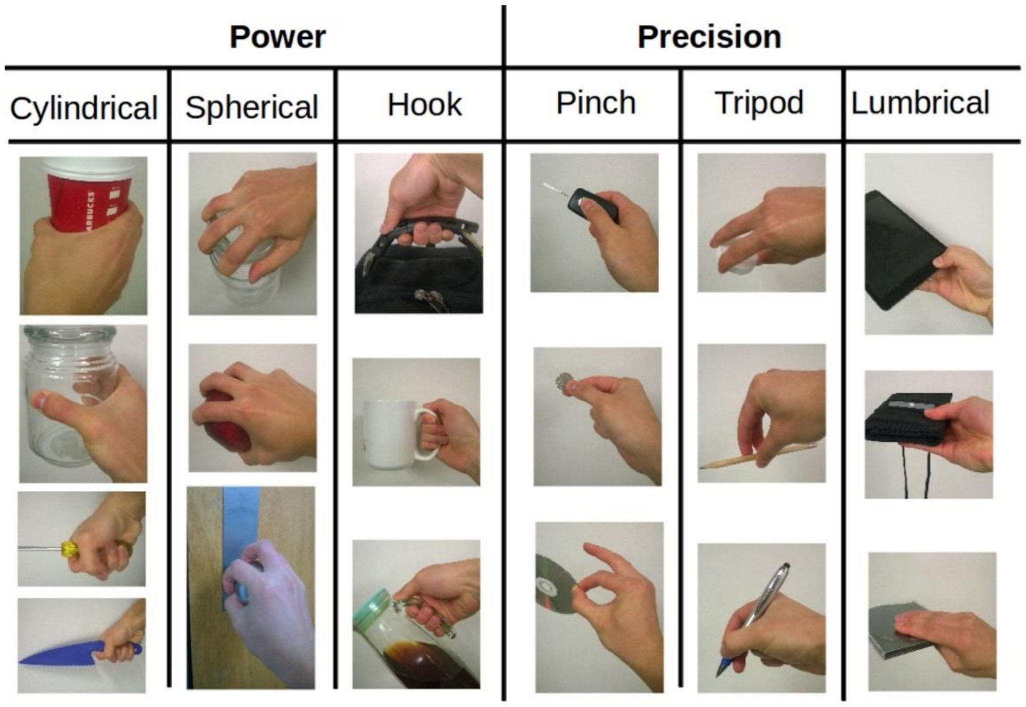


Figure 2: Power and precision grasp types [30].

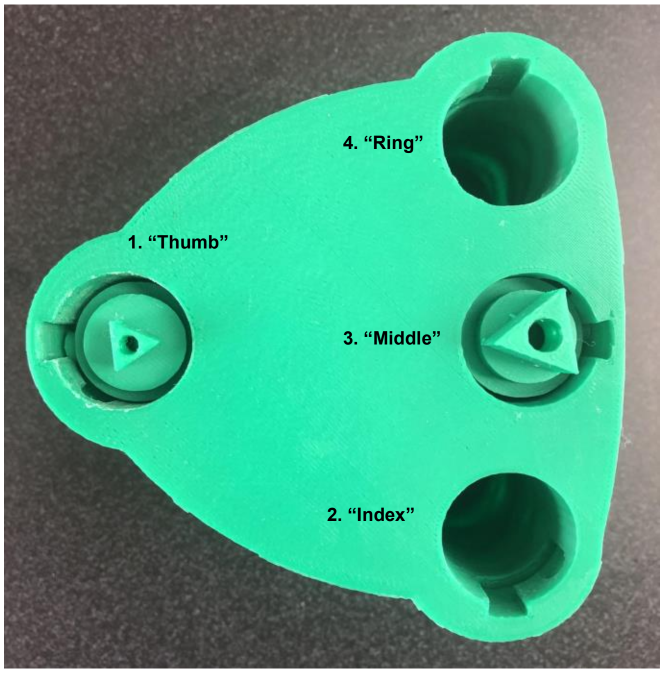


Figure 3: Bottom view of triangular palm structure

Table 2: Grasp Configurations.

		Finger			
		1 - "Thumb"	*2 - "Index"*	*3 - "Middle"*	*4 - "Ring"*
Grasp Types	*Pinch*	✓	☐	✓	☐
	Tripod	✓	✓	☐	✓
	Power	✓	✓	✓	✓

One of the primary issues of this first design is that in the open configuration, the fingers of the gripper do not open as much as desired. Additionally, the design does not include a way of attaching the robotic origami gripper to the Jaco arm, which is the robot we are using to maneuver the gripper in the task space. To address these problems, the team designed a new hexagonal palm that deviates from the initial triangular design. To allow the fingers to open to an optimal width, separate angled faces were created so each finger is slanted outwards.

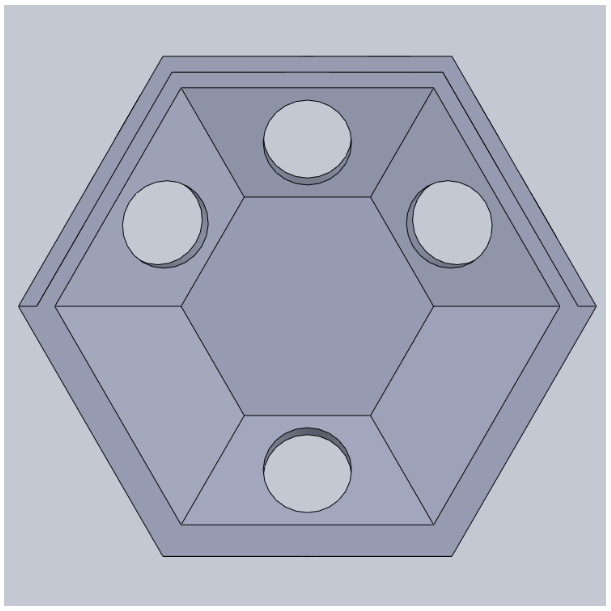


Figure 4: Bottom view of hexagonal palm.

This new design also allows for easier maintenance of the gripper as it incorporates a multilayer design. This made it possible to remove different levels to fix components in specific layers. The base of the palm, seen in Figure 4, is where the four DC motors were kept and the four fingers were attached to the palm. As seen in Figure 5, the midsection of the palm holds the custom PCB, discussed further in Section 4.3.2, with the lip that was created along the bottom edge to ensure the PCB lays flat without falling through. The wires were located between the base and midsection of the palm and connected the motors to the PCB to provide signal and power. The side of the midsection has a slot that allows wires from the gripper to connect to power and leads. Lastly, the lid of the gripper connects the gripper to the Jaco arm, with a total of six holes across the top edge of the lid to ensure a secure connection.

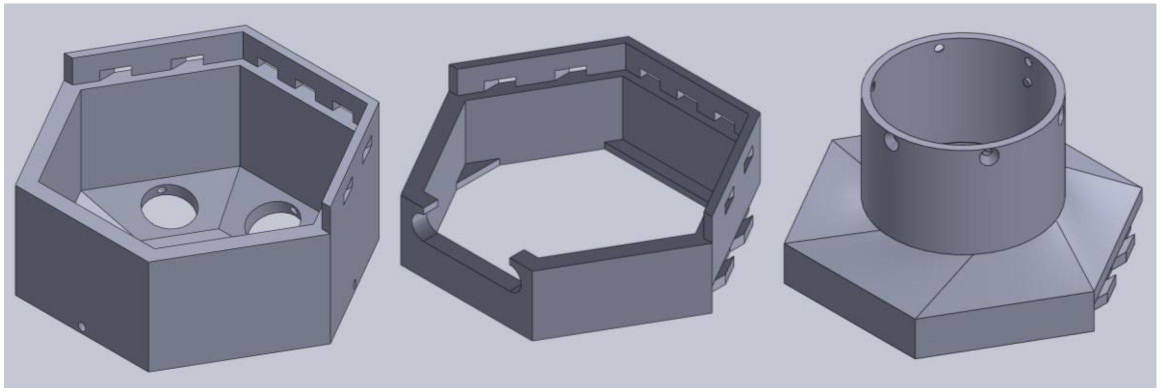


Figure 5: Multilayer design of gripper.

4.1.2 Connectors

A twist and lock mechanism connects the fingers in the initial palm structure. A custom connector was created to attach each finger to the base of the palm. The base of the finger connects to the triangular portion of the connector. At the cylindrical portion of the connector, there is a key that coincides with a cutout created within the finger holes of the palm structure, as seen in Figure 4. This twist and lock mechanism is illustrated in Figure 6, where the connector is inserted upwards into the finger hole and turned approximately 60 degrees clockwise before being pulled back downwards. With this design, the motors can be left within the palm and only the fingers need to be changed to switch between the triangular beam and Yoshimura fingers.

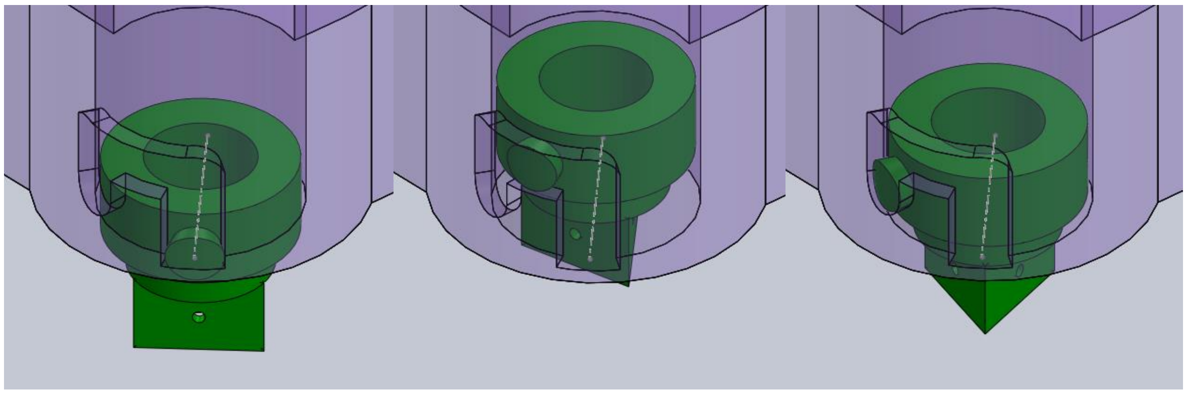


Figure 6: Twist and lock mechanism

The new palm design utilizes a new connector that pivoted away from the twist and lock mechanism. The intention of the first iteration design was to ensure all the fingers were in the correct orientation. However, the team found that when the fingers were in the closed position, the fingertips were offset from the center of the palm. In addition, the connectors popped out of their sockets when the torque provided by the motor was larger than the force applied to the finger.

To mitigate these issues, the new connector is secured to the palm with a 16 mm M3 screw. The connector has a screw hole that is tangent to the outer edge of the hexagonal palm. With this new design, the connector is oriented in the correct direction for all the gripper's fingers. Two versions of the design exist to account for the two finger designs. The triangular prism, seen on the left side of Figure 7, allows the Yoshimura finger to connect to the palm. The longer prism with slanted base acts as the endoskeleton (see Section 4.2.2) for the triangular beam fingers, as shown on the right side of Figure 7.

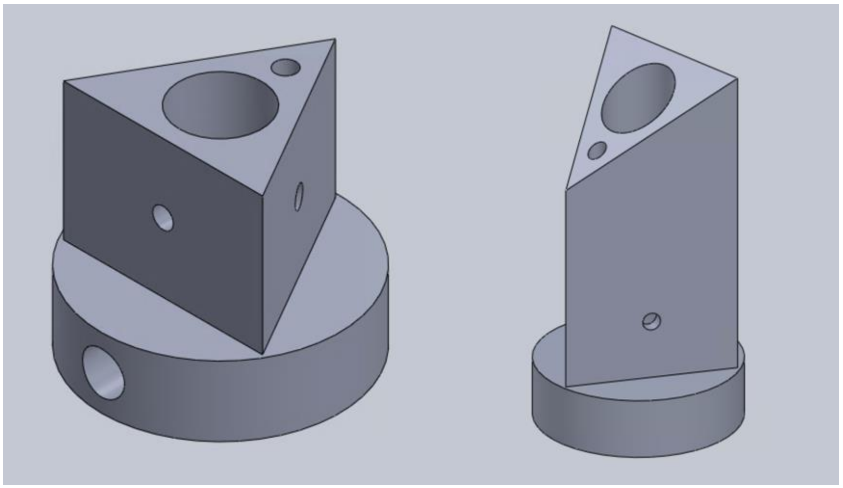


Figure 7: New connector for Yoshimura fingers (left) and triangular beam fingers (right).

4.1.3 Spool

Spools are pieces that attach to the Adafruit N20 DC motor shafts and help contract and expand fingers by either winding or unwinding the fishing line that act as the tendon of the finger. The spools were originally designed with a 1.3 mm lip and 5 mm wide. Each finger's fishing line is attached to a spool by threading through the hole on the spool's lateral area and making a knot at the spool's base. Since the spool was designed to fit the Adafruit N20 DC motor shafts, there is a 2.3 mm flat surface to align the D-shaft.

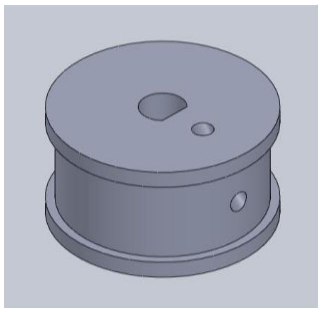


Figure 8: Spool for finger tendons.

Because the fishing line tended to fall off the spool when winding, the width of the spool was doubled, shown in Figure 8. An additional change to the spools connects the two cable holes on the spool via a direct path. Originally, the holes were connected at a right angle, thus making it exceptionally difficult to thread the fishing line through the spool. The direct hole provides unobstructed access for the fishing line to be threaded.

4.1.4 Motor Holders

Motor holders were developed to ensure the motors remain in place while the fingers move and large torques are applied. The flat surface on the bottom of the motor holders complements the shape of the motor and allows the motor to maintain a specific orientation. In total, four motor holders are secured to the bottom of the palm, as shown in Figure 9.

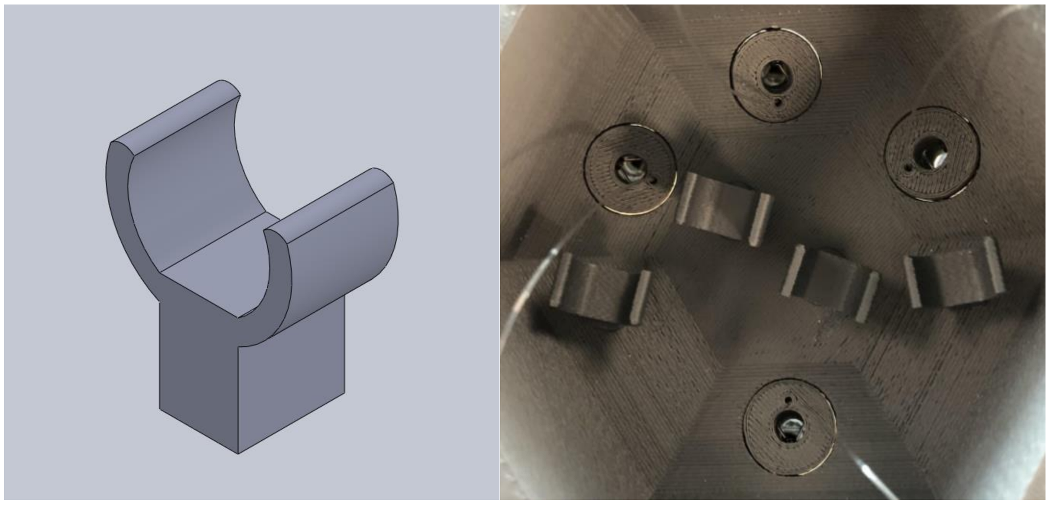


Figure 9: Motor holders.

4.2 Finger Design

4.2.1 Origami

This project explores two types of origami fingers to compare their properties and performance in a gripper. One type of finger is designed based on a simple triangular beam, while the other is based on a traditional Yoshimura pattern.

4.2.1.1 Paper Origami Design Iterations

Upon experimenting with folding paper Yoshimura patterns and triangular linkages that presented varying degrees of structural integrity and bending configurations, it was noted that the Yoshimura pattern with 45º angle diagonals forms rigid structures and the diameter can be changed by the length of paper, as demonstrated in Figure 10. Prototyped Yoshimura patterns with different diagonal angles and radial frequencies are shown in Figure 11. Additionally, a smaller scale version of the folded Yoshimura pattern, seen in Figure 12, establishes the efficacy of using this pattern in an underactuated finger. Exhibited in Figure 13, is a simpler design made from triangular beams and connected folds to create joints. Both Figure 13 and Figure 14 show the paper origami prototypes connected to a motor to demonstrate the bending configurations that occur when a tendon is attached. The paper prototypes helped the team learn folding techniques that were later applied to the origami design printed on PET.

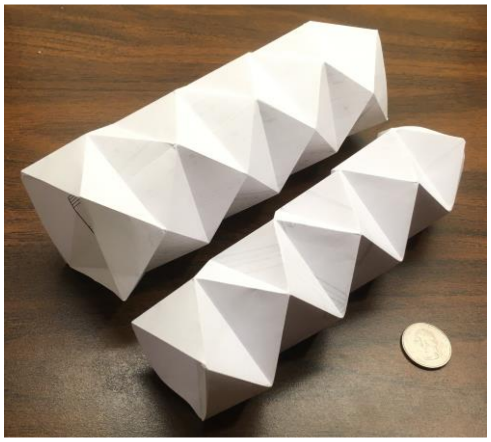


Figure 10: Yoshimura pattern with 45º angle diagonals.

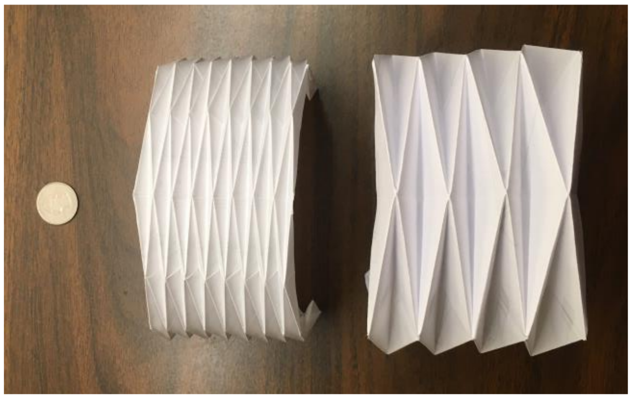


Figure 11: Yoshimura pattern with different radial frequencies and axial lengths.

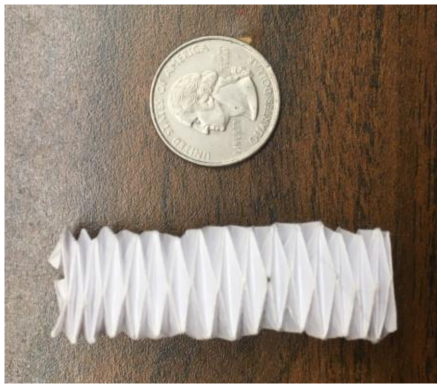


Figure 12: Small scale Yoshimura pattern.

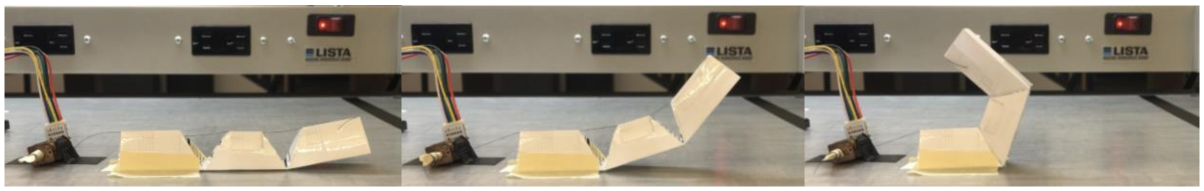


Figure 13: Progression of triangular linkage finger curling.

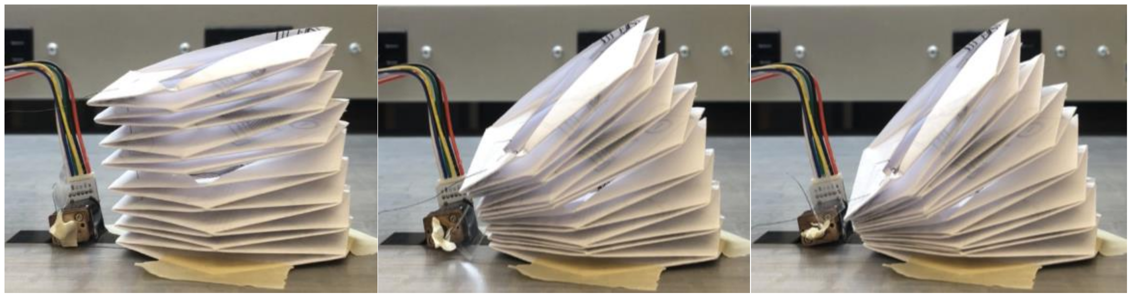


Figure 14: Progression of Yoshimura collapsing.

4.2.1.2 Triangular Beam Finger

The triangular beam finger has multiple triangular prism links attached by one continuous strip of plastic that provides a small amount of spring back. The first triangular beam laser cutting design is shown in Figure 15.

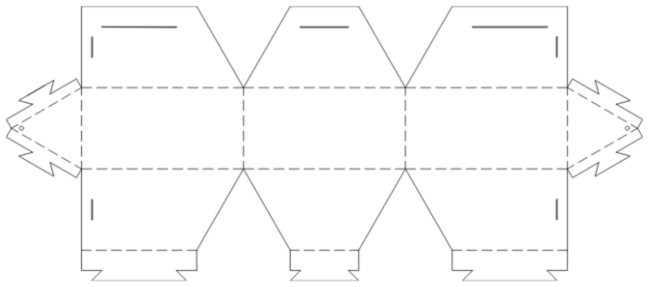


Figure 15: First triangular beam finger design with solid lines to be cut and dashed lines to be perforated by the laser cutter.

`One iteration is designed with the link lengths based on the optimizations done for the 3D printed prosthetic hand by Votta et al. [12]. As shown in Figure 16, the triangular beam with the optimization values resulted in a very slender shape as the origami design requires tabs to fold it together. Additionally, this design did not integrate well with the palm as the palm design by the team has significantly different properties than the 3D printed prosthetic hand. The design also includes flaps at the end for a connector to attach the finger to the palm as well as an internal triangular prism that allows for cable routing, displayed in Figure 17. However, the internal triangular prism did not stay in place with folding, tabs, or glue, so cable routing was implemented through a 3D printed endoskeleton, which is described in Section 4.2.2.

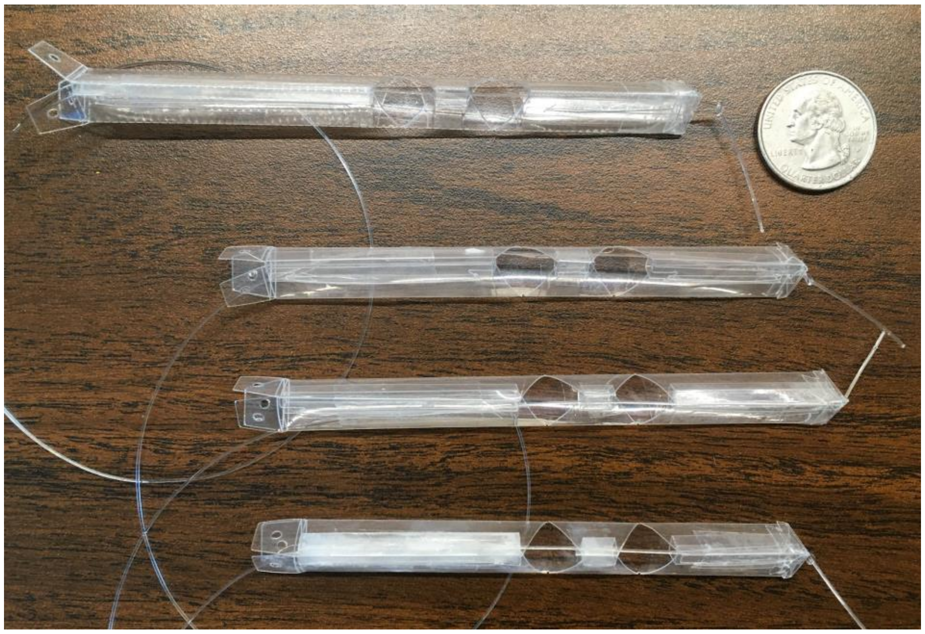


Figure 16: Triangular beam design with optimized link length from Votta et al.

