Soft Web-Based Continuum Robot Grippers

Anthony Carambia
acaramb@g.clemson.edu

Soft Web-Based Continuum Robot Grippers

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

by
Anthony Carambia
May 2022

Accepted by:
Dr. Ian Walker, Committee Chair
Dr. Apoorva Kapadia
Dr. Richard Groff

Abstract

We discuss the potential of soft webs to enhance robotic grasping. Specifically, we explore a novel combination of compliant continuum digits interspersed with a flexible material. The resulting webbed structure offers the potential for new modes of robust and adaptive object grasping. We introduce and describe two webbed grippers featuring alternate modes of actuation: pneumatic muscles and remotely actuated tendons. Experiments with the grippers demonstrate their ability to gently capture small, fragile, and non-cooperative objects.

Acknowledgments

I would first like to thank Dr. Ian Walker for all of his support and guidance throughout this project. He had a lot of faith in me and believed in my success since I joined his lab. With out his extra help to meet deadlines and the vast amount of knowledge he has to offer, I would not have been able to complete this project. I would also like to thank Dr. Richard Groff and Dr. Apoorva Kapadia for taking time out of their busy schedules to serve on my committee.

Next, I would like to thank Dr. Chase Frazelle for all of his help and guidance through the project. I cannot thank him enough for all of the brainstorming, trouble shooting, and many late nights in the lab. Without him I would not have been able to complete this project.

Finally I would like to thank all of my friends and family that supported throughout the project. Their kind words and belief in me was what allowed me to persevere and finish the project strong.

Table of Contents

List of Figures

Chapter 1

Introduction

Robot grasping, and the development of end effectors to enable such grasping, has long been an active topic of research. As early as 1977, a sufficient number and variety of robot hands had been developed to justify a book listing them [14]. Since then, a large variety of dexterous robot hands, along with corresponding theoretical developments in multifingered grasping and manipulation, have emerged [4], [21].

A recent trend has been in the development of "soft" robot hands and grippers [13], which feature physically soft and/or compliant elements. An underlying goal is to expand the variety of tasks and objects which can be grasped [8], [27], along with enabling simpler grasping mechanisms. Numerous innovative grippers with soft or flexible elements have been demonstrated in recent years, e.g. [3], [11], [19].

In this thesis, we consider robot grippers based on continuum robot digits. Continuum robots [29], [30], [31] have smooth, compliant bodies. Usually deployed individually as "tongues, trunks, and tentacles" [5], they can take advantage of bending and compliance throughout their bodies to perform adaptive whole arm grasping [17], [16]. Continuum elements have been combined as fingers in several compliant grippers [15], [10], [2]. Herein, we augment grippers based on continuum "fingers", or digits, with a soft webbed structure to explore novel modes of adaptive grasping.

Webbed structures are found in the natural world, particularly in marine animals, for example in the vampire squid ([12], Fig. 1.1, top left) and some species of octopus ([23], Fig. 1.1, bottom left). Webs are also found in the feet of some birds such as ducks, and in mammals such as the flying lemur ([22], Fig. 1.1, top right). Generally, the function of these natural webs can be

Figure 1.1: Animals with webbed structures. Top left: vampire squid; image credit MBARI. Top right: flying lemur; image credit Norman Lim. Bottom left: octopus; image credit Getty Images. Bottom right: giant amazon otter; image credit Julie Zickfoose, Blogspot.

viewed as “grasping” the surrounding environmental media (e.g. water, air) to aid in locomotion (e.g. swimming, gliding). The first exploration of the use of webbed structures in robotics appears to be in [26], in which the swimming of a robot octopus was enhanced by the incorporation of webs.

In terms of grasping of discrete objects within the environment, animals such as some species of otters ([28], Fig. 1.1, bottom right) have webbed hands which can aid in grasping and manipulation. An initial exploration of the use of webs in soft grippers for robotic grasping of discrete objects was made in [9] and [10]. More recently, a web-based gripper was explored in [6]. However, the base and grasp pattern of the digits in that hand were fixed.