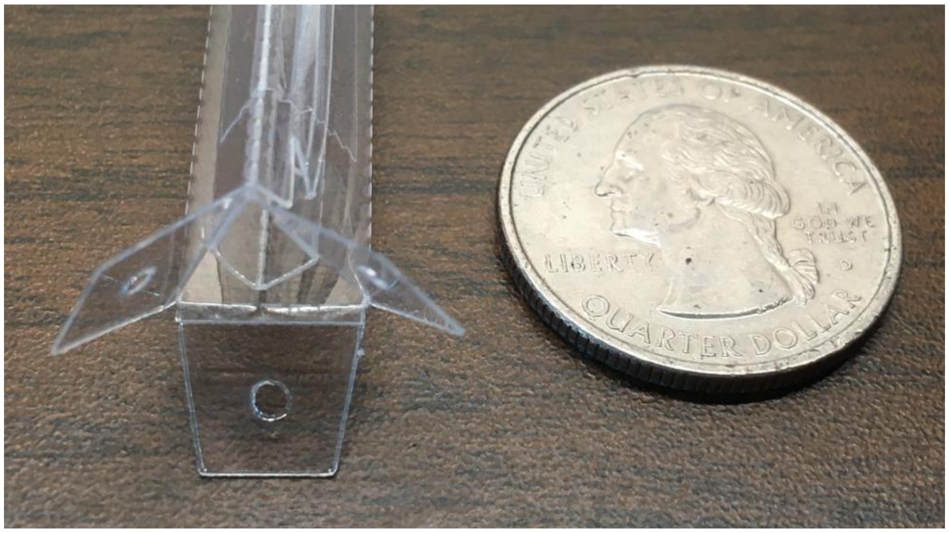


Figure 17: Flaps to attach finger to palm connector and internal triangular prism for cable routing.

To achieve extension in the triangular beam design, a double triangular beam finger was designed to allow a second cable to run on the back of the finger, shown in Figure 18. A fourth link is included in the finger because the first link is rigidly attached to the palm. This allowed the three most distal links to create a more adaptive grasp than only two rotating links.

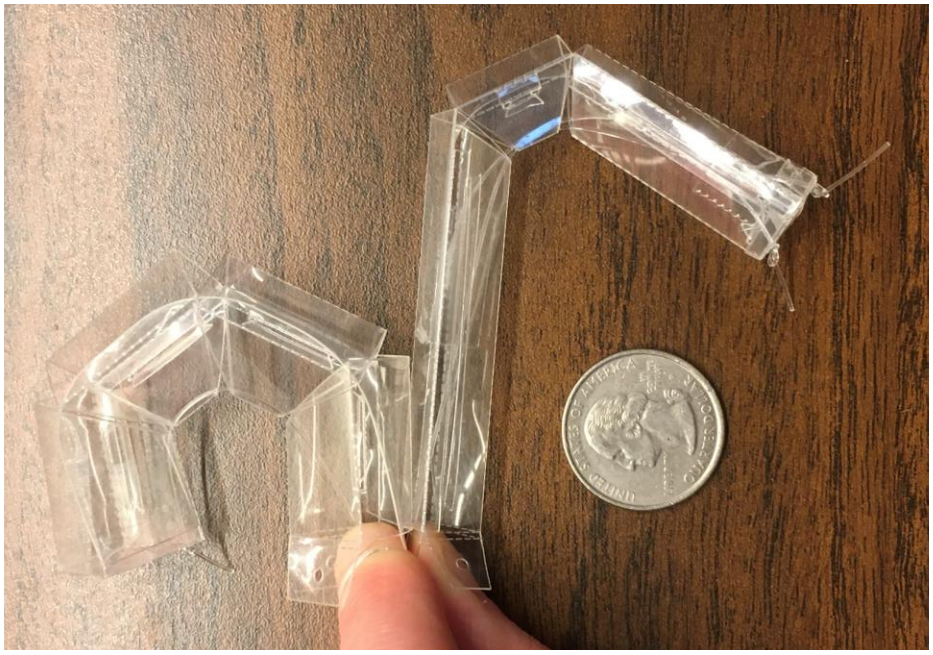


Figure 18: Double triangular beam design with 4 equal length links (left) and optimized lengths from Ann Marie (right).

The final triangular beam design shown in Figure 19 varies slightly from the initial double triangular beam design. The link lengths were adjusted so the proximal link is as short as possible to allow for tabs since it is not a rotating link. The distal link is shorter than the intermediate links to achieve a better closing configuration. This design was created with the intention that the fishing line is secured to the fingertip (discussed in Section 4.2.3), so the end piece to tie off the fishing line was removed from this design.

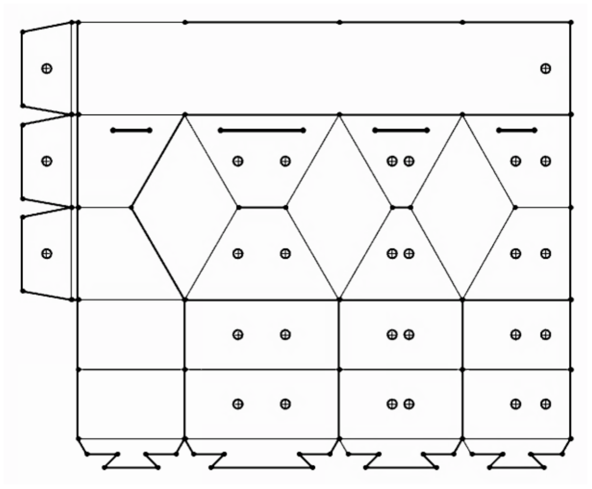


Figure 19: Final design for triangular beam finger with the proximal links (left) and distal links (right).

4.2.1.3 Yoshimura Finger

The other type of finger design is inspired by the Yoshimura pattern and includes tabs and slots to keep the ends together. The design in Figure 20 is adapted from a Yoshimura module in the WPI Soft Robotics Lab.

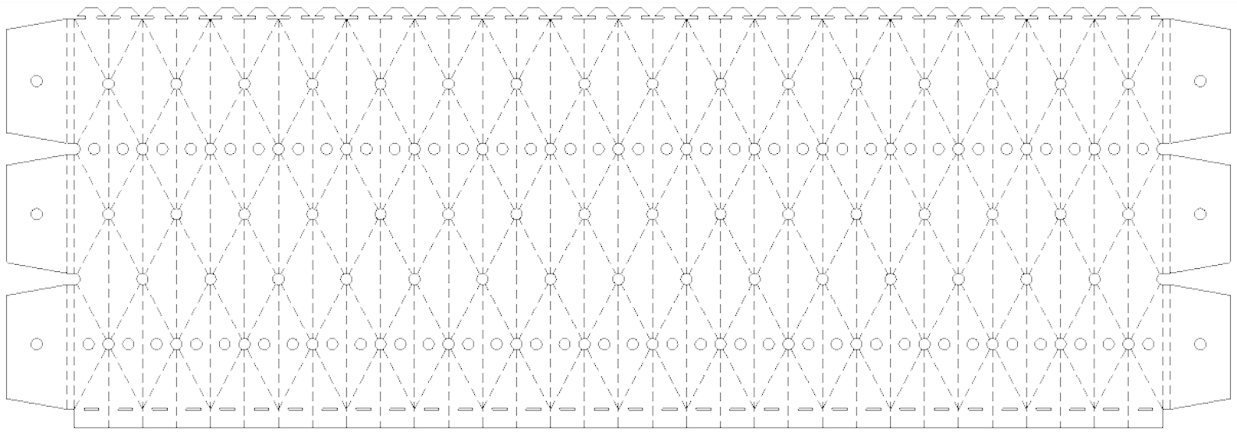


Figure 20: Yoshimura pattern for laser cutting with tabs (top) and slots (bottom) and attachment flaps (left and right side).

The power and focus of the laser had to be adjusted for the plastic to be fully cut but not melted. This was important due to the difficulty associated with inserting tabs of such small size and having the tabs not pop back out. The Yoshimura finger before and after tabs are put together is shown in Figure 21. Due to complications with the laser cutter, a new workflow for designing and printing origami patterns was developed and is found in Appendix A.

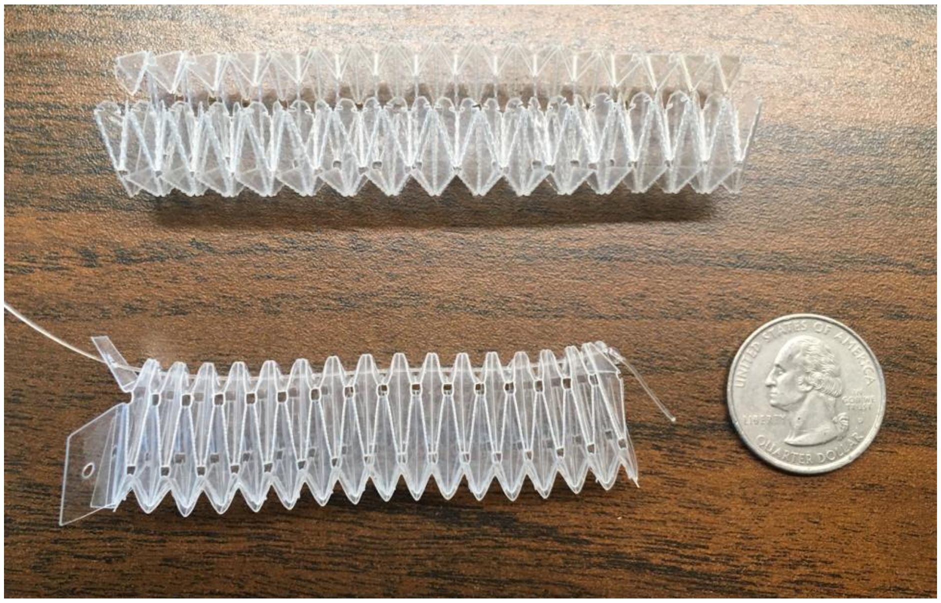


Figure 21: Yoshimura module from the lab with tabs open (top), recreated piece with tabs together and fishing line (bottom).

By repeating the Yoshimura pattern more times, different length fingers were created. After testing the two fingers shown in Figure 22 on the gripper, the shorter finger was chosen because the proportions allowed the fingers to minimally overlap during the grasp.

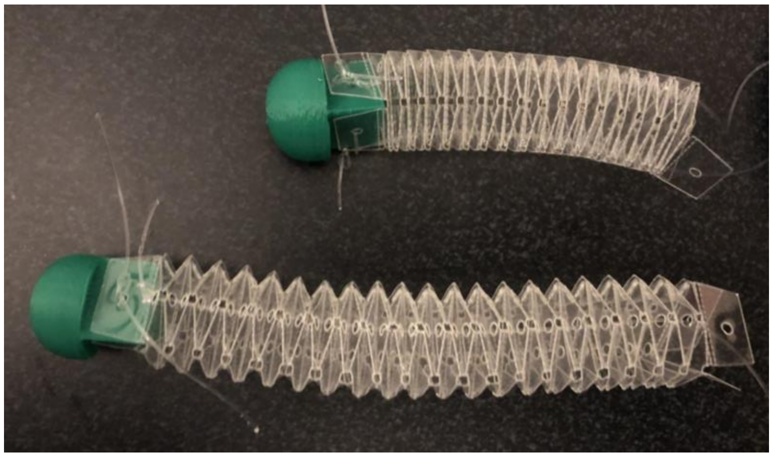


Figure 22: Two lengths of fingers using the Yoshimura pattern.

4.2.2 Endoskeleton

Since cable routing could not be implemented through the folding pattern of the triangular beam design, 3D printed endoskeleton pieces were created to align the tendons of the triangular beam fingers. The endoskeleton is based on a triangular prism with diagonal bases. The cable routing hole is closer to one point of the triangle to provide greater torque to rotate the links. The holes on the lateral faces of the triangular prism are used to secure the endoskeleton to the triangular beam finger with fishing line. The length of each endoskeleton piece corresponds to its triangular beam link length, pictured below in Figure 23.

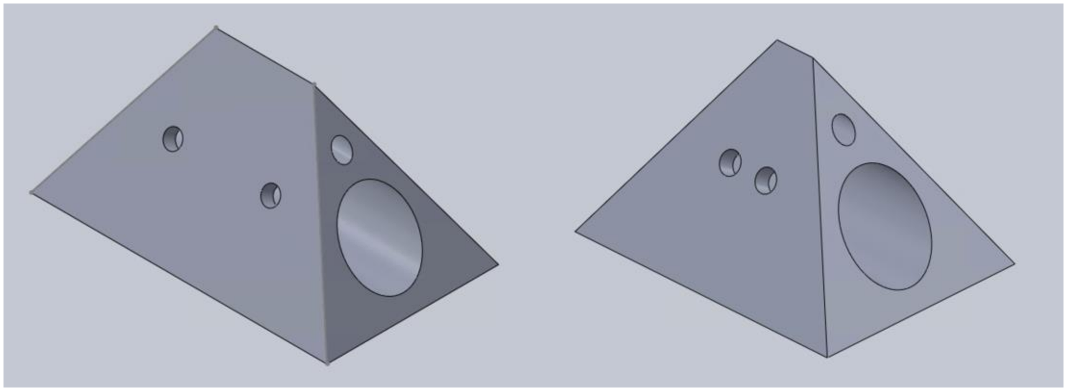


Figure 23: Endoskeleton for link 3 (left) and link 2 (right) of the triangular beam finger.

4.2.3 Fingertip

The fingertip is inspired by the fingertips and force sensors of the 3D printed prosthetic hand designed by Ann Marie Votta et al. [2]. The 3D magnetic force sensor circuits provide information at the tip of the origami fingers regarding the normal force applied as well as any shear forces. A PCB is mounted into the fingertip and a small magnet is inserted into the hole to create the magnetic force sensor, shown in Figure 24. Initially, the team planned to incorporate the force sensors into each of the gripper’s fingers. To achieve this, the triangular prism portion of the fingertip is similar to the connectors discussed in Section 4.1.2 and is used to attach the fingertip to the origami finger. Additionally, the fishing line which acts as the tendons of the finger is run through the triangular prism and affixed to the fingertip with a knot. The Yoshimura finger has a similar fingertip, with the only difference being that the triangular prism section was modified to attach to the Yoshimura origami pattern, displayed in Figure 25.

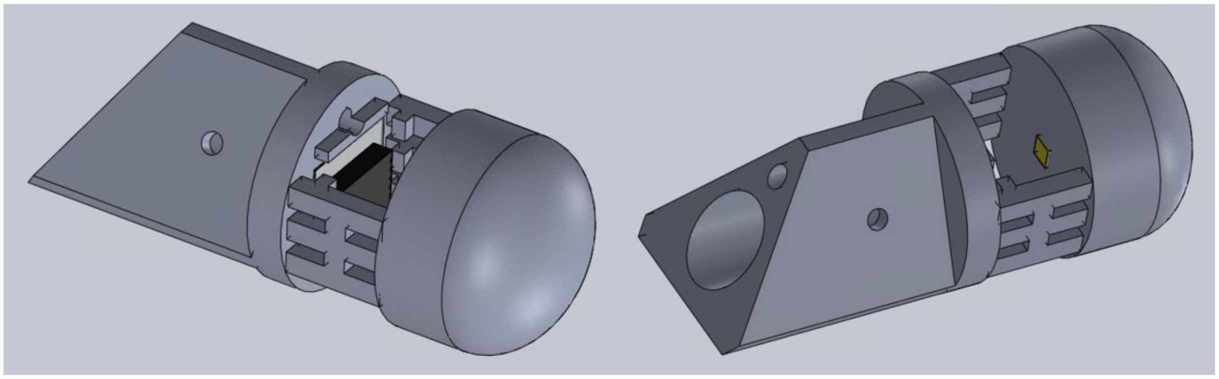


Figure 24: Fingertip for triangular beam fingers, showing the black PCB (left) and the beige cube magnet (right).

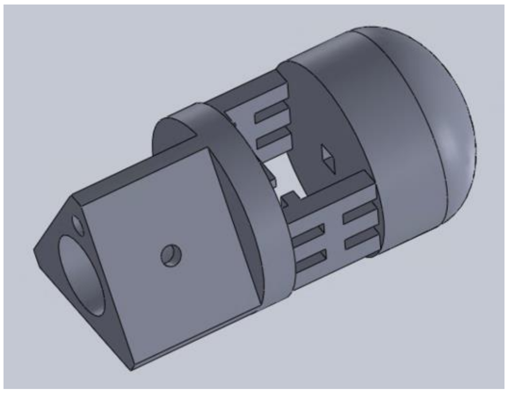


Figure 25: Fingertip for Yoshimura fingers.

4.3 Electronics

4.3.1 Hardware

To determine the optimal motor to drive the tendons of the gripper, the team experimented with several small motors. These tests involved a 3V DC motor as well as an N20 6V DC motor with a pre-attached magnetic encoder. The pre-attached encoders are a beneficial feature as they ensure the motor is correctly reaching the desired speed and position. To allow the motor to rotate both clockwise and counterclockwise, the circuit incorporates a TB6612 DC motor driver. With an H-bridge, it is possible to control the motor direction with two push buttons. When neither button is pressed, the motor is idle. However, whenever push button 1 is depressed, the motor turns counterclockwise and if push button 2 is pressed, the motor rotates clockwise. Appendix B contains the schematic that illustrates the initial motor circuit.

This test circuit examined the capabilities of the Adafruit N20 DC motor. The Adafruit N20 DC motors were chosen for this application due to the motor specifications. These motors are geared at a 1:50 ratio which is responsible for a no-load speed of approximately 200 RPM and provides a torque of about 200 kilogram-centimeters [30]. The second iteration of the circuit design utilizes an Espressif ESP32 DevKit microcontroller to control four total motors in conjunction with an Adafruit FeatherWing motor driver. The Espressif ESP32 Dev Kit is an optimal microcontroller to manage and control the motors because the board has 39 GPIO pins that provides ports for all the external encoders and force sensors that are beneficial to monitor the system [31]. Additionally, this microcontroller is capable of communicating with other devices through SPI, I^{2}C, and transmitting information through WiFi or Bluetooth [31]. The Adafruit FeatherWing was chosen as it can handle a load of up to four motors, which is the requirement for this project. Additionally, this device can communicate with the main microcontroller via I^{2}C [32].

The hardware for this project includes four Adafruit N20 DC motors, an ESP32 board, an Adafruit FeatherWing motor driver, and a 3-axis magnetic force sensor provided by WPI Soft Robotics Lab. The ESP32 microcontroller digital reads each DC motor's encoder and the output of the motor is then managed through the motor driver by the ESP32 using I^{2}C. The hardware is connected to the custom PCB to condense the electronics and circuitry.

4.3.2 Custom PCB

A customized printed circuit board (PCB) was designed for the electrical components of this origami gripper. This PCB was designed to minimize wire jumping as well as to reduce the possibility of incorrect motor wiring. The PCB design incorporates female headers that correspond to the pins on both the ESP32 and FeatherWing motor driver. The female header pins on the board allow for easy maintenance as the boards do not need to be directly soldered to the PCB. The PCB also includes additional female header pins that are associated with the motors' high power, lower power, and encoder wires. These wired connections are illustrated in Figure 26. This PCB design is made for the first iteration triangular palm that was designed and printed in the second quarter of this project.

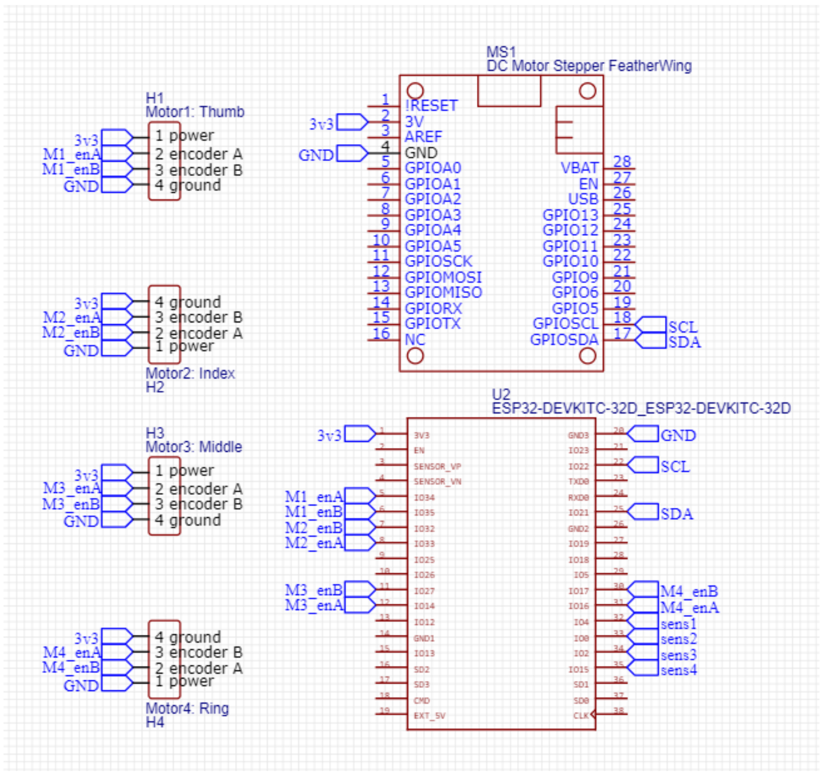


Figure 26: PCB schematic.

When the gripper's palm was redesigned as described in Section 4.1.1, there was a need for a new PCB as the original PCB shape did not fit into the new palm. While making this change, additional female header pins were added as breakout pins to support a more modular design and additional wired connections in the future. The final design of the PCB used in this project is shown in Figure 27.

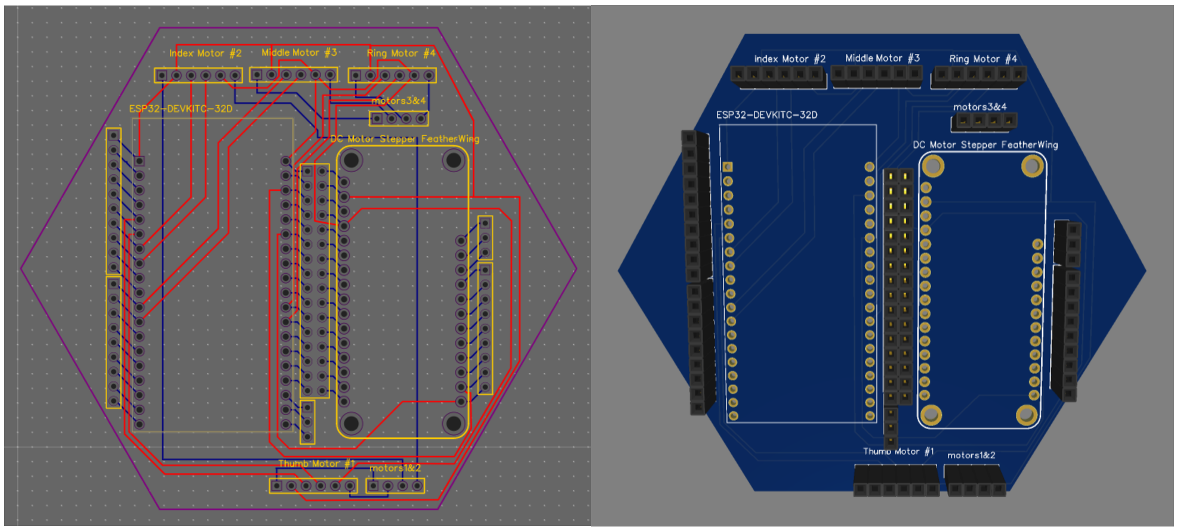


Figure 27: Hexagonal PCB design.

4.4 Controller

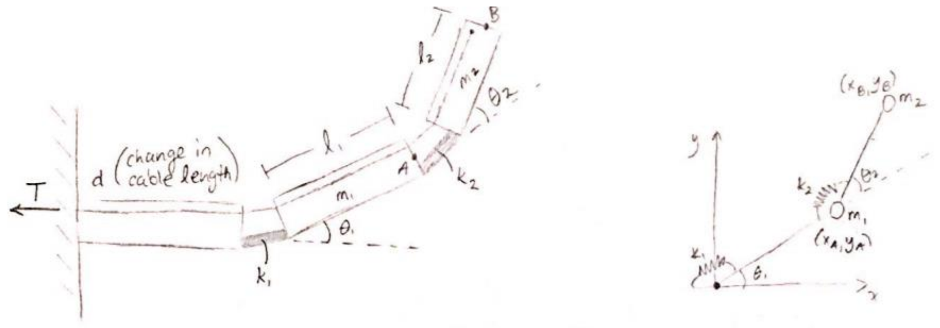
4.4.1 Motor Control Program

During the first quarter of the project, the motor control test code was written using the Arduino IDE. Although it was advantageous that the Arduino IDE provides significant documentation, it also has many limitations that made developing a multi-class program difficult. Therefore, the motor controller test code was subsequently ported from the Arduino IDE into Visual Studio Code. This transition at the end of the second quarter was necessary to ensure that classes (.cpp files) and their headers (.h files) were properly configured rather than using the Arduino .ino files. With this transition, object classes were made to further organize the structure of the code, as shown in the class diagram found in Appendix C.
The Encoder class initializes all the motor encoders and the Motor class creates an instance of an Adafruit DC motor. Using these two classes the Grasp class controls the motors and encoders to perform various grasp patterns. All of these functions are called within the main mqp_gripper class to control the actions of the gripper. As the project progressed, this program grew to incorporate classes that control the motors with proportional derivative (PD) control and that controlled the force sensors.

4.4.2 Dynamic Model and Control

Throughout the second half of the project, the team dedicated significant effort to designing the controller for the triangular beam finger. Although there are two finger designs, the team created a controller for just the triangular beam model. This is because the triangular beam finger

has a much simpler structure that can be related to rigid links and joints while the Yoshimura finger has a continuum design that would be challenging to model.
The triangular beam controller was initially designed based upon the derivations shown in Figure 28 that calculate the kinetic and potential energy of the system. The difference between the potential energy and kinetic energy is used to produce the Lagrangian function. Derivations of the Lagrangian in terms of the joint angles and time are used to compute the Euler-Lagrangian. From the Euler-Lagrangian, the trajectory of the joint angles can be followed. Lastly, ordinary differential equations are utilized to generate the time step and determine joint angles at each time step.



$$x_A = l_1 c_1 \qquad x_B = x_A + l_2 c_{12} = l_1 c_1 + l_2 c_{12}$$
$$y_A = l_1 s_1 \qquad y_B = x_B + l_2 s_{12} = l_1 s_1 + l_2 s_{12}$$

Notation
$c_1 = \cos(\theta_1)$
$s_{12} = \sin(\theta_1 + \theta_2)$

$$\dot{x}_A = -l_1 \dot{\theta}_1 s_1 \qquad \dot{x}_B = -l_1 \dot{\theta}_1 s_1 - l_2 (\dot{\theta}_1 + \dot{\theta}_2) s_{12}$$
$$\dot{y}_B = l_1 \dot{\theta}_1 c_1 \qquad \dot{y}_B = l_1 \dot{\theta}_1 c_1 + l_2 (\dot{\theta}_1 + \dot{\theta}_2) c_{12}$$

$$v_B^2 = \dot{x}_B^2 + \dot{y}_B^2 = l_1^2 \dot{\theta}_1^2 + l_2^2 (\dot{\theta}_1 + \dot{\theta}_2)^2 + 2 l_1 l_2 (\dot{\theta}_1^2 + \dot{\theta}_1 \dot{\theta}_2) c_2$$

$$KE = \underbrace{\tfrac{1}{2} m_1 l_1^2 \dot{\theta}_1^2}_{\text{link 1}} + \underbrace{\tfrac{1}{2} m_2 l_1^2 \dot{\theta}_1^2 + \tfrac{1}{2} m_2 l_2^2 (\dot{\theta}_1 + \dot{\theta}_2)^2 + l_1 l_2 (\dot{\theta}_1^2 + \dot{\theta}_1 \dot{\theta}_2) c_2}_{\text{link 2}}$$

$$PE = \tfrac{1}{2} k_1 \theta_1^2 + \tfrac{1}{2} K_2 \theta_2^2$$

Figure 28: Kinetic and potential energy derivations.

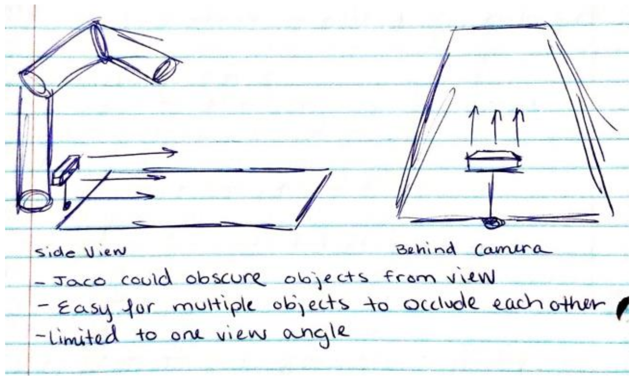
4.4.3 Simulation

The simulation of the triangular beam finger was created in MATLAB using the Robotics Toolbox created by Peter Corke. The simulation was created by designing a simple stick model of the links and joints. Once all the links were connected in a fashion that represented the serial design of the triangular beam finger, the parameters of the controller were defined. This includes proportional and derivative terms of the control as well as the starting and ending joint configurations. Next, the controller takes the inputs to generate a plot of the three joint trajectories as it moves from the starting to ending location. In addition to the plot, the MATLAB program also animates the movement of the modeled finger in a three-dimensional space.

4.5 Computer Vision

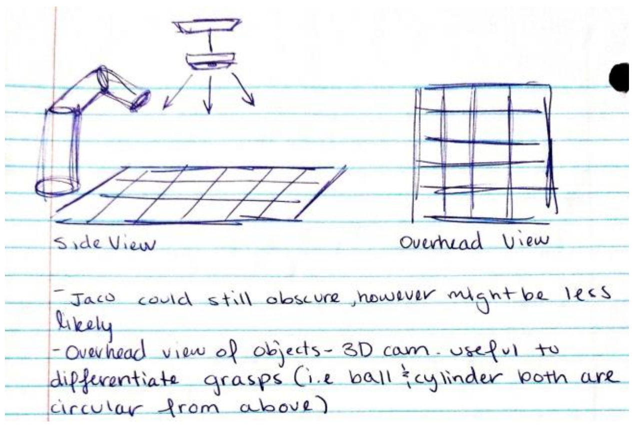
4.5.1 Camera Placement

An important aspect for the vision system involves the position of the camera in the test space and the type of camera used to collect data. A few possibilities for the vision system's camera placement are shown in Figure 29 and Figure 30. Figure 29 places the camera in front of the Jaco arm, overlooking the taskspace from a side view. A few potential issues with this position includes that the Jaco arm could obscure objects from the camera's view as it grasped them, multiple objects on the field could easily occlude one another, and the camera would be limited to a side view of all the objects. Figure 30 places the camera above the taskspace, providing a bird's eye view of the objects. This position could still lead to the camera being obscured by the Jaco arm, but if the arm approaches the object from the side there is less chance for this obstruction to occur. In the latter half of this project however, the camera position was switched to be at an angle. This final placement allows the team to leverage partial point cloud data to better match object templates.



- Jaco could obscure objects from view
- Easy for multiple objects to occlude each other
- Limited to one view angle

Figure 29: Sketch of potential placement of camera to side of Jaco taskspace.



- Jaco could still obscure, however might be less likely
- Overhead view of objects - 3D cam. useful to differentiate grasps (i.e ball & cylinder both are circular from above)

Figure 30: Sketch of potential placement of camera above Jaco taskspace.

A 3D camera was purchased to aid in differentiating objects with similar silhouettes with only 2D information. For example, from a side view, there is a distinct difference between a sphere and a cylinder, however, from above both these objects share the same profile. Without depth information, the vision system could easily classify the sphere as requiring a grasp pattern reserved for cylindrical shapes. An Intel RealSense Depth Camera D435, shown in Figure 31, was chosen due to its depth and RGB capabilities as well as its onboard vision processor board.

Figure 31: Intel RealSense Depth Camera D435.

4.5.2 Initial Experimentation

At the start of the project, the team investigated the systems used by past MQP teams as well as modern object detection techniques. Initial experimentation focused on understanding simpler object detection techniques that utilized image segmentation. The team experimented with 2D object detection with algorithms such as *watershed*, *grabcut*, and OpenCV's *matchShapes*. These initial attempts can be seen in Figure 32 and Figure 33. After further research, *matchShapes* was determined to be an unreliable manner to detect objects. For example, although Figure 32 demonstrates a high degree of accuracy for cup detections, the reading is highly influenced by the lighting, position, and size of the object in the image. In Figure 33, the *grabcut* algorithm is seen to isolate the desired cup object relatively well. However, in the second image of this figure, the table surface is not appropriately segmented from the foreground. These experiments highlighted the drawbacks of working with 2D images for object detection. Although these algorithms can be modified to improve accuracy, it was deemed more efficient to work directly with the information collected by a 3D camera.

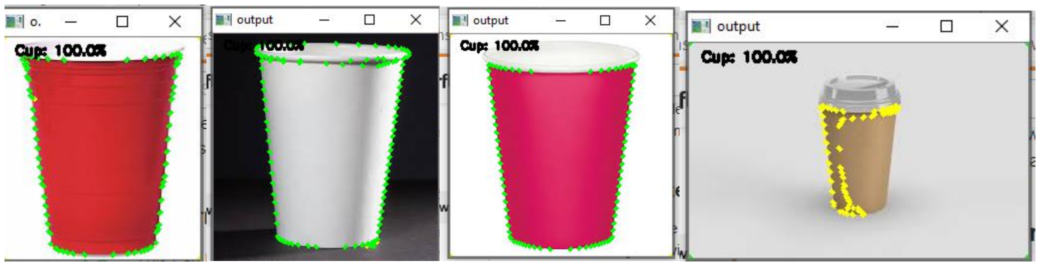


Figure 32: Image segmentation – initial attempts to use OpenCV matchShapes.

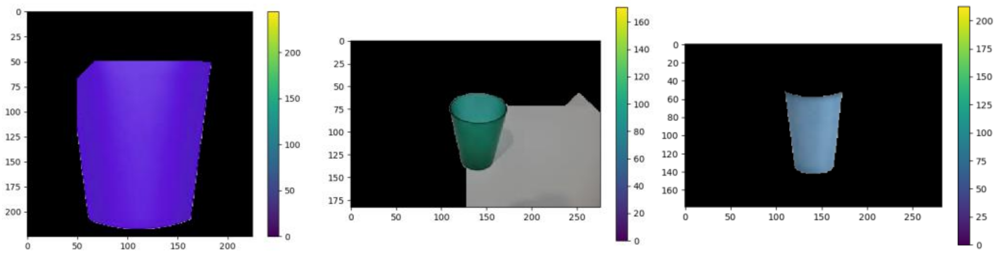


Figure 33: Image segmentation - initial attempts to use OpenCV grabcut.

Based on the research conducted during the first quarter, it was decided to further explore the possibility of using image segmentation techniques on 3D image data to perform object recognition for the gripper. Utilizing template matching along with segmentation techniques eliminated issues associated with creating a training data set for a deep learning object recognition model. As discussed in Section 2.5.2.3, training a deep learning model, even via transfer learning, takes a considerable amount of time and requires the use of a high number of training images. By exploring template matching, the limitation of generating a large amount of training data was eliminated.

For the majority of the second quarter, the camera was positioned from a top down vantage point as described in Section 4.5.1. Therefore, the original templates utilized were similar to 2D shape templates rather than 3D object templates. Figure 34 shows one such template matched to a cylindrical object viewed from the top down. The template is represented by the red dots in the figure. After the angle of the camera was switched to its final position, updated templates were generated to represent simple 3D object shapes that could be matched to 3D object point clouds. The template types used are a combination of filled, unfilled, downsampled, and non-downsampled. A filled template contains points within the area of the object while an unfilled template contains only the points that outline the surface of the object. A downsampled template consists of fewer points than a non-downsampled template as a specified number of points are filtered out. The left-hand figures in Appendix D depict the filled, non-downsampled templates used and the right-hand figures show the unfilled, downsampled templates.

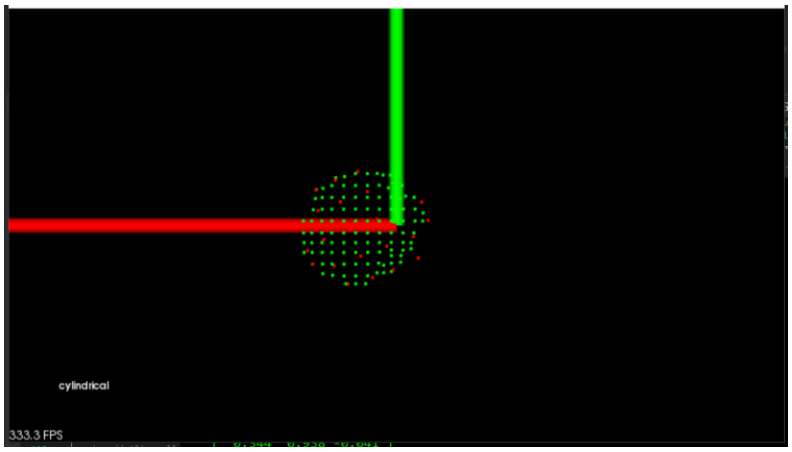


Figure 34: Original template for a cylindrical object (red points).

Appendix E displays the eight objects initially used to test template matching using the modified camera angle. The last two objects proved to be difficult to detect as they could not be distinguished from the plane that was segmented out. These test scenes translated to a point cloud similar to that in Figure 35 which showcases the object in Figure 37 in the middle of the figure. The area of interest is located between the three vertical planes in the scene. The area on the left is outside of the test area.

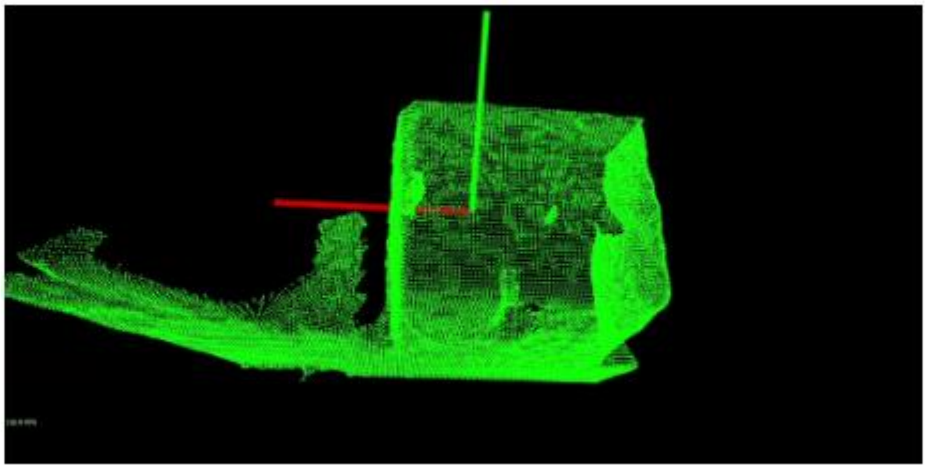


Figure 35: Raw point cloud of Figure 37 (Object is located in the center of ROI).

4.5.3 Object Detection Logic

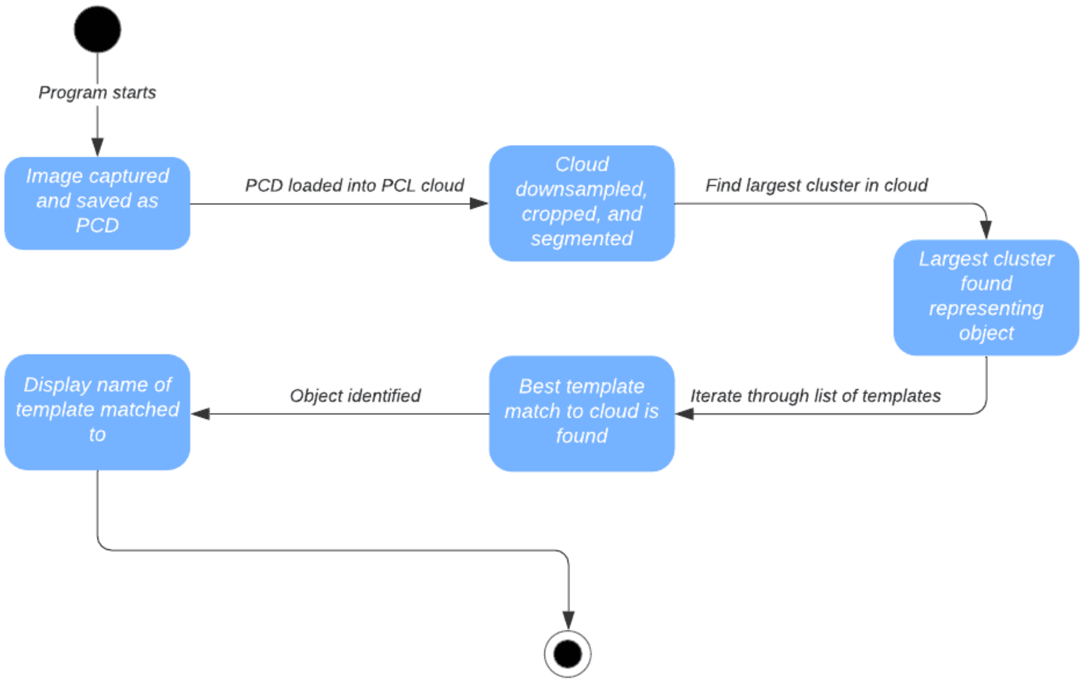

Program starts
Image captured and saved as PCD
PCD loaded into PCL cloud
Cloud downsampled, cropped, and segmented
Find largest cluster in cloud
Largest cluster found representing object
Iterate through list of templates
Best template match to cloud is found
Object identified
Display name of template matched to

Figure 36: Object recognition program logic.

The initial vision system logic flow is outlined in Figure 36. In this flow, the second step is to downsample, crop, and segment the area of interest. Following this process, the raw point cloud in Figure 35 is transformed to the point cloud cluster in Figure 37. Using the Voxel Grid filter supplied by PCL, a 3D voxel grid is created over the input point cloud and the points are then downsampled using their centroid. This reduces the number of points in the raw point cloud and improves the loading and processing time of the point cloud. To minimize the area of the point cloud, the input cloud is cropped using PCL's CropBox implementation where the points within a set of constraints are kept. The resulting point cloud is then segmented using the RANSAC algorithm to remove the major planes within the scene. Finally, PCL's Euclidean clustering algorithm is used to find the largest cluster within the final cloud. This cluster represents the object data point cluster that is to be matched with a template. Finally, the template matched to the object would depend on the object detection method used.

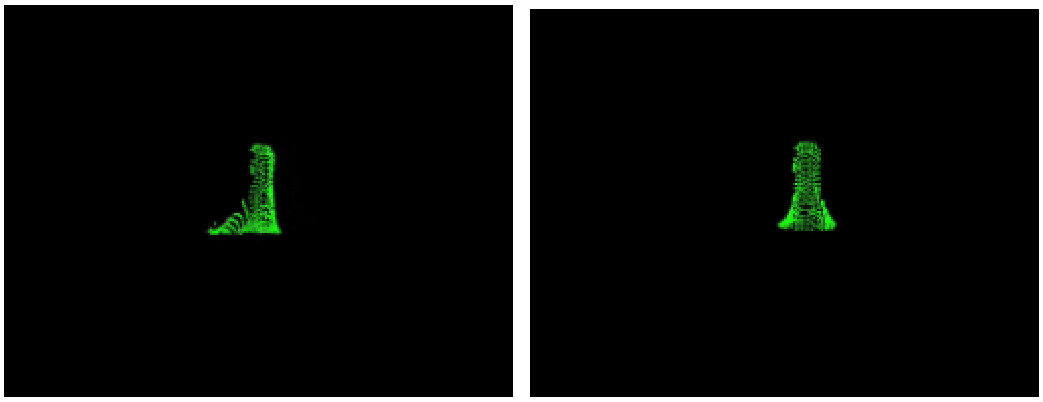
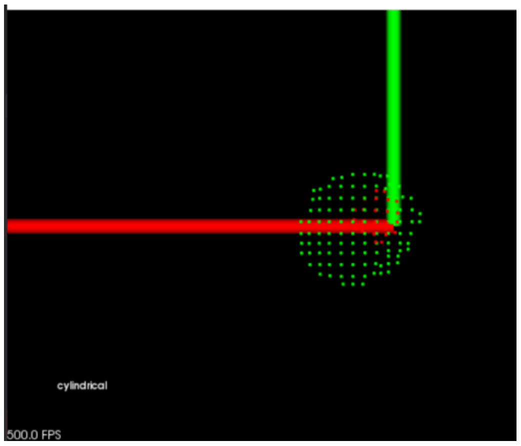


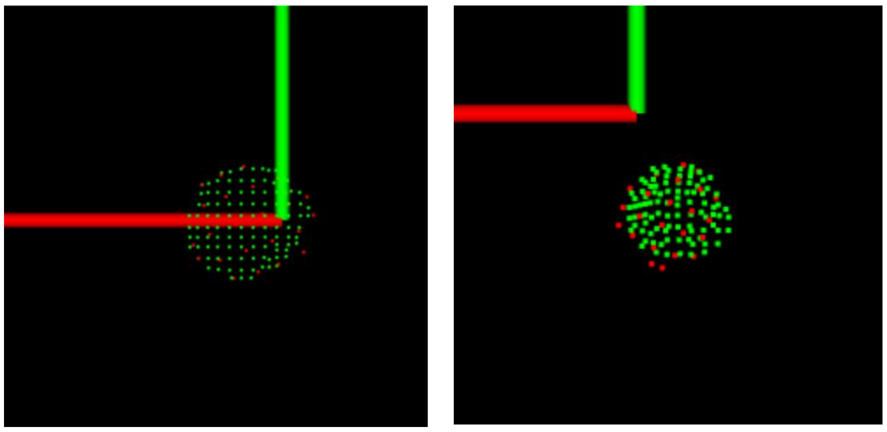
Figure 37: Side and front view of the segmented object from Figure 35.

4.5.4 Template Matching

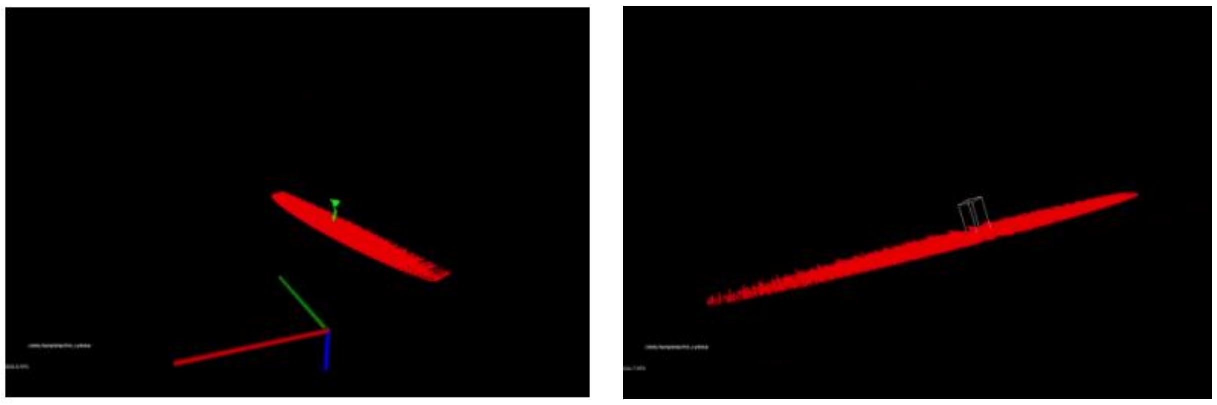
Working with templates, whether 2D or 3D, requires a transformation to find the best fit to an object's point cloud cluster. Initially, first attempts at object detection focused on aligning templates to the object using the *Sample Consensus Initial Alignment* algorithm found in PCL [26]. This method of detection is very computationally intensive as it attempts to estimate the best transformation to align an input cloud to its target. To visualize an alternative to this template alignment, the team explored scaling templates according to principal component analysis (PCA). Without scaling, it was unlikely a template would accurately match to an object. As seen in Figure 38a, the unscaled 2D template identified an object's shape by fitting to a corner of the cluster, however this template was expected to cover the entire object cluster. Initially, simple matrix transformations were used to scale the templates to match an object. However, PCL's PCA functionality was later used to resize the axis of the point clouds according to the scale derived from the target cloud. In this case, the target cloud was the object of interest and the template cloud was the cloud that was scaled. Figure 38b and Figure 38c shows initial attempts at scaling 2D templates using the PCA method. Figure 38b was seen to be successful in modifying the scale of the red template cloud to cover the entire object. However, Figure 38c initially failed as each cloud's axis was scaled with different principal components.



a: *Red, unscaled template matching to the cluster*



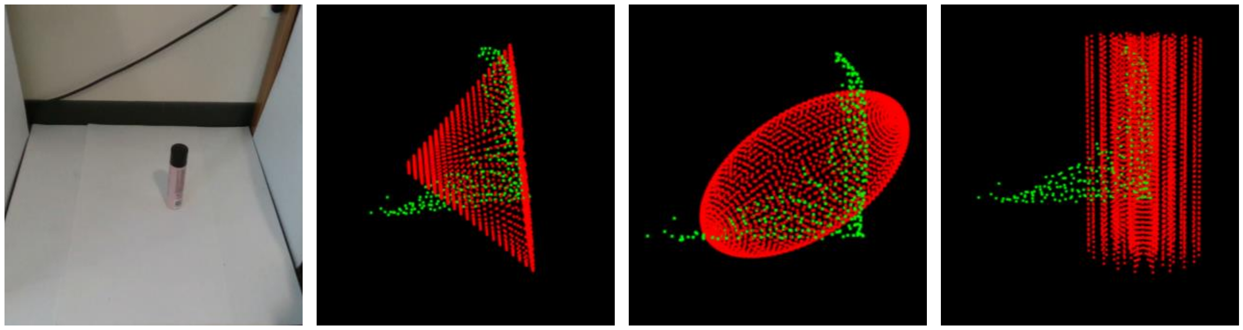
b: *Successful scaling using PCA to match to two different cylindrical objects*



c: *Initial failed attempt at scaling a 3D template using PCA - the Z axis did not scale as intended*

Figure 38: Scaling templates using PCA.