This thesis expands on the above works, introducing compliant webbed hands with variable grasping configurations [7], and demonstrating the ability of these hands to capture non-cooperative and small targets relative to digit size. Novel contributions of the thesis include the first tendon-actuated webbed gripper, the first with independently controllable continuum digits, the first such gripper with variable bases for the fingers (“variable palm”), and the first demonstration of the interception of non-cooperative targets with a webbed gripper.

The thesis is organized as follows. Chapter 2 discusses the underlying motivation and design concept. Chapters 3 and 4 describe, respectively, the development of, and experiments with, webbed grippers based on pneumatically actuated and remote tendon actuated continuum digits. Conclusions and suggestions for future research are provided in Chapter 5.

Chapter 2

Motivation and Design Concept

Inspired by the animals referred to above, and particularly motivated to expand the robustness of robot grasping, herein we explore designs based on multiple actively actuated continuum robot digits, interconnected with a flexible surface material. Fig. 2.1 illustrates the design concept.

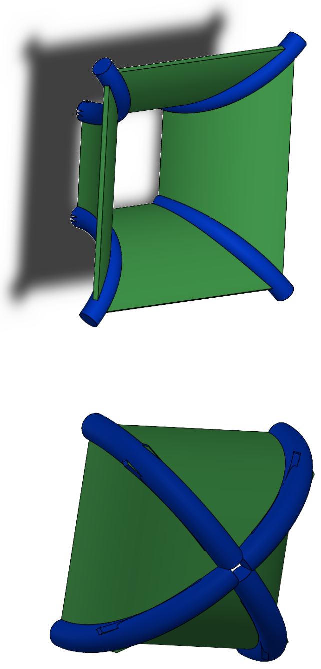

Figure 2.1: Webbed robot design concept. Continuum digits indicted in blue, flexible web in green. Top image: continuum elements in an open configuration; Bottom image: continuum elements shape and close soft web to safely envelop, capture, and stabilize target.

Several issues motivate the incorporation of a flexible web within a "traditional" (blue structure in Fig. 2.1) multi-digit gripper. A key goal is to enhance grasp robustness, allowing the web to compensate for imprecision in the grasp. Such imprecision can arise from control errors in the continuum digits (inherently less precise than traditional robot fingers constructed from rigid links), noisy sensing of positions or motions of objects to be grasped, or in cases where grasping of non-cooperative objects is desired and the object actively tries to avoid the grasp. In these cases, the concept is for the webbing to compensate for such imprecision, preventing objects from "slipping through the gripper fingers", and enhancing the actions of the digits to ensure robust and compliant grasping, providing the gripper envelope surrounds the desired object(s).

The last example above illustrates a second motivation: the exploration of novel grasp modes. The presence of the webbing offers the potential for the gripper to "scoop up" objects, first capturing them in the envelope of the web, enabling subsequent manipulation by the digits. This grasping mode reduces the reliance on high fidelity environmental sensing and fast multi-digit grasping. Object acquisition is no longer constrained by the need for the actuated digits to actively constrain a precisely sensed object, with significant implications for the grasps of objects with non-trivial motion with respect to the gripper. Such novel grasp modes could be exploited in non-traditional applications for robotics, for example the gentle capture and transfer of animals, at a variety of scales. Note that the design concept is highly scalable while also being relatively simple and inexpensive to implement physically (see next sections).

A further goal is to enable the capability of acquiring - though without the ability to dexterously manipulate - very small objects, relative to the gripper work space. One potential long term application is the capture of space debris [20], where end effectors based on the design could "trawl" regions of space, exploiting the flexible but impermeable surface of the webbing to entrap and subsequently transfer debris, even at a small scale.

Note that, while in this work the web material considered is continuous across its two dimensions, i.e. with no "holes," the underlying design does not preclude the use of flexible structures such as traditional nets as interconnecting material. Nets are a key tool in many aspects of human activity, e.g. agriculture (notably fishing [24]). While not considered herein, the use of nets as the webbing material in novel robotic structures of the design concept in Fig. 2.1 could enable a new type of "smart robot net."