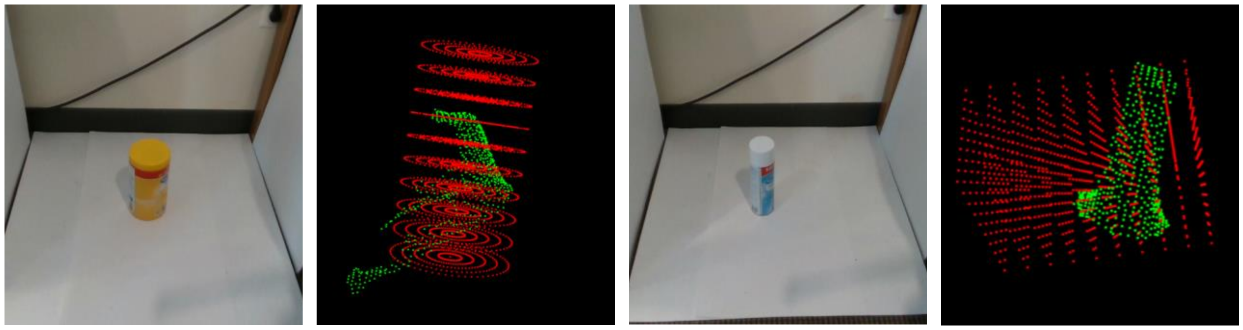
Moving to 3D templates, non-downsampled and filled templates were first tested. These templates took too long to load into the program and to scale to an object's cluster. An additional problem was that shadows cast by an object created trailing points behind the main face of the object as seen in Figure 39a. This caused incorrect templates to be matched to the object cloud as the algorithm tried to align to these points. In Figure 39b and Figure 39c, the expected outcome was that the thin cylinder template would be matched to *cylinder_0*, however, due to the trailing points, the cone and sphere were determined to be better matched. On removing these from the list of possible templates, the thin cylinder was observed as the best match and the alignment of the template was as expected.



a: *cylinder_0* ***b:*** *Cone matched* ***c:*** *Sphere matched* ***d:*** *Thin cylinder matched*

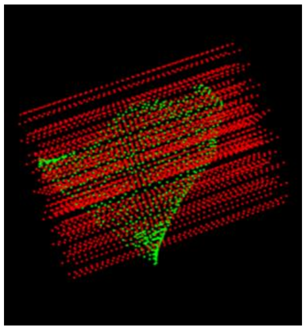
Figure 39: Non-downsampled template matching with cylinder_0.

The following figures showcase the same issue as above. In the case of *cylinder_1* in Figure 40b, sometimes the correct template was chosen, but the template was not aligned as expected.

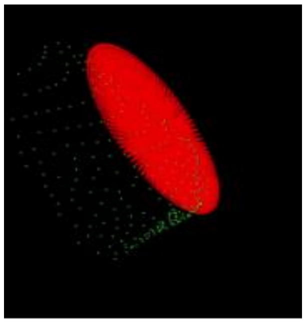


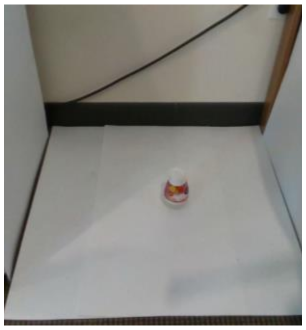
a: *cylinder_1* ***b:*** *Cylinder matched to cylinder_1* ***c:*** *cylinder_2* ***d:*** *Rectangle matched to cylinder_2*

e: *hook_0*


f: *Thin cylinder matched to hook_0*

g: *hook_1*


h: *Sphere matched to hook_1*


i: *tripod_0*

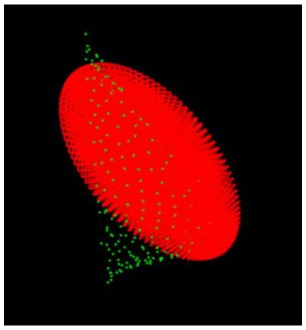
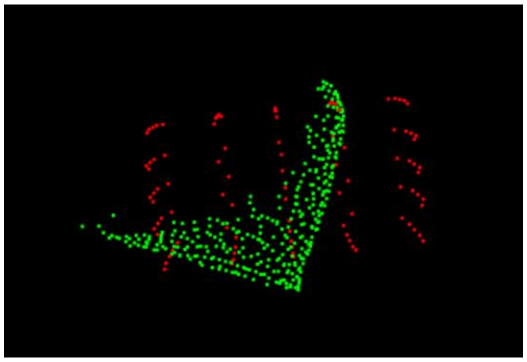

j: *Sphere matched to tripod_0*

Figure 40: Template matching with multiple objects.

On testing downsampled and unfilled templates, the team found that the loading time of the templates into the program as well as the time to find the best alignment improved drastically. Although the matching was still slower than expected, it was better than using the non-downsampled templates. Figure 41 shows how the individual templates are aligned for *cylinder_0*. As noted previously, the template attempts to encapsulate all points of the cloud, thereby not aligning as expected. In all these cases, the trailing points of the cluster interfere and alter the alignment of the template.


a: *Match to rectangle template*

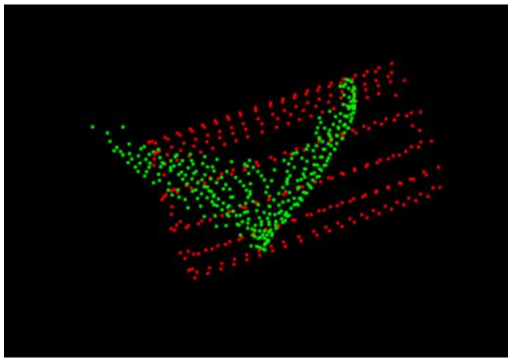

b: *Match to thin cylinder template*

Figure 41: Downsampled template matching with cylinder_0.

In experimenting with aligning a template to best fit a cluster, a few issues were made clear. The first being that a template would scale to best fit the given cluster, this was an issue when the object contained unexpected points in its cluster. Additionally, a template would not maintain its aspect ratio when scaled as each axis was individually scaled. This meant that an incorrect template was chosen when too many of its points were matched to an incorrect position on the object. This led to the problem that although a correct template was chosen, it may not have been fit to the object as expected.

4.5.5 Principal Component Analysis

PCA essentially reduces the dimensionality of data by calculating principal components i.e axes where the data has the most variance. The first component attempts to encompass the maximum amount of data possible and then attempts to encompass the rest in the 2nd and 3rd component. Because of this, we can eliminate the information of one of the last components if necessary as they will hold the least amount of information. By calculating the components of both a template and an object of interest, we can then scale and compare the components to determine the best match.

The initial intention of using PCA was to visually scale the templates to match the objects of interest. However, in the latter half of the project the issues involving the system not accurately matching the correct template to the object present in the scene and the time it took to scale were solved by leveraging PCA as the method of object detection. On initialization of the program, the user provides a set of point cloud data templates whose principal components are calculated. When the system must detect an object, it segments it out of its background and calculates the object’s principal components as well. After scaling a template’s first component to match that of the object, the second and third components then determine the template the object matched with. This match occurred in 1408249 microseconds, or approximately 1.4 seconds.

4.6 System Architecture

This project required an integrated system for the components involved. Because the team ultimately needed to integrate the gripper as an end effector to the Jaco arm, it was decided that ROS would be used to orchestrate the system. ROS has various available packages that facilitate the integration with the Jaco arm as well as with the gripper’s individual components. The team planned to leverage a controller node to coordinate the interaction between the nodes of the system. Originally, the nodes planned to include a camera, processing, robot, and gripper node that would have a publisher and subscriber relationship. The preliminary design of this architecture can be seen in Figure 42 where the nodes are represented by circles and topics by rectangles. The topics these nodes publish and subscribe to hold information necessary for other nodes to properly react to changes in the system and environment. The controller then subscribes to the majority of these topics to coordinate the actions of the nodes. In this diagram, the camera node represents the Intel RealSense depth camera, the robot node the Jaco arm, the processing node the object detection logic, and the gripper node the origami gripper. The gripper node in particular holds the motor control code defined in Section 4.4.

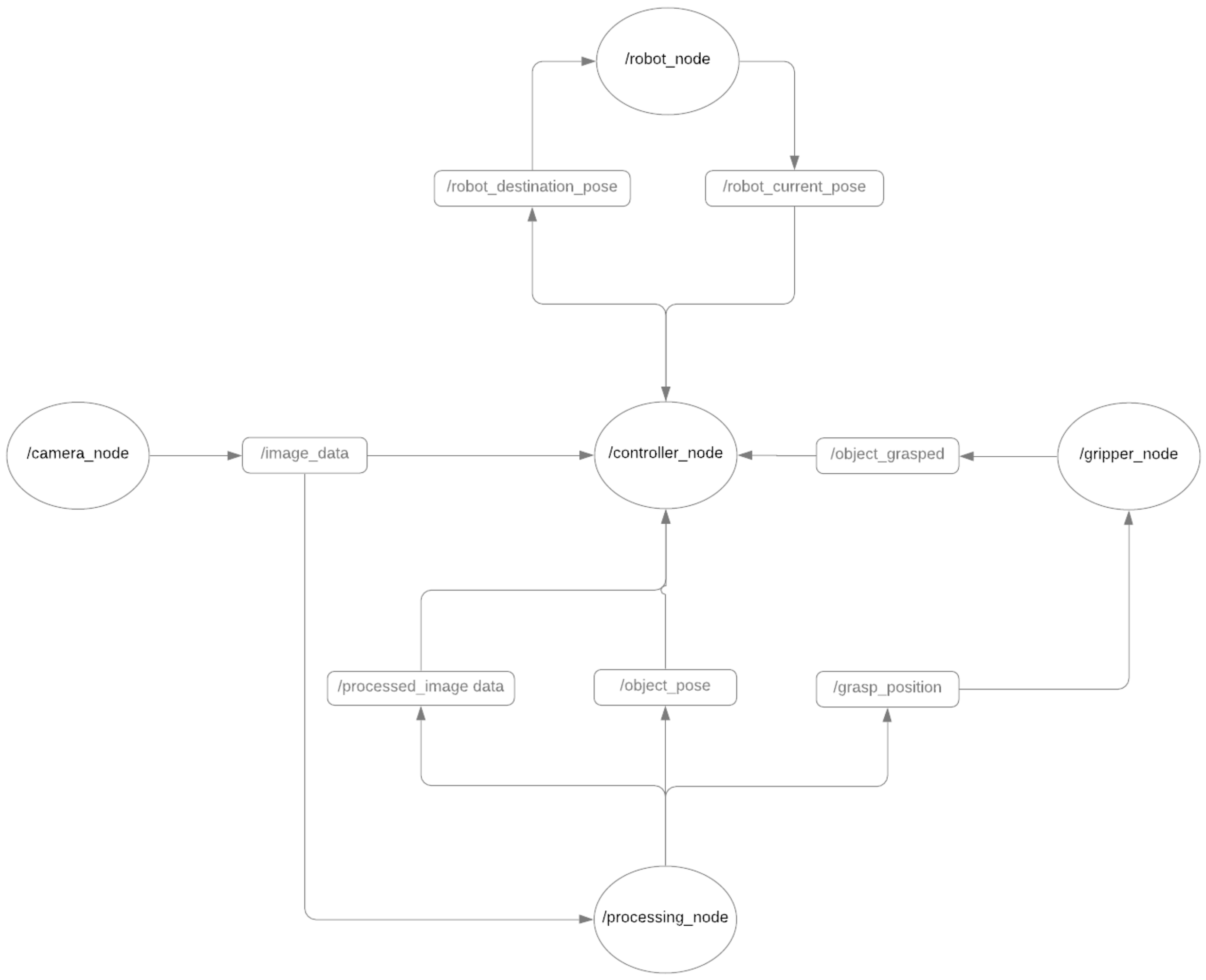


Figure 42: Preliminary ROS software architecture.

5. Results

In this section, we discuss both the final status of the project as well as the final design and implementation of our overall gripper system.

5.1 Flexible Robotic Origami Gripper System

Section 3.2 lists task specifications that were outlined at the beginning of the project that identified the guidelines and parameters of the MQP. Below is a table that summarizes the status of these task specifications at the conclusion of our project. As can be seen in Table 3, 27 of the 30 task specifications were met.

Table 3: Task Specification Completion Status

#	Task Specification	Complete?
1	The total cost of the project should not exceed $600 (MQP student budget).	✓
2	The maximum weight of the gripper including its onboarded motors must not exceed 333 grams (weight of Ann Marie’s hand).	✓
3	The payload weight should be a minimum of 50 grams (the weight of a small ball).	✓
4	The gripper must be able to successfully grasp an object up to an 8cm diameter/width (or have a crevice or protrusion for the gripper to hold it).	✗
5	In a closed gripper position, the gripper should be confined to a 25 cm cube.	✓
6	The gripper will contain origami modules.	✓
7	The gripper will contain a single modular palm structure with 2 sets of interchangeable origami fingers.	✓
8	There must be at least 2 different origami finger designs.	✓
9	The gripper will have at least 3 fingers (maximize grasp potential).	✓
10	There should be at least one force sensor per gripper finger.	✗
11	The gripper in coordination with the Jaco arm must reach a final grasp position within 30 seconds.	✓
12	The gripper must close around the object within 1 second.	✓
13	The gripper must be able to grasp objects that do not require a change in wrist orientation.	✓
14	The gripper should have a maximum of 1 motor per finger.	✓
15	The gripper should have a maximum of 4 motors per gripper.	✓
16	Motors should fit within the palm of the gripper.	✓
17	The gripper must not shake to the point of gripping failure.	✓
18	The system must be able to respond to input from the vision system.	✓
19	An object must be identified by the vision system within less than 120 seconds.	✓

20	The vision system must be able to detect the shape of an object to indicate the appropriate grasp for the gripper.	✓
21	The vision system must be able to accurately recognize an object within the Jaco arm's task space that is fully present in the camera's field of view (i.e not expected to recognize occluded objects).	✓
22	The object will be recognized using partial object shape data (i.e it will identify a soda can as a cylinder).	✓
23	The vision system will determine the location of the object to a degree of accuracy where it will be able to successfully grasp the object.	✗
24	The camera size should not exceed 90 mm x 25 mm x 25 mm.	✓
25	The vision system and the gripper will use up to 2 total processing units.	✓
26	System architecture should be designed in a modular fashion.	✓
27	The vision system should utilize an RGB-D camera to capture depth information.	✓
28	Wires should have enough slack so that they do not unplug from ports.	✓
29	The gripper design should ensure the mechanical and electrical components are easily accessible if maintenance is required (should not require taking apart the entire module if one component is in disrepair).	✓
30	The gripper should not endanger any humans.	✓

Task specification #2 references the requirement to design a gripper with a maximum weight of 333 grams. Table 4 displays a detailed breakdown of the gripper's weight. Listed under "Parts" are the individual components that make up the origami gripper. The "Assemblies" section lists the masses of the assembled triangular beam finger gripper and Yoshimura finger gripper.

Table 4: Origami gripper masses

Mass (g)	**Parts**
173.65	Assembled palm
10.087	1 triangular beam finger with spool
6.463	1 Yoshimura with spool
57.35	PCB
15.4	1 single motor with wires
Mass (g)	**Assemblies**
332.95	Assembled gripper with triangular beam fingers
318.45	Assembled gripper with Yoshimura fingers

5.2 Final Gripper

The designing, prototyping, and initial testing culminated in the following gripper design. The gripper consists of four origami fingers and a multilevel 3D printed gripper palm, shown in Figure 43.

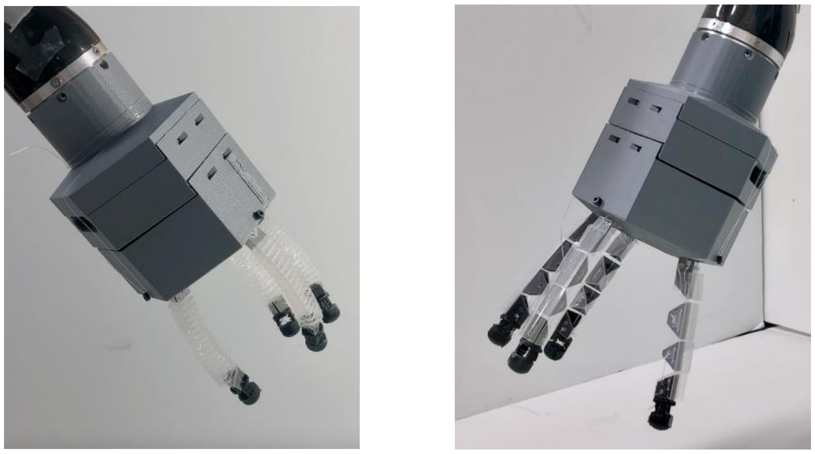


Figure 43: Final gripper design with Yoshimura fingers (left) and triangular beam fingers (right).

5.2.1 Palm Structure

Each layer of the gripper's multilayer palm provides a different function that helps connect the fingers to the Jaco arm robot. The base of the palm, seen in Figure 44 as the highlighted layer in the leftmost image, is where the connectors of each assembled finger are screwed into the palm. The connectors are angled at 14.04 degrees. This allows the robot to grasp objects up to 6.2 cm wide. The midsection of the palm, shown in the center image of Figure 44, ensures the proper placement of the custom PCB. The slot on the side of this layer aids in the cable management of multiple power wires. Lastly, the lid of the palm, pictured in the rightmost image of Figure 44, maintains a secure connection between the gripper and the Jaco arm.

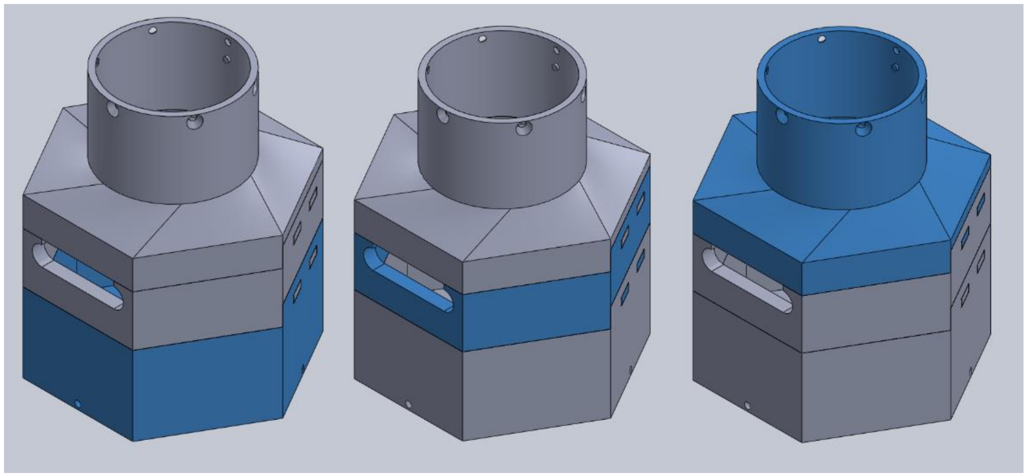


Figure 44: The complete multilayer gripper highlighting the palm (left), middle layer (center), and lid (right).

5.2.2 Origami Fingers

The final origami finger designs can be seen below. Each Yoshimura finger weighs approximately 6.5 g and measures a length of 11.1 cm. Meanwhile, the triangular beam fingers each weigh approximately 10 g and measure a length of 11.6 cm. Both are tendon-driven with fishing line and are actuated by a single Adafruit N20 DC motor per finger. When contracted, the fingers appear as shown in Figure 45. In contrast, when no tension is applied to the fishing line by the motor, the fingers are relaxed as shown in Figure 46.

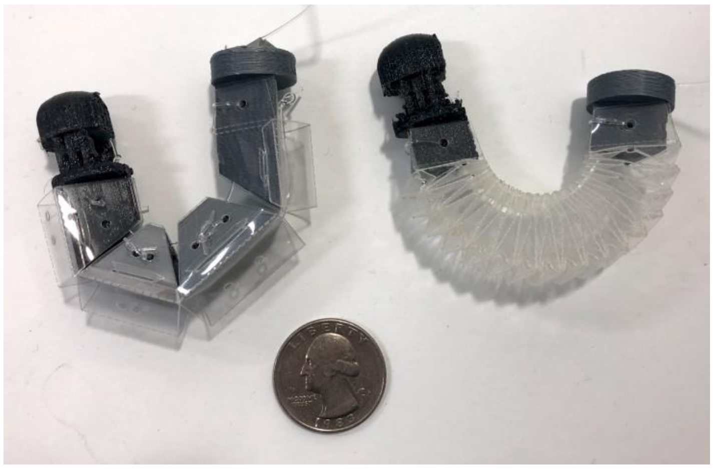


Figure 45: Triangular beam finger (left) and Yoshimura finger (right) shown in the closed configuration.

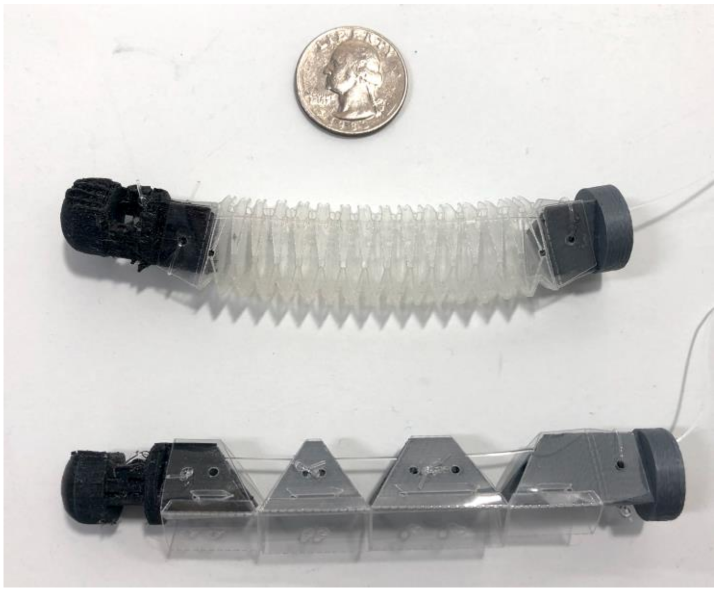


Figure 46: Yoshimura finger (top) and triangular beam finger (bottom) with fingertips, connectors, and triangular beam endoskeleton.

5.2.3 Grasp Configurations

The gripper is capable of four types of finger configurations or grasps: open, pinch, tripod, and power, seen in Figure 47. In the open configuration, no motors are engaged, and all the fingers are relaxed. For the pinch grasp, motors for the thumb and middle finger contract their respective fingers to perform the pinch. The tripod grasp requires three motors to run for the thumb, index, and ring finger. Finally, the power grasp utilizes all four motors, so all the fingers are contracted.

The gripper was able to grasp three types of objects with both the triangular beam and Yoshimura fingers, as seen in Figure 48. The pinch grasp is used for thin cylindrical objects, the tripod grasp is used for spherical objects, and the power grasp is used for sturdy cylindrical objects. The spherical objects had to be placed on a support stand so they would not roll away during the execution of the grasp. For the power grasp, the gripper required human assistance to lift the object to a position at which the gripper could secure a stable hold on the object. A common problem with grasping was the fingertips have a low coefficient of friction which resulted in objects slipping before a stable grasp was achieved. From our testing, we found that the performance of the triangular beam varied from that of the Yoshimura finger. Since the triangular beam finger is made up of multiple rigid links, the fingers struggled to conform to the shape of certain objects despite the flexible joints. In contrast, the Yoshimura fingers were better able to conform to the objects than the triangular beam fingers as the Yoshimura fingers exhibit a continuum design.

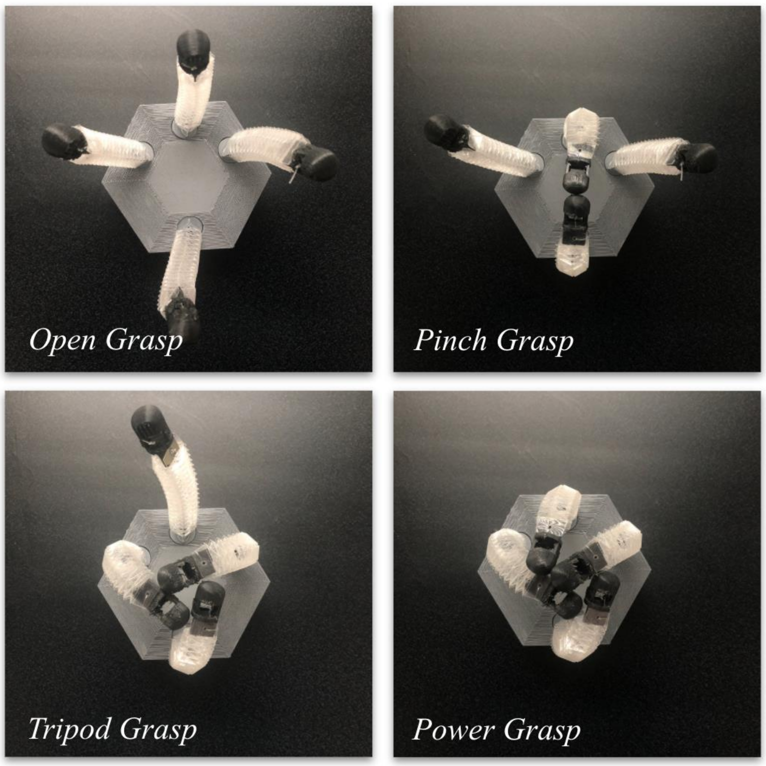


Figure 47: The open grasp and three closing grasp types of the gripper.

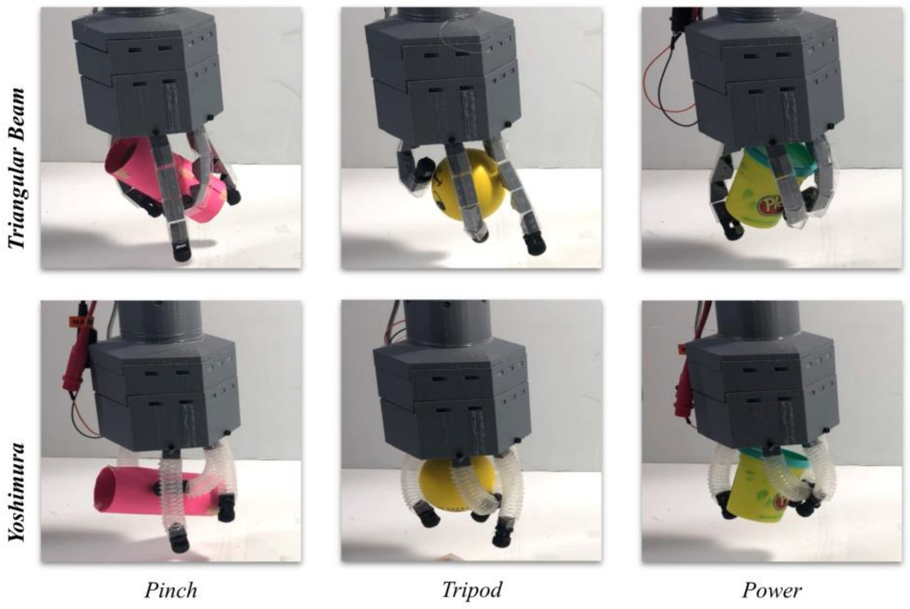


Figure 48: Gripper with triangular beam and Yoshimura fingers grasping 3 types of objects.

5.3 Control System

The control system for our origami robotic gripper consists of the electronics, the motor control C++ program, and a MATLAB simulation of expected results. The electronics are all connected to a custom PCB designed by the team to improve cable management. The Espressif ESP32 Dev Kit and Adafruit FeatherWing motor drivers are attached to the top of the PCB while the headers pins that connect to the four motors are mounted to the bottom. With the boards connected to the top side of the PCB, it is simpler to reset or enable the boards when necessary. The header pins for the motors soldered to the bottom side of the PCB as it is closer to the motors. Figure 49 shows the motors and boards attached to the PCB.

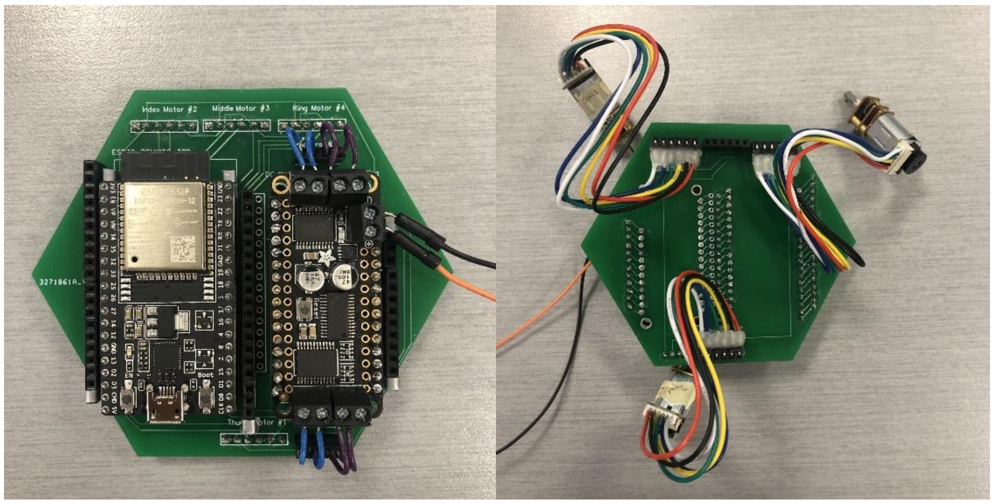


Figure 49: Top view of PCB (left) and bottom view of PCB (right).

To program the electronics to perform the functions we desired, we created a motor control program in C++. The control class diagram of this multiclass program is illustrated in the class diagram in Appendix F. This program is integrated with ROS to determine the grasps required for different objects. Once the program is given a command, the gripper will execute the specified grasp. The controlled encoder and force sensor readings are used to control the movement of the motors. The setpoints for each finger of the opened and closed grasps are configured as class members. The controller programs the motors to continuously rotate to the desired setpoints. The Encoder, Motor, and Force Sensing classes all flow into the Grasp class. The configurations and initializations for each component are found in their respective classes. The Control class applies the controller to the inputs of the encoders and sensors to produce the outputs of the motor.

The last piece of the control system was the controller design, which was designed in MATLAB. This controller originally used the potential energy and kinetic energy equations of the triangular beam finger to produce the Lagrangian equation. The Lagrangian equation is simply the difference of the kinetic energy and potential energy. The Lagrangian equation then provides the foundation for the Euler-Lagrangian equation, which helps determine the trajectory of the joint angles. The Lagrangian, as well as the Euler-Lagrangian equations for both joints, are written out below.

$$E_K = 0.5 * m_1 * {l_1}^2 * {\dot{\theta}_1}^2 + 0.5 * m_2 * {l_1}^2 * {\dot{\theta}_1}^2 + 0.5 * m_2 * {l_2}^2 * \left(\dot{\theta}_1 + \dot{\theta}_2\right)^2 + l_1 * l_2 * \left({\dot{\theta}_1}^2 + \dot{\theta}_1 * \dot{\theta}_2\right) * \cos(\theta_2)$$

$$E_P = 0.5 * k_1 * {\theta_1}^2 + 0.5 * k_2 * {\theta_2}^2$$

$$Lagrangian = E_K - E_P$$

Figure 50: Lagrangian equation.

$$Joint\ 1\ Euler\ Lagrangian:$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_1} - \frac{\partial L}{\partial \theta_1} - \tau_1 = 0$$

$$\left[{l_1}^2 * {m_1}^2 * \ddot{\theta}_1 + {l_1}^2 * {m_2}^2 * \ddot{\theta}_1 + {l_2}^2 * m_1 * \left(\ddot{\theta}_1 + \ddot{\theta}_2\right) + l_1 * l_2 * \cos(\theta_2) * \left(2 * \dot{\theta}_1 + \dot{\theta}_2\right) - l_1 * l_2 * \sin(\theta_2) * \dot{\theta}_2 * \left(2 * \dot{\theta}_1 + \dot{\theta}_2\right)\right] - [-k_1 * \theta_1] - \tau_1 = 0$$

$$Joint\ 2\ Euler\ Lagrangian:$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_2} - \frac{\partial L}{\partial \theta_2} - \tau_2 = 0$$

$$\left[m_2 * {l_2}^2 * \ddot{\theta}_1 + m_2 * {l_2}^2 * \ddot{\theta}_2 + l_1 * l_2 * \ddot{\theta}_1 * \cos(\theta_2) - l_1 * l_2 * \dot{\theta}_1 * \sin(\theta_2) * \dot{\theta}_2\right] - \left[-k_2 * \theta_2 - l_1 * l_2 * \sin(\theta_2) * ({\dot{\theta}_1}^2 + \dot{\theta}_1 * \dot{\theta}_2)\right] - \tau_2 = 0$$

Figure 51: Euler Lagrangian equations.

Since the Euler-Lagrangian equations for both joints are equivalent to zero, MATLAB was able to determine the joint acceleration of both joints using the solve function. Next, the team implemented ordinary differential equations to produce the time step used to track the trajectory of the joint angles. The torques from the Euler-Lagrangian equations were derived in terms of the input force, which in our application was the tension applied by the motor. The torque equations are found in Figure 52.

$$\tau_1 = -l_1 * \sin(\theta_1) * (F * \cos(\theta_1)) + l_1 * \cos(\theta_1) * (F * \sin(\theta_1))$$

$$\tau_2 = -(l_1 * \sin(\theta_1) + l_2 * \sin(\theta_1 + \theta_2)) * (F * \cos(\theta_1 + \theta_2)) + (l_1 * \cos(\theta_1) + l_2 * \cos(\theta_1 + \theta_2)) * (F * \sin(\theta_1 + \theta_2))$$

Figure 52: Joint torque equations.

The results produced by the MATLAB controller are given in Figure 53, each plot illustrates the predicted trajectory of the joint angles.

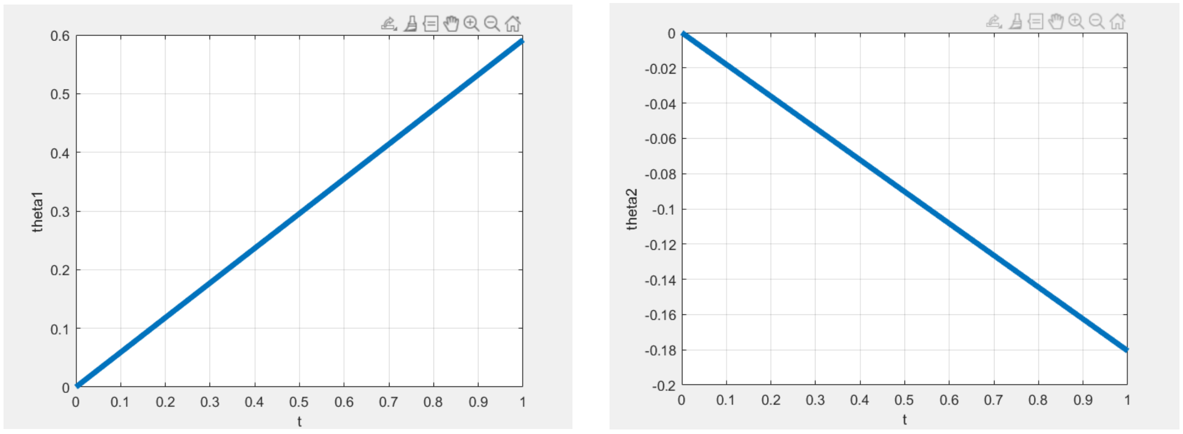


Figure 53: Diverging plots from the original controller.

The plots of the joint angles produced by the controller appear to diverge, an issue that is further addressed in Section 6.2. Considering the plots from the controller were diverging, the team decided to also implement a controller based upon a dynamic model by Jinho Kim et al. [33]. In their research, the team had created a dynamic model that reflected a three-link system, similar to our triangular beam finger. By inputting the parameters based on the measurements and properties of our triangular beam finger, the new dynamic model produced the following plot, shown in Figure 54.

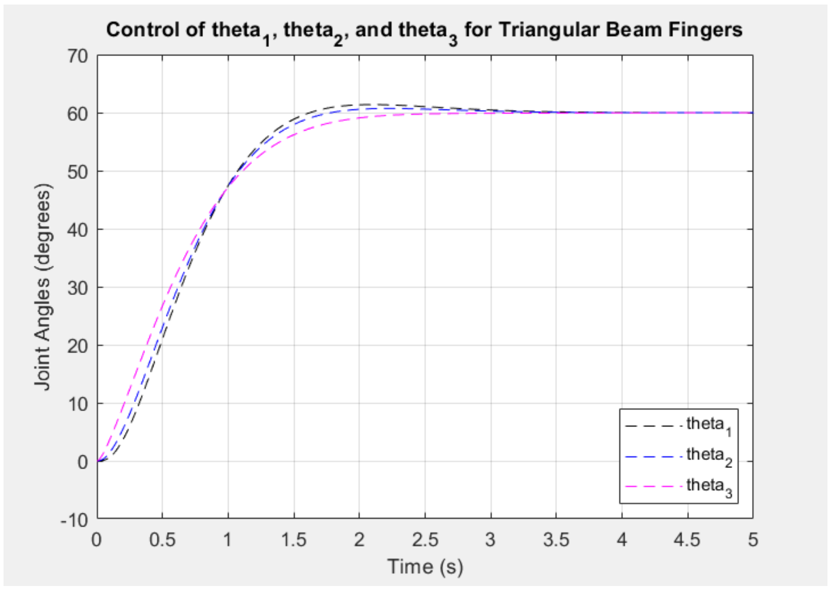


Figure 54: Plot of joint angles of triangular beam finger control.

As can be seen, all three joint angle trajectories are plotted on the same graph. It appeared that the first joint and second joints overshot before reaching the setpoint. Meanwhile, the third joint appeared to slowly approach the setpoint without any overshoot.

In addition to the joint angle plot, a simulation was created to further illustrate the change of joint angles as a force was applied to the system. The simulation represented a single origami triangular beam finger. As seen below in Figure 55, the model had four links, just like the triangular beam. However, when the triangular beam finger was mounted onto the gripper, the base link was stationary. Since the fourth link of the system acts as a ground, the finger still represents a three-link system. This simulation works well as it replicates the movement of the triangular beam joints when it closes. The MATLAB code for the controller model as well as the simulation can be found in Appendix G.

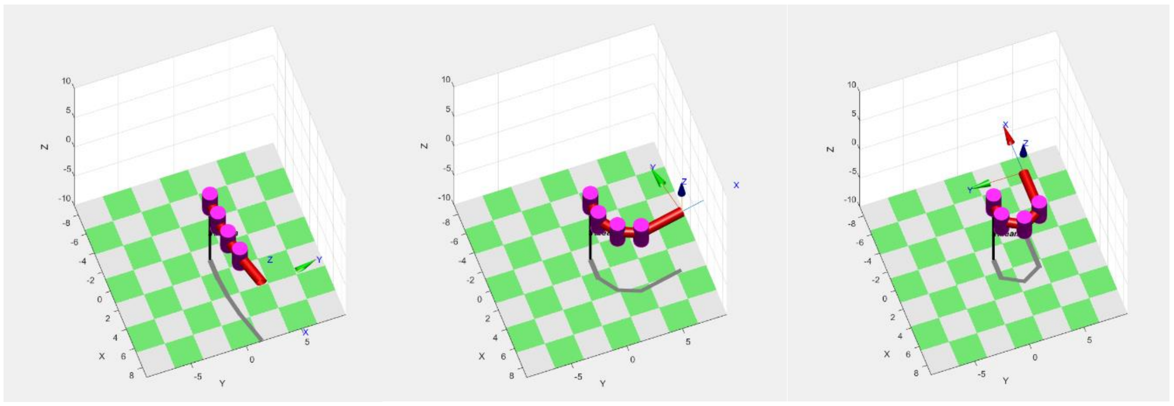


Figure 55: Simulation of a single triangular beam finger.

5.4 Object Detection

To test the implementation of the object detection system, three test objects were chosen to encompass the three grasp configurations of the gripper. As described in Section 4.5, the final object detection system used PCA to classify these objects, shown in Figure 56, according to pre-selected down sampled and unfilled templates. From left to right, the grasp type needed for each object was pinch, tripod, and power. The templates that correlated to each grasp are listed in Table 5 along with the identification number used to recognize the templates in experiments.

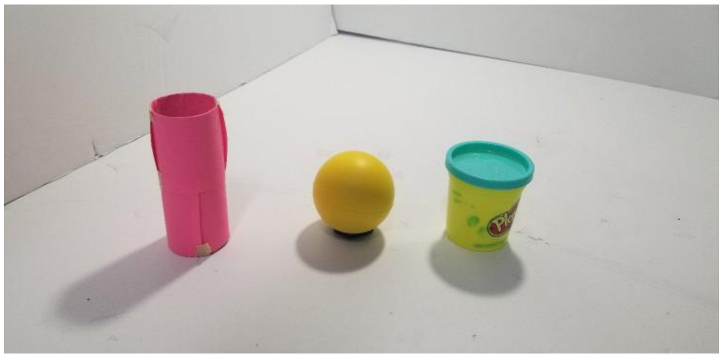


Figure 56: Final set of test objects.

Table 5: Templates corresponding to each grasp configuration.

Grasp	Template	ID #
Pinch	Thin Cylinder	1
Tripod	Cone	2
	Sphere	3
Power	Rectangle	4
	Cylinder	0

The figures below outline the process of identifying each of the selected objects. Figure 57 demonstrates the process for a thin cylindrical object that requires a pinch grasp. The upper left corner of the figure displays the object from the camera's point of view. The next image highlights the region of interest in the test scene while the right-hand image shows the identified clusters in this region. The cluster in red represents the point cloud of the object of interest and in the lower left image, the point cloud is seen overlaid over the initial test scene. Upon finding this cluster, the algorithm iterates through the five templates and determines that the thin cylinder best matches the target object, shown in the fifth sub image in red.

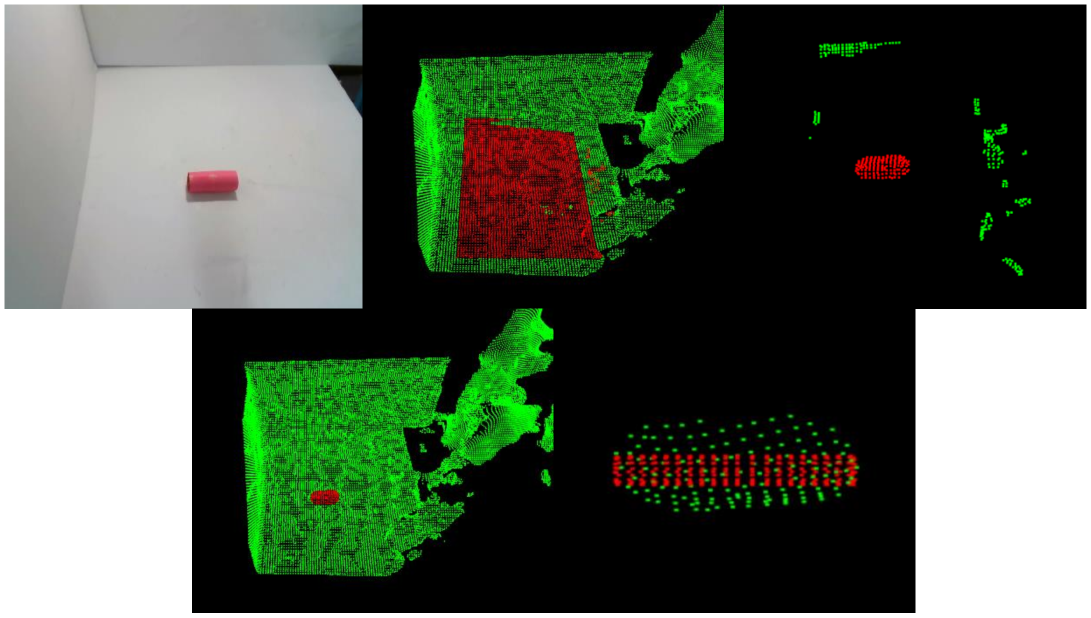


Figure 57: Process of identifying pinch configuration.

Using PCA on the target object's point cloud from Figure 57 yielded the eigenvalues listed in Table 6. These values were then used to calculate the best template match by comparing them to each of the template's calculated eigenvalues as well. The result of this process is shown in depth in Table 7. The three principal components (PC) are listed with both their scaled and unscaled values. As shown in the table, PC1 is scaled to that of the object's and then the absolute error between the template and object's eigenvalues is calculated. Per these calculations, template 1 was determined to be the closest match to the object. As seen below, the percent error for template 1 for both PC2 and PC3 is the least among all the templates.

Table 6: Eigenvalues corresponding to the pink test object.

	PC1	**PC2**	**PC3**
Object Eigenvalues	1.38E-03	5.47E-04	6.23E-05

Table 7: Eigenvalues of all templates compared to the first object's eigenvalues.

	Template 0 (Cylinder) Eigenvalues			
	Unscaled	**Scaled**	**Absolute Error**	**Percent Error**
PC1	3.80E-01	1.38E-03	0.00E+00	0.00%
PC2	3.62E-01	1.32E-03	7.68E-04	140.38%
PC3	1.70E-01	6.17E-04	5.54E-04	889.73%
	Template 1 (Thin cylinder) Eigenvalues			
	Unscaled	**Scaled**	**Absolute Error**	**Percent Error**
PC1	1.02E-01	1.38E-03	0.00E+00	0.00%
PC2	5.04E-03	6.85E-05	4.79E-04	87.48%
PC3	4.38E-03	5.95E-05	2.80E-06	4.49%
	Template 2 (Cone) Eigenvalues			
	Unscaled	**Scaled**	**Absolute Error**	**Percent Error**
PC1	2.31E-01	1.38E-03	0.00E+00	0.00%
PC2	2.25E-01	1.34E-03	7.97E-04	145.75%
PC3	6.33E-02	3.78E-04	3.15E-04	505.94%
	Template 3 (Sphere) Eigenvalues			
	Unscaled	**Scaled**	**Absolute Error**	**Percent Error**
PC1	3.38E-01	1.38E-03	0.00E+00	0.00%
PC2	3.33E-01	1.36E-03	8.12E-04	148.46%
PC3	3.27E-01	1.34E-03	1.27E-03	2045.26%
	Template 4 (Rectangle) Eigenvalues			
	Unscaled	**Scaled**	**Absolute Error**	**Percent Error**
PC1	1.10E-01	1.38E-03	0.00E+00	0.00%
PC2	1.10E-01	1.38E-03	8.34E-04	152.35%
PC3	1.10E-01	1.38E-03	1.32E-03	2116.05%

For the last two test objects, Table 8 and Table 9. provide a summarized overview of the matched template. Only the matched template's eigenvalues are displayed within these tables. Like the first object, the object in Figure 58 is identified and matched best to template 2, the cone template. The last two images in the figure show the alignment of the object to the cone template from a side view as well as a bottom-up view. Ultimately, this object is determined to need a tripod grasp based on Table 3. Likewise, the third object in Figure 59 is matched to template 0 and shown to require a power grasp. Ultimately, the object detection method chose the grasp configuration that was expected for each of the three objects.

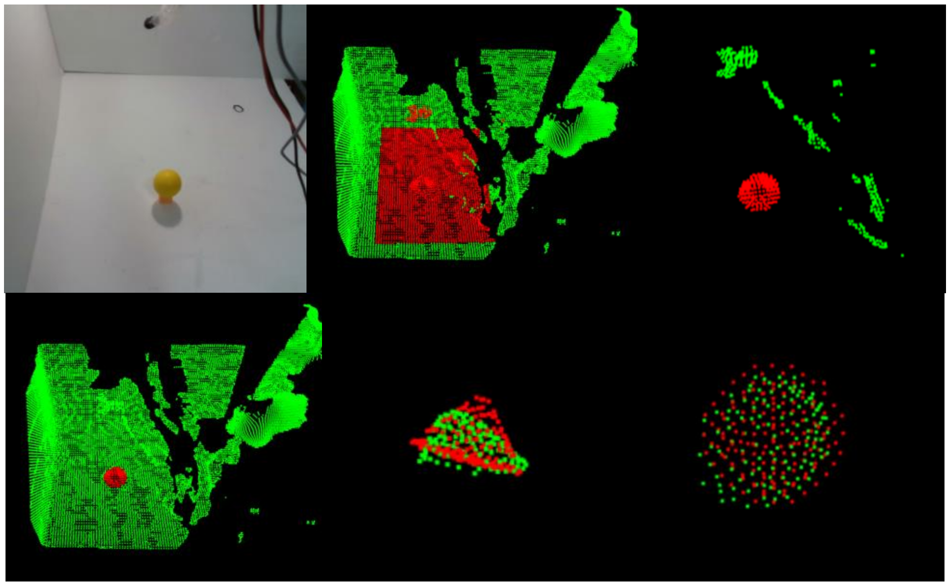


Figure 58: Process of identifying tripod configuration.