In the following chapters, we describe two alternatively actuated hardware realizations of the design concept: actuated using pneumatic muscles and actuated using remotely actuated tendons. Illustrative experiments with the prototypes, and their relative advantages, are reported and discussed.

Chapter 3

Pneumatically Actuated Web

In this chapter, we describe the development of, and experimentation with, the first of two realizations (seen in Fig. 3.1) of the design concept, utilizing pneumatically driven actuators (McKibben muscles) as the digits of the gripper.

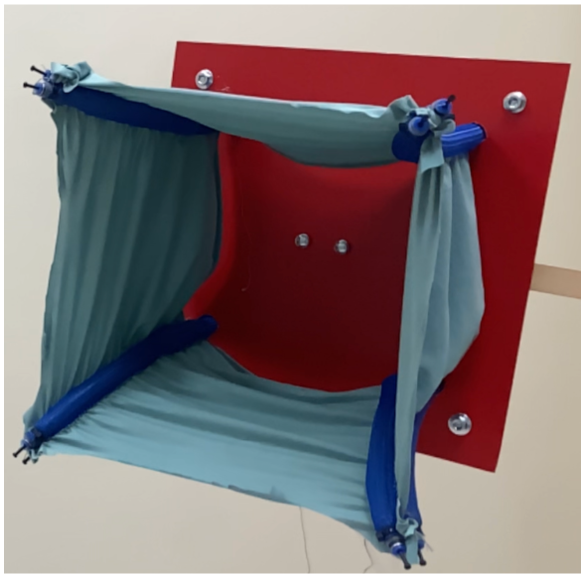

Figure 3.1: Soft Robot Realization

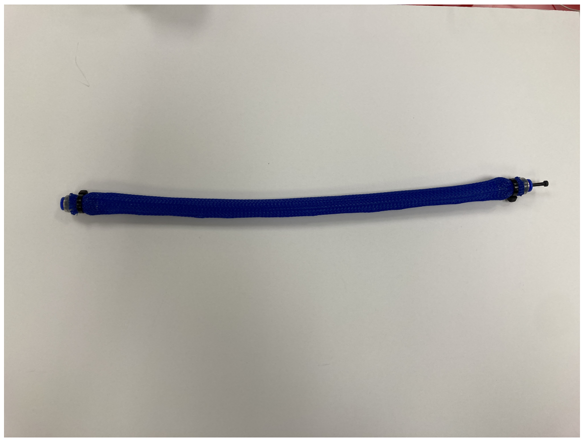

Figure 3.2: Single McKibben Muscle

3.1 Prototype Development

The gripper is composed of eight individual McKibben muscles shown in Fig. 3.2, coupled (sewn) together in pairs. Each McKibben muscle was configured as a contracting muscle in which a supply of pressurized air or fluid causes the muscle's length to decrease in proportion to the pressure applied. The muscles were composed of four components. Fig. 3.3 depicts these components. Each McKibben Muscle has one contracting braided sleeve, one inflation tube, two air hose adapters, one plug, and two zip ties to secure each of the air hose adapters in place. The pairing of two muscles together creates a planar continuum body that bends in the direction of the shortest muscle (i.e. the muscle supplied with the highest pressure). An example sketch of the planar motion for such a pair of McKibben muscles is shown in Fig. 3.4. By alternating actuation for one or the other muscle in a pair at a time, the opening and closing of the gripper could be achieved as the collective muscle pairs bend in and out. The muscle pairs were arranged in a square pattern (shown in Fig. 3.1) with one pair at each corner. The pairs were oriented such that bending planes of each pair intersected with the center of the gripper, resulting in the inner-most muscles closing the gripper when actuated and the outer-most muscles opening the gripper. The unactuated length of each McKibben muscle was 36cm, with the muscled pairs spaced 23cm apart along the perimeter of the square arrangement.