Table 8: Eigenvalues of template 2 compared to the second object's eigenvalues.

	Template 2 (Cone) Eigenvalues				Object Eigenvalues
	Unscaled	**Scaled**	**Absolute Error**	**Percent Error**	
PC1	2.31E-01	8.06E-04	0.00E+00	0.00%	8.06E-04
PC2	2.25E-01	7.85E-04	2.55E-04	48.05%	5.30E-04
PC3	6.33E-02	2.20E-04	1.10E-04	99.64%	1.10E-04

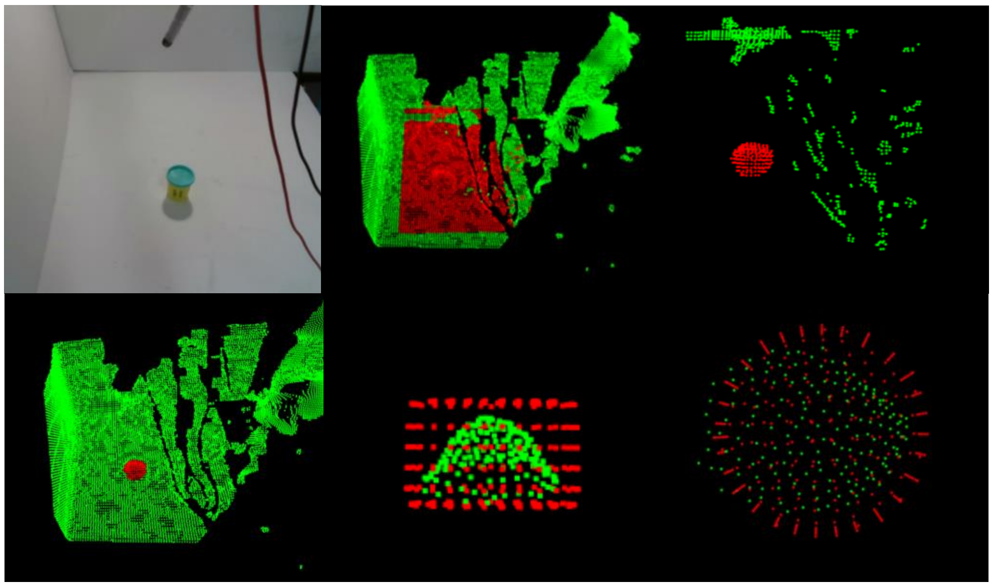


Figure 59: Process of identifying power configuration.

Table 9: Eigenvalues of template 0 compared to the third object's eigenvalues.

	Template 0 (Cylinder) Eigenvalues				Object Eigenvalues
	Unscaled	**Scaled**	**Absolute Error**	**Percent Error**	
PC1	3.80E-01	1.22E-03	0.00E+00	0.00%	1.22E-03
PC2	3.62E-01	1.17E-03	4.75E-04	68.76%	6.91E-04
PC3	1.70E-01	5.47E-04	3.87E-04	240.87%	1.61E-04

5.5 System Architecture

The final implementation of the gripper's system architecture leveraged ROS Melodic to orchestrate the integration between components. This system was written in C++ and depended on the use of the Intel RealSense SDK, realsense-ros, rosserial, and the Point Cloud Library versions available in Ubuntu 18.04. Similar to the design discussed in Section 4.6, Figure 60 shows the final system design that utilizes the *publisher* and *subscriber* relationship to pass information between nodes. The camera node is launched using the *launch* files provided by *realsense-ros* and publishes to more than the listed topic. This particular topic provides a point cloud of the current scene and can be viewed through *rviz*. The *rosserial* package is used to communicate to the gripper node as a serial node as it runs on an ESP32. To run the system, it is necessary to upload the motor control program previously discussed in Section 5.3 via Visual Studio Code's PlatformIO extension. Once uploaded, a launch file is used to begin all other nodes in the system. If the serial node is not running, the system will fail to publish information to it.

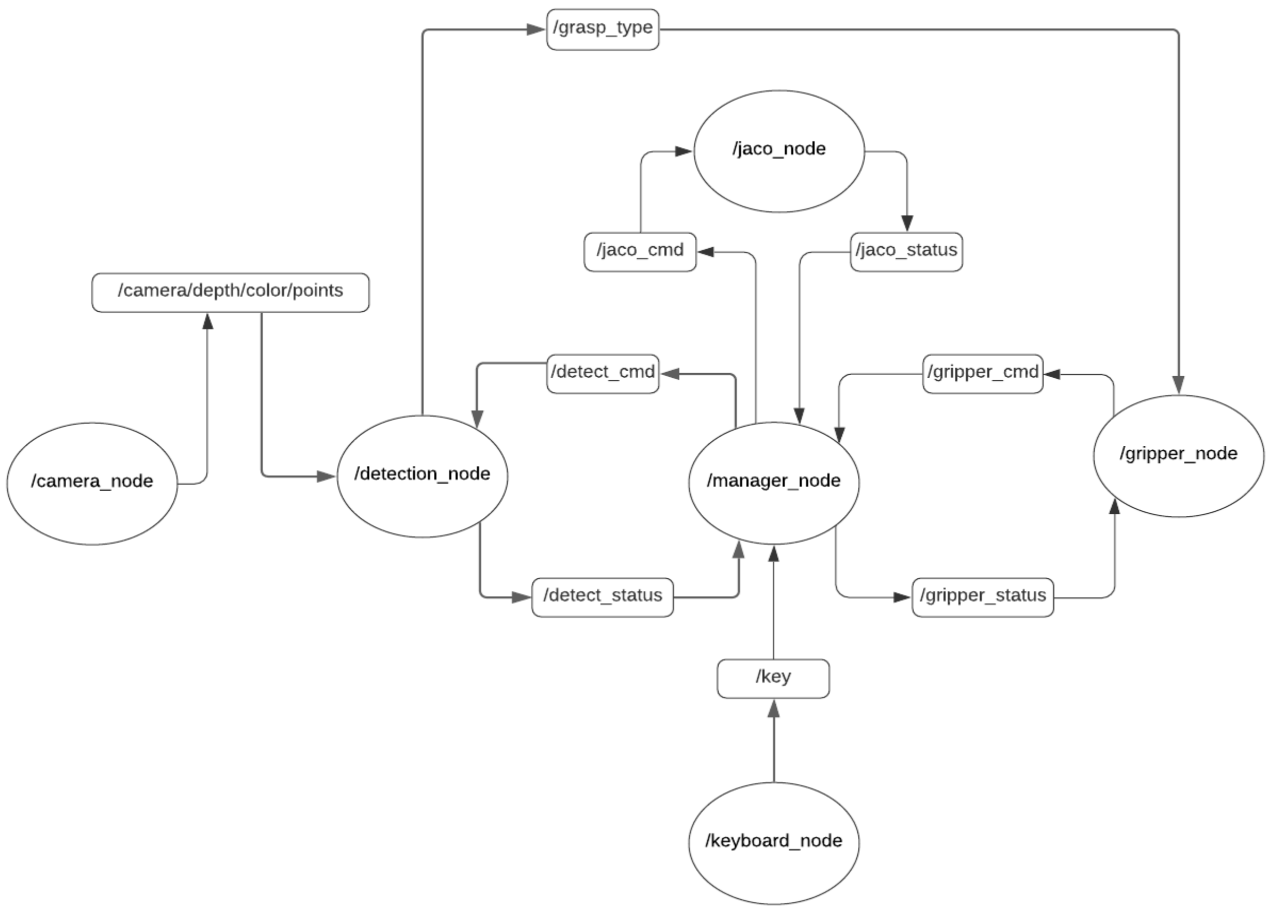


Figure 60: Final ROS software architecture design.

Figure 61 includes a class diagram of the ROS nodes listed above with the exception of the *gripper* node. For this class diagram, refer to Section 4.6. It is important to note that only public member variables and functions are listed within this diagram. The *keyboard*, *jaco*, and *manager* nodes have minimal functions as their purpose is to provide callbacks for the topics they are subscribed to. It is important to note that although the *jaco* node is included in this diagram for descriptive purposes, during the experiments the Jaco arm was controlled separately from the system. The processing node is held within an overarching *object detection* package as it contains an instance of an object detection specific *manager* class. This class coordinates the interaction between the other classes necessary to perform object detection. The logic discussed in Section 4.5.3 was kept and further expanded upon. The most significant change was that the *TemplateMatcher* class uses PCA to make an appropriate match.

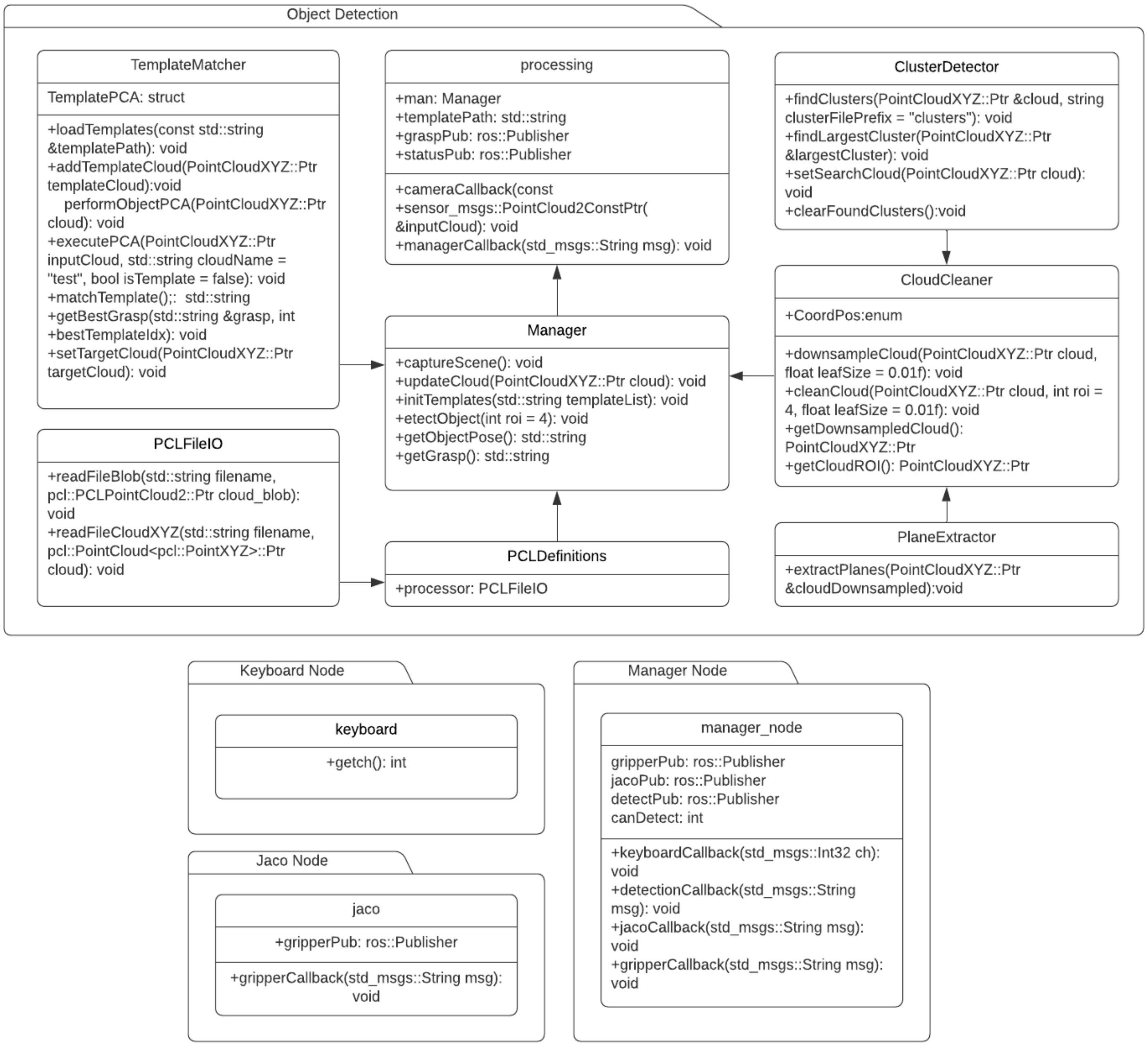

Object Detection
TemplateMatcher
TemplatePCA: struct
+loadTemplates(const std::string &templatePath): void
+addTemplateCloud(PointCloudXYZ::Ptr templateCloud):void
performObjectPCA(PointCloudXYZ::Ptr cloud): void
+executePCA(PointCloudXYZ::Ptr inputCloud, std::string cloudName = "test", bool isTemplate = false): void
+matchTemplate();: std::string
+getBestGrasp(std::string &grasp, int +bestTemplateIdx): void
+setTargetCloud(PointCloudXYZ::Ptr targetCloud): void
processing
+man: Manager
+templatePath: std::string
+graspPub: ros::Publisher
+statusPub: ros::Publisher
+cameraCallback(const +sensor_msgs::PointCloud2ConstPtr(&inputCloud): void
+managerCallback(std_msgs::String msg): void
ClusterDetector
+findClusters(PointCloudXYZ::Ptr &cloud, string clusterFilePrefix = "clusters"): void
+findLargestCluster(PointCloudXYZ::Ptr &largestCluster): void
+setSearchCloud(PointCloudXYZ::Ptr cloud): void
+clearFoundClusters():void
CloudCleaner
+CoordPos:enum
+downsampleCloud(PointCloudXYZ::Ptr cloud, float leafSize = 0.01f): void
+cleanCloud(PointCloudXYZ::Ptr cloud, int roi = 4, float leafSize = 0.01f): void
+getDownsampledCloud(): PointCloudXYZ::Ptr
+getCloudROI(): PointCloudXYZ::Ptr
Manager
+captureScene(): void
+updateCloud(PointCloudXYZ::Ptr cloud): void
+initTemplates(std::string templateList): void
+etectObject(int roi = 4): void
+getObjectPose(): std::string
+getGrasp(): std::string
PCLFileIO
+readFileBlob(std::string filename, pcl::PCLPointCloud2::Ptr cloud_blob): void
+readFileCloudXYZ(std::string filename, pcl::PointCloud<pcl::PointXYZ>::Ptr cloud): void
PCLDefinitions
+processor: PCLFileIO
PlaneExtractor
+extractPlanes(PointCloudXYZ::Ptr &cloudDownsampled):void
Keyboard Node
keyboard
+getch(): int
Jaco Node
jaco
+gripperPub: ros::Publisher
+gripperCallback(std_msgs::String msg): void
Manager Node
manager_node
gripperPub: ros::Publisher
jacoPub: ros::Publisher
detectPub: ros::Publisher
canDetect: int
+keyboardCallback(std_msgs::Int32 ch): void
+detectionCallback(std_msgs::String msg): void
+jacoCallback(std_msgs::String msg): void
+gripperCallback(std_msgs::String msg): void

Figure 61: Class diagram of the ROS nodes used.

6. Discussion

At the conclusion of this MQP, the team accomplished the design and production of a flexible robotic origami gripper system. As mentioned in the previous chapter, there were 30 total task specifications that the team had determined at the beginning of this project. By the end, we were able to accomplish all but three. The three task specifications we were unable to complete include: grasping an object that is 8 cm in diameter or width (Task Specification #4), implementing at least one force sensor per gripper finger (Task Specification #10), and determining the location of the object with the vision system (Task Specification #23).

6.1 Gripper Design

The gripper can grasp objects up to 6.2 cm. However, once the object size exceeds this diameter, the gripper is unable to perform stable grasps. This issue is partly influenced by the fact that the fingers are placed too close together to allow for such a large grasp. With the fingers attached only 2.5 cm from the center of the base, this design created a compact gripper that could comfortably grasp objects smaller than 6.2 cm. Furthermore, the design and behavior of the fingers contributed to the inability to grasp large objects. Both the triangular beam and Yoshimura fingers are designed to approximately contract to the shape of a small arc, which prevents the finger from wrapping around such a wide object to secure a grasp. In addition, the fingertips have a relatively low coefficient of friction thus not providing enough traction to provide a stable grasp for large and heavy objects.

6.2 Control System

We were unable to add a force sensor to each of the four fingers as there was a lack of available force sensors in the WPI Soft Robotics Lab. As a result, rather than using four force sensors as previously planned, only one force sensor was used in the entire system. The team placed the sensor on the thumb as it is the only finger used in all grasp types. This ensured that each grasp had the potential to be force controlled to an extent. However, even with the restriction of the single force sensor, the force control was unable to be fully implemented. The code for the force control component of the switching control is within the program, but the force sensor was unable to produce understandable readings. The team believes a wiring connection problem propagated from either the original soldering or the sensor PCB could have caused the aforementioned issue.

With regards to the switching control, the code was written to have the motors controlled initially by position control until there was a threshold force of 100 mT detected at the fingertip. Once this force was detected, the controller would switch to force control ensuring the motors still rotated until the force sensor detected a maximum force of 200 mT. As mentioned, the force sensors were unable to properly detect forces, thus resulting in a system controlled purely by position control.

There was significant difficulty creating the controller throughout the course of this project. Early in the project, it was decided that creating a controller for the Yoshimura module would be too difficult to accomplish. Developing a dynamic model for the Yoshimura module would impose the challenge of understanding the nonlinear behavior of the continuum manipulator. With infinite degrees of freedom, it would be extremely difficult to determine its movement trajectory especially if outside forces were to be introduced. Therefore, the team focused on the dynamic model of the triangular beam finger. These fingers closely resembled a three-link serial manipulator, which made the designing of the controller significantly less complex. Although the triangular beam fingers were made of a flexible material and were much more compliant than typical rigid linkages, those properties of the finger were neglected to simplify the system.

Originally, the team attempted to design the controller starting with the dynamic model. Throughout the duration of the controller design, it was unclear whether the system was solvable as there were more unknowns than known values in a three-joint system. As mentioned in the results, this method ended in plots of the joint trajectories diverging. Most likely this issue may have arisen from errors in the derivations of the potential and kinetic energies, which would subsequently lead to the creation of an incorrect dynamic model.

The implementation of the dynamic model designed by Jinho Kim et al. [33] helped to achieve a significantly more understandable and realistic response curve. Since this was a symbolic dynamic model, the team was able to use our own parameters that reflected the measurements of the triangular beam finger. Still using the same foundation as the previous controller, ordinary differential equations were used to increment the time. The output generated three joint angles all appearing to converge at the given setpoint.

6.3 Object Detection System

Of the task specifications related to the vision system, all but one of them were achieved. Of particular interest was the time spent trying to detect an object. Prior to the final method of detection, the system was quite slow in trying to match a template. However, on switching to PCA, the match was significantly faster. Task Specification #23 concerned determining the pose of an object within the Jaco arm’s task space and was not completed in this iteration of the project. Although the system was able to identify the corresponding grasps for three test objects, the identification was still dependent on the test conditions. The lighting and positioning of the object could affect the accuracy of the detection system. Because only one camera was leveraged, the team only had partial point cloud data of the target object. Although this was better than viewing the object from top-down, the data was not enough to eliminate the problem introduced by poor lighting conditions. In such a setting, the shadow of the object was considered a part of the point cloud data and subsequently, the system tried to accommodate this. At times, this led to an object’s point cloud resembling that of another. This was most prevalent in the detection of a power and tripod grasp. The method of determining the correct template relies on finding the minimum average absolute error of the second and third principal components. Therefore, if the point clouds of two objects yielded similar principal components, it could misidentify the expected grasp. In Section 5.4, the choice between a tripod and power grasp was close for the second and third objects.

The test objects were chosen for their ability to both be grasped by the gripper and detected by the system. Solid, bright colors allowed the objects to be distinct against the white backdrop of the task space. Smaller objects than the pink test object discussed in Section 5.4 were not chosen as these objects slipped between the gripper’s fingertips and the vision system could not accurately and reliably segment these objects from the background. Tests showed that even when the point cloud of a small object was found, it was ignored in favor of the largest cluster that was not removed from the scene by the segmentation algorithm. The team determined that this method of detection proved more useful for these larger test objects. Another contributing factor to the selection of the grasp was the templates used to identify the grasp. The shape and dimensions of these 3D templates influenced the principal components determined by PCA.

6.4 System Architecture

The final system architecture simplified the integration between the gripper's software components. By utilizing ROS, the components easily transferred information and allowed for the system to be launched with a few commands. Appendix H outlines the steps the team followed to create a virtual machine for this system. As mentioned in Section 5.5, the packages and libraries of note included Intel RealSense SDK, *realsense-ros*, *rosserial*, and the *Point Cloud Library*. Each of these provided necessary functionality to the system although there were several challenges that arose over the course of the project. A problem that arose with the use of *rosserial* was that during testing, the *gripper node* would time out and unsync with the *master node*. When this occurred, the system had to be restarted to resync the board. We determined that this issue arose when the motor control code spent too long reaching the desired grasp position. Other users of *rosserial* expressed this could occur when the time between spinning the node was too long. The *realsense-ros* package provided functions helpful for debugging and testing the program. By recording the test scene and saving to a bag file, the camera node code can then be launched using the recorded stream. By using the launch file arguments, the team could launch using either the live camera stream or a from a file.

When the program runs, the camera publishes every frame to the */camera/depth/color/points topic* and keeps track of the current frame within the *detection node*. By leveraging the *keyboard node* to read user input, the *manager node* then directs the *detection node* to take the most recent saved frame and process it. All nodes report back to the *manager* node if they have completed their task and only until they have reported back does the *manager* command the next node in the process to continue running the program. An advantage to this modularized approach is that each of the nodes are independent of one another and may be switched out with other nodes as long as the topics are still published as expected. For example, the *camera* node could be switched out with another node that publishes similar data. Another component of our system included integrating the Jaco arm. For the purposes of our tests, we independently maneuvered the arm to the desired locations. Due to time constraints, the foundation for the integration of the *jaco* node was added to the system architecture, but the implementation is incomplete. To run the Jaco arm without facing issues caused by the *kinova-ros* package, the team recommends using Ubuntu 18.04.

Before the team switched development to a virtual machine, both the object detection and motor control programs were running on a Windows environment. To begin testing integration methods in this environment the team had initially experimented with Google's Firebase. However, it was quickly decided to switch to ROS to consolidate the system. Although the advantages of ROS heavily outweighed the disadvantages, an issue present in the system is that the topics in the system are constantly publishing and receiving information from nodes. This has the potential of affecting the performance of the system if important published information is dropped before it is received. Due to time constraints, the team did not modify the system to use ROS services which would remedy this issue.

6.5 Impacts of COVID-19

All in all, the team is satisfied with the hard work of the members as well as collaborators who helped make this project a success. The COVID-19 pandemic contributed to various difficulties that affected the planning and outcome of this project. Due to COVID-19 restrictions, the team had limited access to the lab because of the space capacity of the lab and campus closures throughout the year. These limitations resulted in slower development and prototyping of fingers as we could not laser cut, solder, or access a power supply. Additionally, our team could not meet in person at the beginning of the year, which made integration later in the project more challenging. Furthermore, it was difficult to gain access to previously developed materials such as the force sensor. Despite the challenges of COVID-19, the help of collaborators ensured the team was able to address the above obstacles.

7. Conclusions and Recommendations

The FROG MQP was designed on the premise that origami soft robots would prove advantageous in gripper applications for their light weight, low cost, and modularity. By utilizing triangular beam and Yoshimura origami fingers, a position control system, and an object detection system, the resulting gripper was able to detect objects, identify the correct grasp pattern, and execute the grasp.

The work accomplished in this MQP leads the team to believe that there is much room for exploration into the use of origami modules in robots. At the beginning of this MQP, the team set out to design and build a flexible robotic origami gripper that was able to recognize and grasp various objects. Throughout our project, the team analyzed the benefits of using foldable soft materials to create strong origami structures. This was evident as the gripper prototypes with either the triangular beam or Yoshimura fingers weighed less than 333 g and were able to withstand payloads of over triple its weight. Furthermore, the project investigated the advantages that hybrid controllers offer. Our origami fingers are underactuated, which makes their movement highly unpredictable. The introduction of a position controller allowed the team to control the grasps utilizing the finger's position. Preliminary research was also conducted to explore the feasibility of using a force sensor as part of a hybrid controller scheme. Moreover, this MQP examined the use of computer vision to supplement the gripper's abilities. An Intel RealSense depth camera was used to collect point cloud data to process with image segmentation algorithms. This application of computer vision allowed the gripper to respond to its environment and grasp various types of objects by identifying a corresponding grasp type based on the detected shape of the object using a computationally efficient principal component analysis (PCA). At the conclusion of the project, the team reaffirmed that there is potential to utilize the benefits of origami robots in other applications.

Based upon the results of our work, we identified several recommendations that could improve this project in the future. There is potential to further experiment with different joint stiffnesses along the triangular beam finger. This would allow for a better closing configuration of the triangular beam finger for grasping objects. Additionally, we recommend an exploration of Yoshimura and triangular beam finger combinations to better understand if certain origami fingers are more suitable for specific finger positions. Another aspect to explore is the use of a second set of tendons to control the extension of the fingers since it was not integrated into the final design of the gripper. Furthermore, there is potential to improve upon the control of the gripper by fully incorporating a hybrid position and force controller. This addition would ensure proper feedback from the system both from the fingers' position as well as the force detected at the fingertips. In terms of the vision system, the detection of small and irregular objects could be improved as the system can incorrectly segment them as part of a plane. There is the potential to modify the use of PCA and introduce an object detection method capable of distinguishing between a wider array of objects. Through the implementation of these recommendations, the Flexible Robotic Origami Gripper could be developed into a more robust system with more adaptive grasps.

References

[1] C. D. Onal, R. J. Wood, and D. Rus, “An Origami-Inspired Approach to Worm Robots,” *IEEE/ASME Trans. Mechatron.*, vol. 18, no. 2, pp. 430–438, Apr. 2013, doi: 10.1109/tmech.2012.2210239.

[2] A. M. Votta, S. Y. Gunay, D. Erdogmus, and C. Onal, “Force-Sensitive Prosthetic Hand with 3-axis Magnetic Force Sensors,” presented at the *2019 IEEE International Conference on Cyborg and Bionic Systems (CBS)*, Sep. 2019, doi: 10.1109/cbs46900.2019.9114463.

[3] S. G. Faal, F. Chen, W. Tao, M. Agheli, S. Tasdighikalat, and C. D. Onal, “Hierarchical Kinematic Design of Foldable Hexapedal Locomotion Platforms,” *Journal of Mechanisms and Robotics*, vol. 8, no. 1, Aug. 2015, doi: 10.1115/1.4030468.

[4] J. Santoso, E. H. Skorina, M. Luo, R. Yan, and C. D. Onal, “Design and analysis of an origami continuum manipulation module with torsional strength,” presented at the *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, doi: 10.1109/iros.2017.8206027.

[5] “History of Robots and Robotics,” *Thomas*, Oct. 2020. Accessed on: Oct. 6, 2020. [Online]. Available: https://www.thomasnet.com/articles/automation-electronics/history-of-robotics/.

[6] D. Rus and M. T. Tolley, “Design, fabrication and control of origami robots,” *Nat Rev Mater*, vol. 3, no. 6, pp. 101–112, May 2018, doi: 10.1038/s41578-018-0009-8.

[7] A. M. Mehta and D. Rus, “An end-to-end system for designing mechanical structures for print-and-fold robots,” presented at the *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, doi: 10.1109/icra.2014.6907044.

[8] K. Miura and T. Tachi, “Synthesis of Rigid-Foldable Cylindrical Polyhedra”, *Journal of ISIS-Symmetry*, pp. 204-213, Aug. 2010.

[9] Ł. Jaworski and R. Karpiński, “Biomechanics of the Human Hand,” *JTEME*, vol. 3, no. 1, pp. 28–33, Jun. 2017, doi: 10.35784/jteme.536.

[10] M. Vergara, J. L. Sancho-Bru, V. Gracia-Ibáñez, and A. Pérez-González, “An introductory study of common grasps used by adults during performance of activities of daily living,” *Journal of Hand Therapy*, vol. 27, no. 3, pp. 225–234, Jul. 2014, doi: 10.1016/j.jht.2014.04.002.

[11] V. K. Nanayakkara, G. Cotugno, N. Vitzilaios, D. Venetsanos, T. Nanayakkara, and M. N. Sahinkaya, “The Role of Morphology of the Thumb in Anthropomorphic Grasping: A Review,” *Front. Mech. Eng.*, vol. 3, Jun. 2017, doi: 10.3389/fmech.2017.00005.

[12] R. Mutlu, G. Alici, M. in het Panhuis, and G. M. Spinks, “3D Printed Flexure Hinges for Soft Monolithic Prosthetic Fingers,” *Soft Robotics*, vol. 3, no. 3, pp. 120–133, Sep. 2016, doi: 10.1089/soro.2016.0026.

[13] L. U. Odhner et al., “A compliant, underactuated hand for robust manipulation,” *The International Journal of Robotics Research*, vol. 33, no. 5, pp. 736–752, Feb. 2014, doi: 10.1177/0278364913514466.

[14] G. P. Kontoudis, M. Liarokapis, K. G. Vamvoudakis, and T. Furukawa, “An Adaptive Actuation Mechanism for Anthropomorphic Robot Hands,” *Front. Robot. AI*, vol. 6, Jul. 2019, doi: 10.3389/frobt.2019.00047.

[15] A. M. Votta et al., “Kinematic Optimization of an Underactuated Anthropomorphic Prosthetic Hand,” presented at the *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020.

[16] V. Niola, C. Rossi, and S. Savino, “Influence of the Tendon Design on the Behavior of an Under-Actuated Finger,” in *Advances in Service and Industrial Robotics*, Springer International Publishing, 2017, pp. 1033–1042.

[17] A. Owen-Hill, “Robot Vision vs Computer Vision: What’s the Difference?” *Robotiq*. July 2016. [Online]. Available: https://blog.robotiq.com/robot-vision-vs-computer-vision-whats-the-difference.

[18] “Object Recognition,” *MathWorks*, 2020. [Online]. Available: https://www.mathworks.com/solutions/image-video-processing/object-recognition.html.

[19] J. M. Wormley, K. Lafontant,N. Sujumnong, and R. A. Edwards, “Vision-based Intelligent Prosthetic Robotic Arm,” Major Qualifying Project, Dept. Mechanical Eng. and Electrical and Computer Eng., Worcester Polytechnic Institute, Worcester, MA, USA, 2015. [Online]. Available: https://digitalcommons.wpi.edu/mqp-all/3579.

[20] K. M. Sullivan and M. T. Merlin, “Frankenhand: An Intelligent Prosthetic,” Major Qualifying Project, Dept. Mechanical Eng., Worcester Polytechnic Institute, Worcester, MA, USA, 2016. [Online]. Available: https://digitalcommons.wpi.edu/mqp-all/3084/.

[21] “About,” *OpenCV*, 2020. [Online]. Available: https://opencv.org/about/.

[22] “Deep Neural Networks (dnn module),” *OpenCV*, 2020. [Online]. Available: https://docs.opencv.org/master/d2/d58/tutorial_table_of_content_dnn.html.

[23] A. M. Votta, J. W. Wennersten, R. J. Rivas, and T. C. Chaulk, “Humanoid Stereoscopic Vision System: A Systematic Redesign of the WPI Motor Eyes Mechanism,” Major Qualifying Project, Dept. Mechanical Eng., Worcester Polytechnic Institute, Worcester, MA, USA, 2017. [Online]. Available: https://digitalcommons.wpi.edu/mqp-all/938/.

[24] A. G. Howard et al., “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” 2017. [Online]. Available: arXiv:1704.04861.

[25] W. Liu et al., “SSD: Single Shot MultiBox Detector,” in *Computer Vision – ECCV 2016,* Springer International Publishing, 2016, pp. 21–37.

[26] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” presented at the *2011 IEEE International Conference on Robotics and Automation (ICRA)*, May 2011, doi: 10.1109/icra.2011.5980567.

[27] G. R. O’Dell, S. J. Mayer, and Y. B. Sipka, “Dexter: A Smart Prosthetic Device for Transradial Amputees,” Major Qualifying Project, Dept. Mechanical Eng., Electrical and Computer Eng., and Biomedical Eng., Worcester Polytechnic Institute, Worcester, MA, USA, 2017. [Online]. Available: https://digitalcommons.wpi.edu/mqp-all/720/.

[28] C. J. Buchanan, “Sensitive Calligraphy Robot & Design Review Creation,” Major Qualifying Project, Dept. Computer Science, and Humanities and Arts., Worcester Polytechnic Institute, Worcester, MA, USA, 2014. [Online]. Available: http://digitalcommons.wpi.edu/mqp-all/1944.

[29] Y. Wang, “Impedance Control Without Force Sensors with Application in Homecare Robotics,” Doctor of Philosophy Thesis, Dept. Mechanical Eng, The University of British Columbia, Vancouver, BC, Canada, 2014. [Online]. Available: https://open.library.ubc.ca/media/download/pdf/24/1.0167046/1

[30] Y. Yang, C. Fermuller, Y. Li, and Y. Aloimonos, “Grasp type revisited: A modern perspective on a classical feature for vision,” presented at the *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* Jun. 2015, doi: 10.1109/cvpr.2015.7298637.

[31] "N20 DC Motor with Magnetic Encoder - 6V with 1:298 Gear Ratio," *Adafruit*. [Online]. Available: https://www.adafruit.com/product/4641.
[32] "ESP32 - DevKitC," *Components101*, 19-Oct-2018. [Online]. Available: https://components101.com/microcontrollers/esp32-devkitc.
[33] "DC Motor + Stepper FeatherWing Add-on For All Feather Boards," *Adafruit*. [Online]. Available: https://www.adafruit.com/product/2927.
[34] J. Kim, A. S. Lee, K. Chang, B. Schwarz, S. A. Gadsden, and M. Al-Shabi, "Dynamic Modeling and Motion Control of a Three-Link Robotic Manipulator," Proceedings of *International Conference on Artificial Life and Robotics*, vol. 22, pp. 380–383, Jan. 2017, doi: 10.5954/icarob.2017.gs8-3.

Appendices

Appendix A: SolidWorks to Adobe Illustrator Workflow for Laser Cutting

In SolidWorks:

- Create 2D origami pattern
- Measure (and write down for use in Adobe Illustrator) the total length and width of design
- Saved as DWG file

In AutoCad:

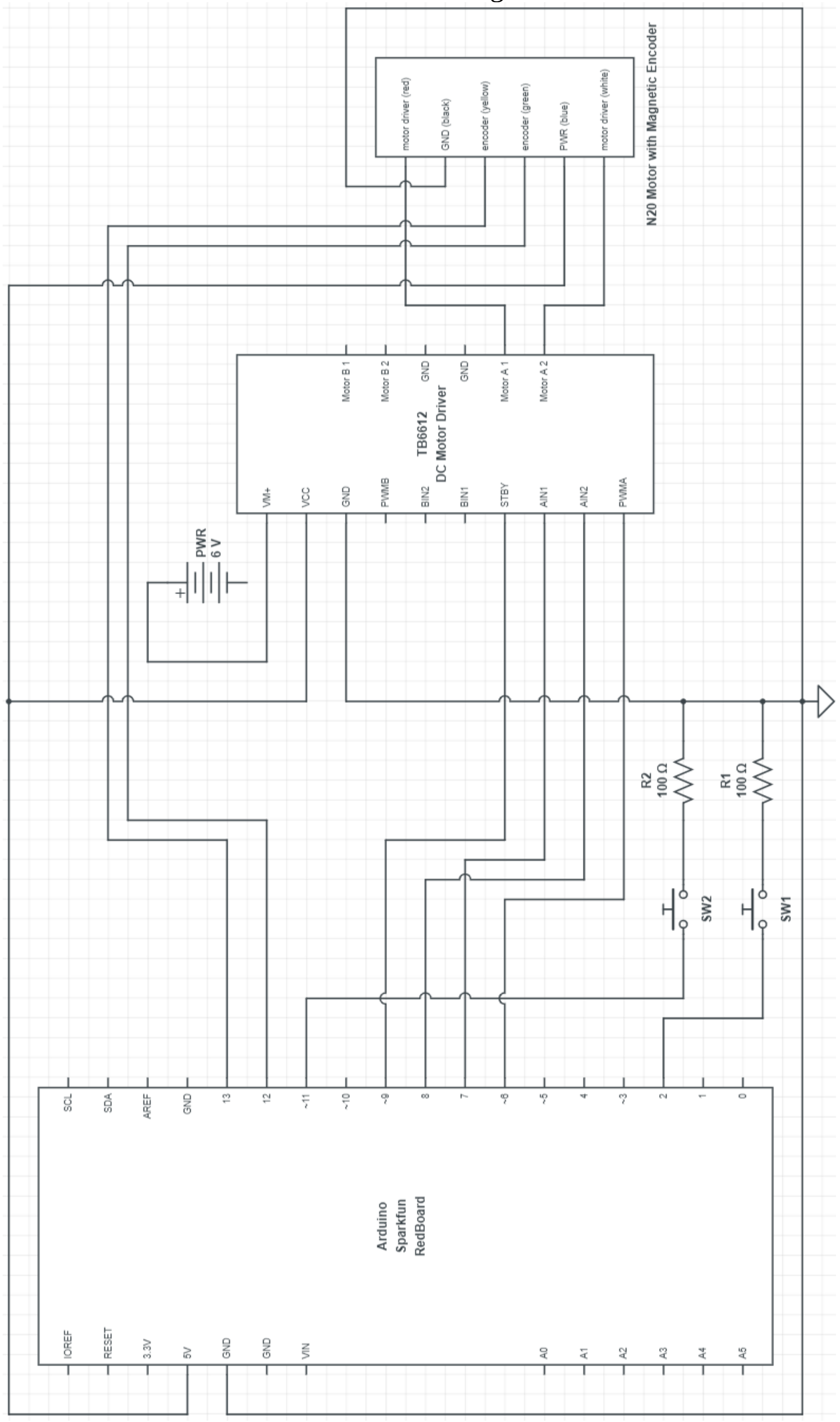
- Opened the DWG file
- Select lines to be perforated and change to HIDDEN line type
- Export as .eps file

In Adobe Illustrator:

- Open .eps file
- Set artboard size to Legal and orientation to Landscape
- Scale origami pattern to correct size using the total length dimension written down from SolidWorks (pattern originally shows up smaller)
- Select all lines and change stroke to 0.001
    - The dashed lines from AutoCad transfer over
- Print to laser cutter

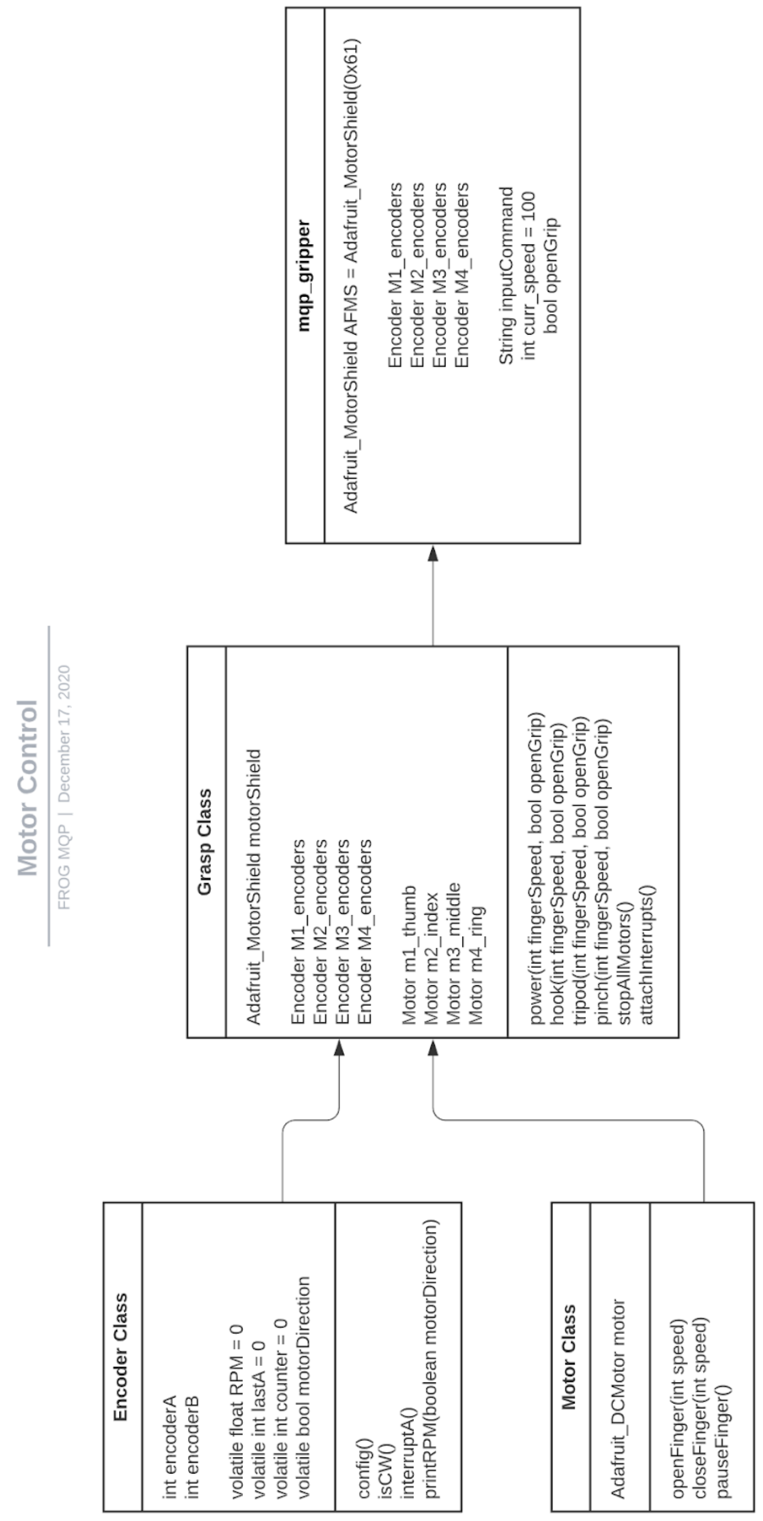
Appendix B: First Iteration Motor Circuit

Schematic of the initial test circuit used to drive the origami modules.

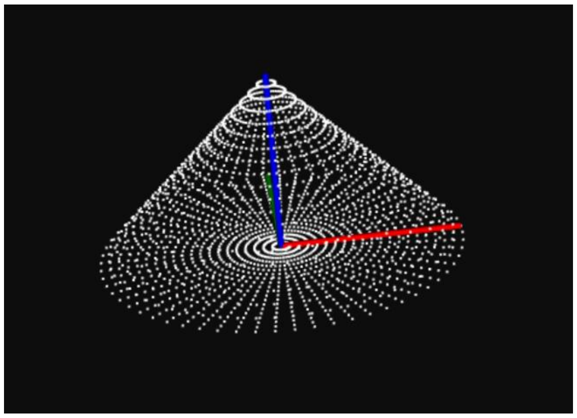


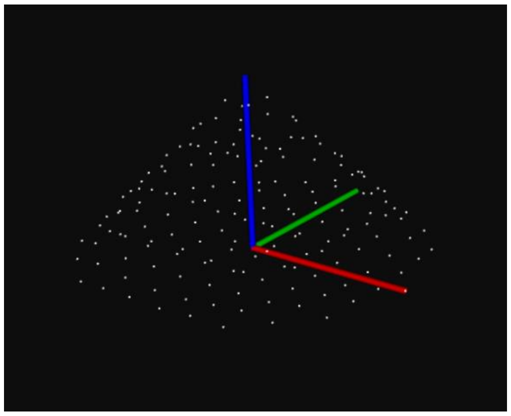
Appendix C: Initial Motor Control Code Structure

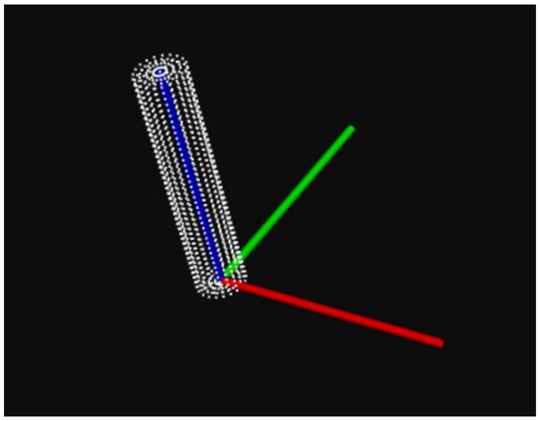
Class diagram of motor control program in beginning stages of the project.

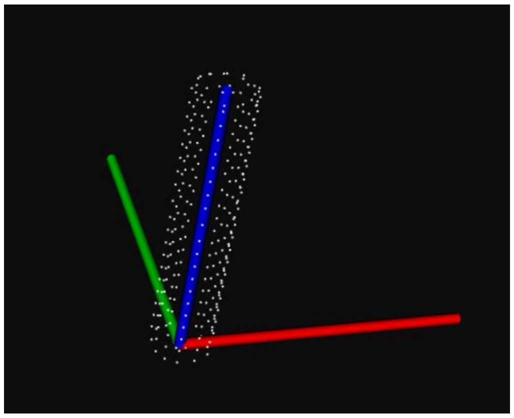


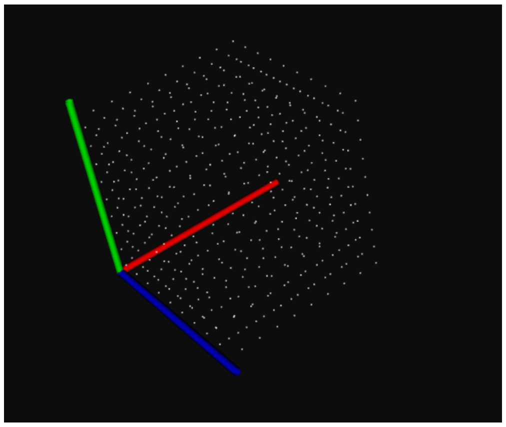
Appendix D: Object Templates

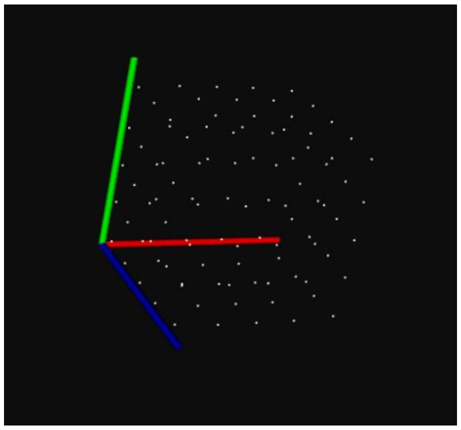

a: *Non-downsampled cone template*

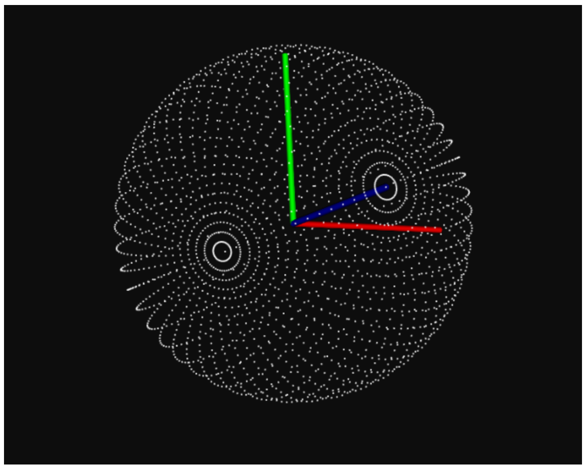

b: *Downsampled cone template*


c: *Non-downsampled thin cylinder template*

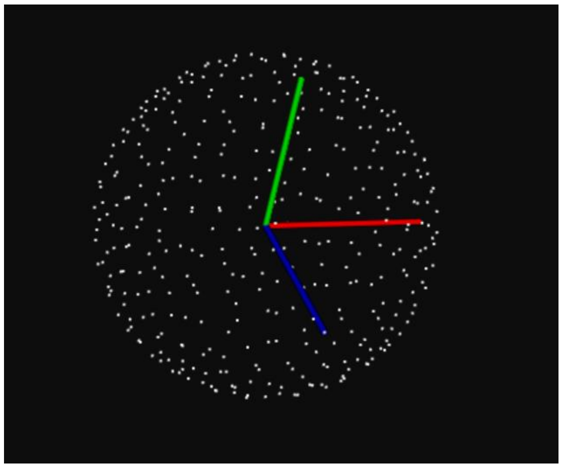

d: *Downsampled thin cylinder template*


e: *Non-downsampled cube template*

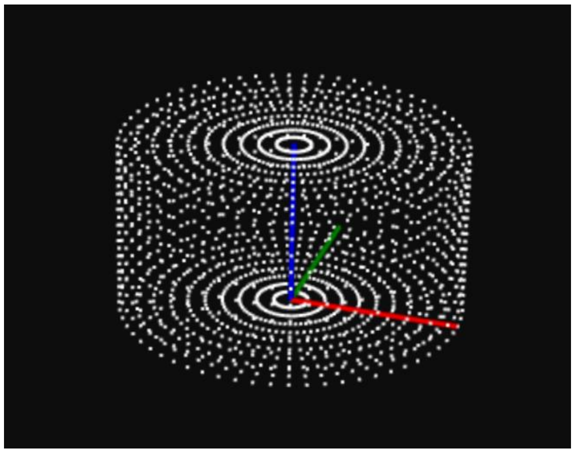

f: *Downsampled cube template*



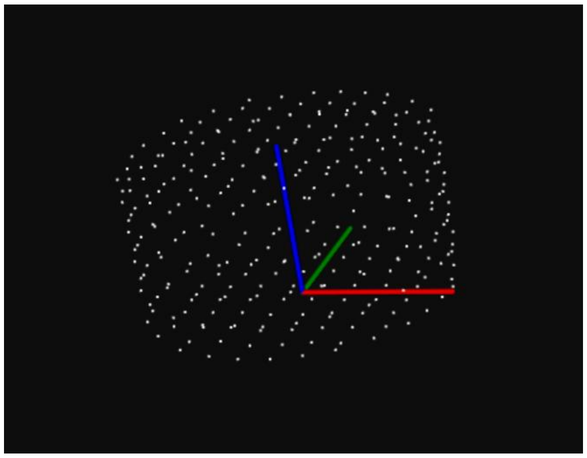
g: *Non-downsampled sphere template*



h: *Downsampled sphere template*



i: *Non-downsampled cylinder template*



j: *Downsampled cylinder template*

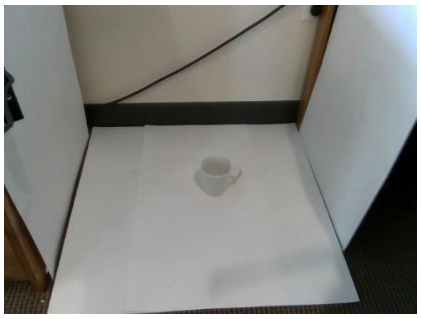
Appendix E: Test Images

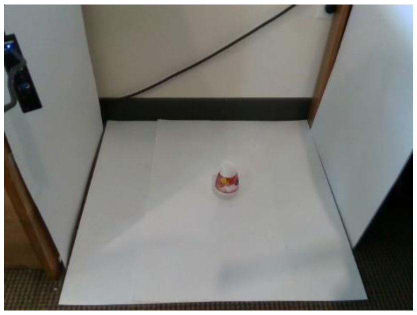
***a:** Test image - cylinder_0*

***b:** Test image - cylinder_1*

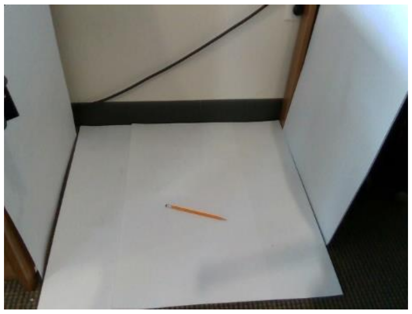
***c:** Test image - cylinder_2*

***d:** Test image - hook_0*


***e:** Test image - hook_1*

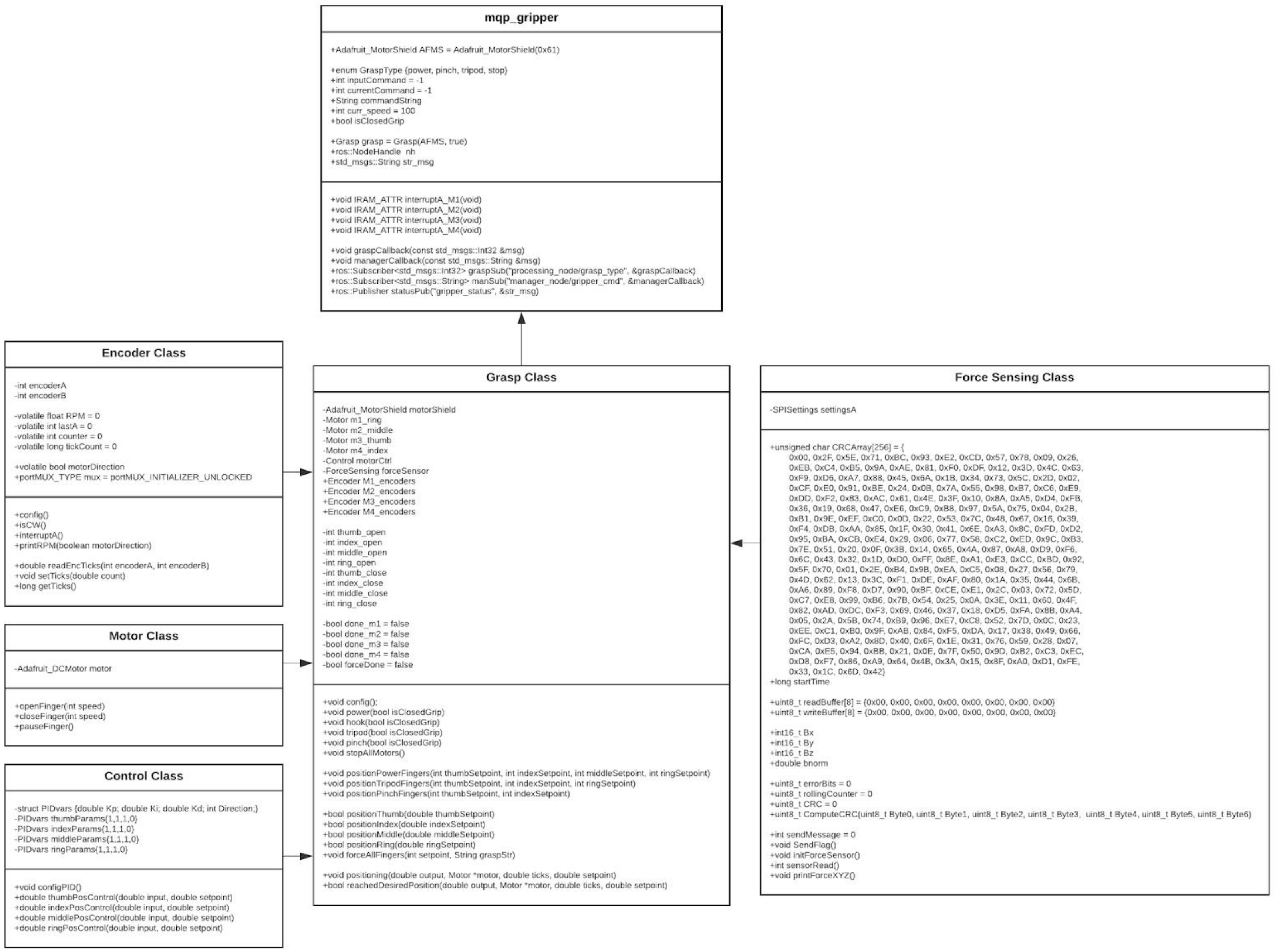

***f:** Test image - tripod_0*

***g:** Test image - pinch_0*


***h:** Test image - pinch_1*

Appendix F: Final Motor Control Code Structure

Final class diagram of motor control program.



mqp_gripper

```
+Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x61)

+enum GraspType {power, pinch, tripod, stop}
+int inputCommand = -1
+int currentCommand = -1
+String commandString
+int curr_speed = 100
+bool isClosedGrip

+Grasp grasp = Grasp(AFMS, true)
+ros::NodeHandle  nh
+std_msgs::String str_msg
```

```
+void IRAM_ATTR interruptA_M1(void)
+void IRAM_ATTR interruptA_M2(void)
+void IRAM_ATTR interruptA_M3(void)
+void IRAM_ATTR interruptA_M4(void)

+void graspCallback(const std_msgs::Int32 &msg)
+void managerCallback(const std_msgs::String &msg)
+ros::Subscriber<std_msgs::Int32> graspSub("processing_node/grasp_type", &graspCallback)
+ros::Subscriber<std_msgs::String> manSub("manager_node/gripper_cmd", &managerCallback)
+ros::Publisher statusPub("gripper_status", &str_msg)
```

Encoder Class

```
-int encoderA
-int encoderB

-volatile float RPM = 0
-volatile int lastA = 0
-volatile int counter = 0
-volatile long tickCount = 0

+volatile bool motorDirection
+portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED
```

```
+config()
+isCW()
+interruptA()
+printRPM(boolean motorDirection)

+double readEncTicks(int encoderA, int encoderB)
+void setTicks(double count)
+long getTicks()
```

Motor Class

```
-Adafruit_DCMotor motor
```

```
+openFinger(int speed)
+closeFinger(int speed)
+pauseFinger()
```

Control Class

```
-struct PIDvars {double Kp; double Ki; double Kd; int Direction;}
-PIDvars thumbParams{1,1,1,0}
-PIDvars indexParams{1,1,1,0}
-PIDvars middleParams{1,1,1,0}
-PIDvars ringParams{1,1,1,0}
```

```
+void configPID()
+double thumbPosControl(double input, double setpoint)
+double indexPosControl(double input, double setpoint)
+double middlePosControl(double input, double setpoint)
+double ringPosControl(double input, double setpoint)
```

Grasp Class

```
-Adafruit_MotorShield motorShield
-Motor m1_ring
-Motor m2_middle
-Motor m3_thumb
-Motor m4_index
-Control motorCtrl
-ForceSensing forceSensor
+Encoder M1_encoders
+Encoder M2_encoders
+Encoder M3_encoders
+Encoder M4_encoders

-int thumb_open
-int index_open
-int middle_open
-int ring_open
-int thumb_close
-int index_close
-int middle_close
-int ring_close

-bool done_m1 = false
-bool done_m2 = false
-bool done_m3 = false
-bool done_m4 = false
-bool forceDone = false
```

```
+void config();
+void power(bool isClosedGrip)
+void hook(bool isClosedGrip)
+void tripod(bool isClosedGrip)
+void pinch(bool isClosedGrip)
+void stopAllMotors()

+void positionPowerFingers(int thumbSetpoint, int indexSetpoint, int middleSetpoint, int ringSetpoint)
+void positionTripodFingers(int thumbSetpoint, int indexSetpoint, int ringSetpoint)
+void positionPinchFingers(int thumbSetpoint, int indexSetpoint)

+bool positionThumb(double thumbSetpoint)
+bool positionIndex(double indexSetpoint)
+bool positionMiddle(double middleSetpoint)
+bool positionRing(double ringSetpoint)
+void forceAllFingers(int setpoint, String graspStr)

+void positioning(double output, Motor *motor, double ticks, double setpoint)
+bool reachedDesiredPosition(double output, Motor *motor, double ticks, double setpoint)
```

Force Sensing Class

```
-SPISettings settingsA
```

```
+unsigned char CRCArray[256] = {
     0x00, 0x2F, 0x5E, 0x71, 0xBC, 0x93, 0xE2, 0xCD, 0x57, 0x78, 0x09, 0x26,
     0xEB, 0xC4, 0xB5, 0x9A, 0xAE, 0x81, 0xF0, 0xDF, 0x12, 0x3D, 0x4C, 0x63,
     0xF9, 0xD6, 0xA7, 0x88, 0x45, 0x6A, 0x1B, 0x34, 0x73, 0x5C, 0x2D, 0x02,
     0xCF, 0xE0, 0x91, 0xBE, 0x24, 0x0B, 0x7A, 0x55, 0x98, 0xB7, 0xC6, 0xE9,
     0xDD, 0xF2, 0x83, 0xAC, 0x61, 0x4E, 0x3F, 0x10, 0x8A, 0xA5, 0xD4, 0xFB,
     0x36, 0x19, 0x68, 0x47, 0xE6, 0xC9, 0xB8, 0x97, 0x5A, 0x75, 0x04, 0x2B,
     0xB1, 0x9E, 0xEF, 0xC0, 0x0D, 0x22, 0x53, 0x7C, 0x48, 0x67, 0x16, 0x39,
     0xF4, 0xDB, 0xAA, 0x85, 0x1F, 0x30, 0x41, 0x6E, 0xA3, 0x8C, 0xFD, 0xD2,
     0x95, 0xBA, 0xCB, 0xE4, 0x29, 0x06, 0x77, 0x58, 0xC2, 0xED, 0x9C, 0xB3,
     0x7E, 0x51, 0x20, 0x0F, 0x3B, 0x14, 0x65, 0x4A, 0x87, 0xA8, 0xD9, 0xF6,
     0x6C, 0x43, 0x32, 0x1D, 0xD0, 0xFF, 0x8E, 0xA1, 0xE3, 0xCC, 0xBD, 0x92,
     0x5F, 0x70, 0x01, 0x2E, 0xB4, 0x9B, 0xEA, 0xC5, 0x08, 0x27, 0x56, 0x79,
     0x4D, 0x62, 0x13, 0x3C, 0xF1, 0xDE, 0xAF, 0x80, 0x1A, 0x35, 0x44, 0x6B,
     0xA6, 0x89, 0xF8, 0xD7, 0x90, 0xBF, 0xCE, 0xE1, 0x2C, 0x03, 0x72, 0x5D,
     0xC7, 0xE8, 0x99, 0xB6, 0x7B, 0x54, 0x25, 0x0A, 0x3E, 0x11, 0x60, 0x4F,
     0x82, 0xAD, 0xDC, 0xF3, 0x69, 0x46, 0x37, 0x18, 0xD5, 0xFA, 0x8B, 0xA4,
     0x05, 0x2A, 0x5B, 0x74, 0xB9, 0x96, 0xE7, 0xC8, 0x52, 0x7D, 0x0C, 0x23,
     0xEE, 0xC1, 0xB0, 0x9F, 0xAB, 0x84, 0xF5, 0xDA, 0x17, 0x38, 0x49, 0x66,
     0xFC, 0xD3, 0xA2, 0x8D, 0x40, 0x6F, 0x1E, 0x31, 0x76, 0x59, 0x28, 0x07,
     0xCA, 0xE5, 0x94, 0xBB, 0x21, 0x0E, 0x7F, 0x50, 0x9D, 0xB2, 0xC3, 0xEC,
     0xD8, 0xF7, 0x86, 0xA9, 0x64, 0x4B, 0x3A, 0x15, 0x8F, 0xA0, 0xD1, 0xFE,
     0x33, 0x1C, 0x6D, 0x42}
+long startTime

+uint8_t readBuffer[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}
+uint8_t writeBuffer[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}

+int16_t Bx
+int16_t By
+int16_t Bz
+double bnorm

+uint8_t errorBits = 0
+uint8_t rollingCounter = 0
+uint8_t CRC = 0
+uint8_t ComputeCRC(uint8_t Byte0, uint8_t Byte1, uint8_t Byte2, uint8_t Byte3,  uint8_t Byte4, uint8_t Byte5, uint8_t Byte6)

+int sendMessage = 0
+void SendFlag()
+void initForceSensor()
+int sensorRead()
+void printForceXYZ()
```

Appendix G: MATLAB Modeling and Simulation code

The code below includes two files. The file MQP_linkTracking.m establishes the controller using the dynamic model. Meanwhile, MQP_simulation.m utilizes the controller to control the input and simulate a controlled output from the fingers.

MQP_linkTracking.m

```
function [ dx ] = MQP_linkTracking( t,x,xf,Kp,Kd,a1,a2 )
%param should include: a1, a2 (trajectory parameters),
%m1,m2,m3,I1,I2,I3,l1,l2,r1,r2,r3 (system parameters);
I1=10; I2 = 10; I3 = 10;
m1=.01; r1=.01;
m2=.01; r2=.01;
m3=.01; r3=.02;
l1=.02; l2=.02;
g=9.8;

%% variables
% joint angles and joint velocities
theta1 = x(1);
theta2 = x(2);
theta3 = x(3);
theta1_dot = x(4);
theta2_dot = x(5);
theta3_dot = x(6);

% simplified sin/cos equations
c1 = cos(theta1);
c2 = cos(theta2);
c3 = cos(theta3);
c12 = cos(theta1 + theta2);
c23 = cos(theta2 + theta3);
c123 = cos(theta1 + theta2 + theta3);
s2 = sin(theta2);
s3 = sin(theta3);
s23 = sin(theta2 + theta3);

%% building matrices
% building mass matrix
a11 = m1*r1^2 + m2*(l1^2+r2^2+2*l1*r2*c2) + m3*(l1^2 + l2^2 + r3^2 + 2*l1*l2*c2 + ...
    2*l1*r3*c23 + 2*l2*r3*c3) + I1 + I2 + I3;
a12 = m2*(r2^2 + l1*r2*c2) + m3*(l2^2 + r3^2 + l1*l2*c2 + l1*r3*c23 + 2*l2*r3*c3) + I2
+ I3;
a13 = m3*(r3^2 + l1*r3*c23 + l2*r3*c3) + I3;
a22 = m2*r2^2 + m3*(l2^2 + r3^2 + 2*l2*r3*c3) + I2 +I3;
a23 = m3*(r3^2 + l2*r3*c3) + I3;
a33 = m3*r3^2 + I3;

% building damping matrix
b1        =        -m2*l1*r2*(2*theta1_dot        +theta2_dot)*s2*theta2_dot        -
m3*(l1*l2*(2*theta1_dot+theta2_dot)*s2 + ...
    l1*r3*(2*theta1_dot+theta2_dot+theta3_dot)*s23                                  +
l2*r3*(2*theta1_dot+2*theta2_dot+theta3_dot)*s3);
b2      =      -m2*(l1*r2*(theta1_dot^2      +      theta1_dot*theta2_dot)*s2      +
l1*r2*theta1_dot*s2*theta2_dot) - ...
    m3*(l1*l2*(theta1_dot^2    +    theta1_dot*theta2_dot)    +l1*r3*(theta1_dot^2    +
theta1_dot*theta2_dot + ...
    theta1_dot*theta3_dot)*s23   +  l2*r3*(theta1_dot^2   +   2*theta1_dot*theta2_dot   +
theta1_dot*theta3_dot + ...
    theta2_dot^2   +   theta2_dot*theta3_dot)*s3   +   l1*l2*theta1_dot*s2*theta2_dot   +
l1*r3*theta1_dot*theta2_dot*s23 + ...
```

```
    l2*r3*(2*theta1_dot + 2*theta2_dot + theta3_dot)*s3);
b3 = -m2*(l1*r3*(theta1_dot^2 + theta1_dot*theta2_dot + theta1_dot*theta3_dot)*s23 +
l2*r3*(theta1_dot^2 + ...
    2*theta1_dot*theta2_dot     +     theta1_dot*theta3_dot     +     theta2_dot^2     +
theta2_dot*theta3_dot)*s3 + ...
    l1*r3*theta1_dot*theta3_dot*s23 + l2*r3*(theta1_dot+theta2_dot)*s3*theta3_dot);

% building gravity matrix
g1 = g*(c1*(m1*r1+m2*l1+m3*l1) + c12*(m2*r2 + m3*l2) + c123*(m3*r3));
g2 = g*((m2*r2 + m3*l2)*c12 + m3*r3*c123);
g3 = g*(m3*r3*c123);

%% creating the matrices

Mmat = [a11 a12 a13
    a12 a22 a23
    a13 a23 a33];
Cmat = [b1 0 0
    b2 0 0
    b3 0 0];
Gmat = [g1
    g2
    g3];

%% calculations
% compute the control input for the system, which
% should provide the torques
K = [Kp, Kd];
e = x(1:6,1) - xf;
tau = -K*e;
dtheta= x(4:6,1);

ddq = Mmat\(tau -Cmat*dtheta - Gmat);

% use the computed torque and state space model to compute
% the increment in state vector.
% compute dx = f(x,u); the rest
% of which depends on the dynamic model of the robot.
dx(1,1) = x(4,1);
dx(2,1) = x(5,1);
dx(3,1) = x(6,1);
dx(4,1) = ddq(1,1);
dx(5,1) = ddq(2,1);
dx(6,1) = ddq(3,1);

end
```

MQP_simulation.m

```
%% clear
clc
clear all;
close all;

%% creating the robot
% link lengths
len0 = 2;
len1 = 2;
len2 = 2;
len3 = 4;

% creating links
link0 = Revolute('a', len0, 'alpha', 0, 'qlim', [0 0]);
link1 = Revolute('a', len1, 'alpha', 0, 'qlim', [0 pi/3]);
link2 = Revolute('a', len2, 'alpha', 0, 'qlim', [0 pi/3]);
link3 = Revolute('a', len3, 'alpha', 0, 'qlim', [0 pi/3]);

% creating robot
robot = SerialLink([link0 link1 link2 link3], 'name', 'tribeam')

% config in form [fixed joint, joint1, joint2, joint3]
config1 = [0 0 0 0];             % plot fully expanded
config2 = [0 pi/3 pi/3 pi/3];    % plot fully contracted
% config3 = [0 pi/8 pi/8 pi/8];    % plot slighyly contracted

%% modeling
% parameters for the controller
x0= [config1(2),config1(3),config1(4),0,0,0];
tf = 5;
xf = [config2(2); config2(3); config2(4); 0; 0; 0];
Kp = diag([300, 700, 300]);
Kd = diag([200, 200, 200]);
a1 = 1;
a2 = 1;

% controller
options = odeset('RelTol',1e-4,'AbsTol',[1e-4, 1e-4, 1e-4, 1e-4,1e-4, 1e-4,]);
[T,X] = ode45(@(t,x) MQP_linkTracking(t,x,xf,Kp,Kd,a1,a2),[0 tf],x0, options);

% recording all the joint values for the contorlled config2
for k = 1:length(X(:,1))
    controlled_config2 = [0, X(k,1), X(k,2), X(k,3)];
end
%% plotting the robot
% configure timing and trajectory
t = [0:.05:1.5]';
qtg = jtraj(config1, controlled_config2, t);

% initializing arrays that will store all the joints
joint1Arr = [];
joint2Arr = [];
joint3Arr = [];

% animated plot config
fh = figure();
fh.WindowState = 'maximized';

% plotting the robot and recording joint angles
tiledlayout(1,2)
nexttile
for q=qtg'
```

```
  robot.plot(q','floorlevel',-1,'zoom', .8, 'noraise', 'tilesize',3 ,...
  'jointcolor','m', 'linkcolor', 'r','base','basecolor','k');
  view(70,60);
  joint1Arr = [joint1Arr; q(2)];
  joint2Arr = [joint2Arr; q(3)];
  joint3Arr = [joint3Arr; q(4)];
end

% initializing an customizing plot
nexttile
plot(T, (X(:,1)*180/pi),'k--');
hold on
plot(T, (X(:,2)*180/pi),'b--');
hold on
plot(T, (X(:,3)*180/pi),'m--');
h = animatedline('Marker','o', 'Color','k');
i = animatedline('Marker','^','Color', 'b');
j = animatedline('Marker','s','Color', 'm');
legend('ideal theta_1','ideal theta_2','ideal theta_3','simulation theta_1','simulation
theta_2','simulation theta_3', 'Location', 'southeast');
title('Control of theta_1, theta_2, and theta_3 for Triangular Beam Fingers');
xlabel('Time (s)');
ylabel('Joint Angles (degrees)');
xlim([0 2]);
index = 1;

% plotting joint angles on a graph
for x = 0:1.5/length(joint1Arr):1.5
    addpoints(h,x,joint1Arr(index)*180/pi);
    addpoints(i,x,joint2Arr(index)*180/pi);
    addpoints(j,x,joint3Arr(index)*180/pi);
    drawnow;
    pause(0.2);
    if index < 31
        index = index + 1;
    end
end
```

Appendix H: Virtual Machine Environment Setup

Steps to set up a virtual machine environment for the system:

1. Use VMWare or VirtualBox to install Ubuntu 18
2. Install ROS Melodic
3. Install Intel RealSense SDK
    a. If the camera is not detected, unplug and plug back in the USB connection.
    b. Installs the realsense viewer
    c. Test install:
        i. Run *realsense-viewer* (without camera connected use a bag file to launch)
4. Install ROS Wrapper
    a. (Installs librealsense library)
    b. Source the devel/setup.bash after installation
    c. Test install:
        i. roslaunch realsense2_camera rs_camera.launch filters:=pointcloud
        ii. *rosrun rviz rviz* (Should see point cloud after modifying world view)
5. Git clone repo of the project's ROS package and install
6. Install rosserial in ROS workspace
    a. sudo apt-get install ros-melodic-rosserial-arduino
    b. sudo apt-get install ros-melodic-rosserial
7. Install VSCode and add PlatformIO extension
8. Clone gripper-control code and import project into PlatformIO
9. Upload code to ESP32
    a. sudo chmod a+rw /dev/ttyUSB0
    b. /dev/ttyUSB0 was automatically detected, but if having issues connecting, add udev rules of PlatformIO
10. Verify that everything is installed by launching command #2 above
11. Install kinova-ros package for ROS Melodic

Commands to run:

1. In launch file change the default argument to desired bagfile:
    a. *<arg name="rosbag_filename" value="/path/to/bagfile/test.bag"/>*
2. roslaunch ros_object_detection frog_mqp.launch cam_type:=file
    a. Make sure that code is uploaded to ESP32 beforehand

Appendix I: Relevant GitHubs

To note, only collaborators can view the code as they are private repositories. Please contact gr-frog-mqp@wpi.edu for further information.

Motor Control

This GitHub link contains the code that pertains to driving the motors attached to the origami finger tendons and the controlling the system with a hybrid position and force sensing controller: https://github.com/smspry/MQP-Gripper.v1.git.

Vision System

This GitHub link contains the code related to the ROS package created for everything excluding the gripper node: https://github.com/mariamedi/ros_object_detection