Each pair of muscles was connected to adjacent muscle pairs with a knit fabric web (composed of 96% polyester and 4% spandex) sewn in place between the individual muscles. The fabric was measured to match approximately the 23cm distance between muscle pairs. This was done by taking approximately 92cm of the knit fabric sewing in the pairs 23cm apart. The first pair was

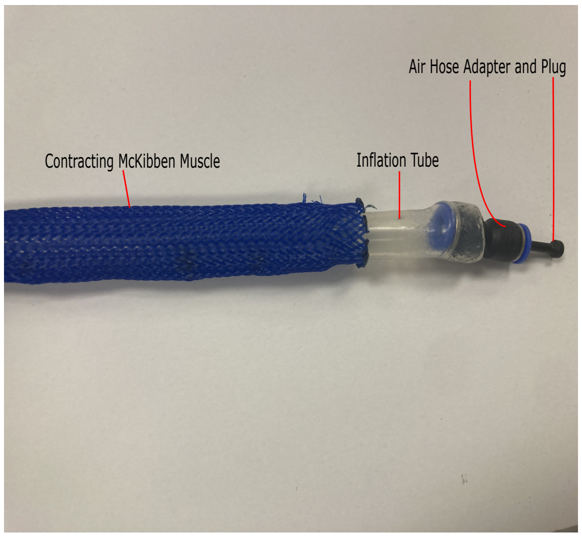

Figure 3.3: McKibben Muscle Components

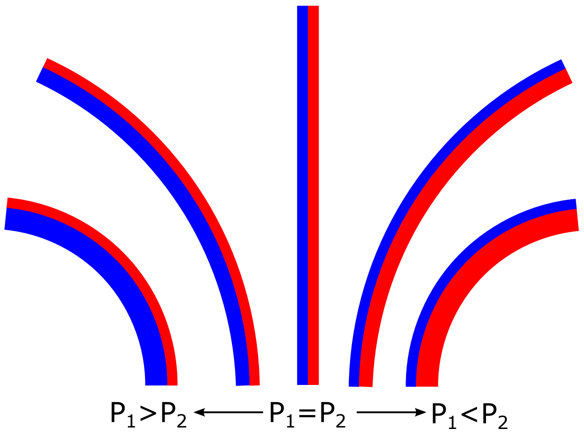

Figure 3.4: Demonstration of planar pair of McKibben muscles. Increasing difference in pressure leads to increased bending towards muscle with higher pressure.

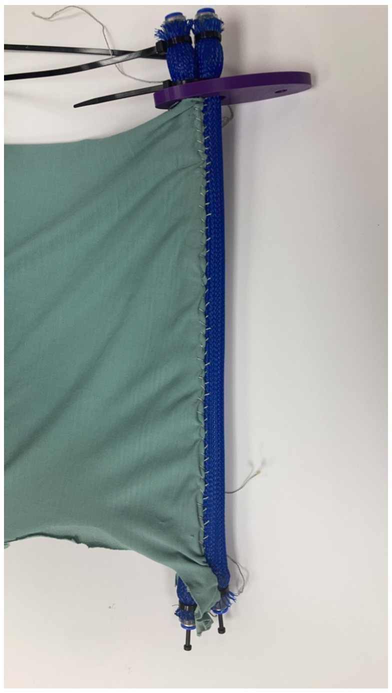

Figure 3.5: Sewing For Each McKibben Muscle Pair

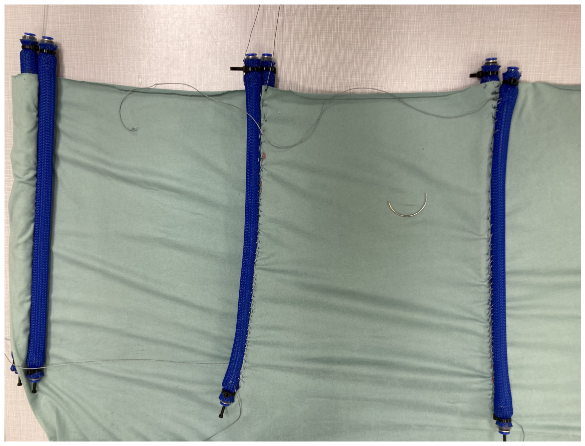

Figure 3.6: Gripper/Web Sewing Process

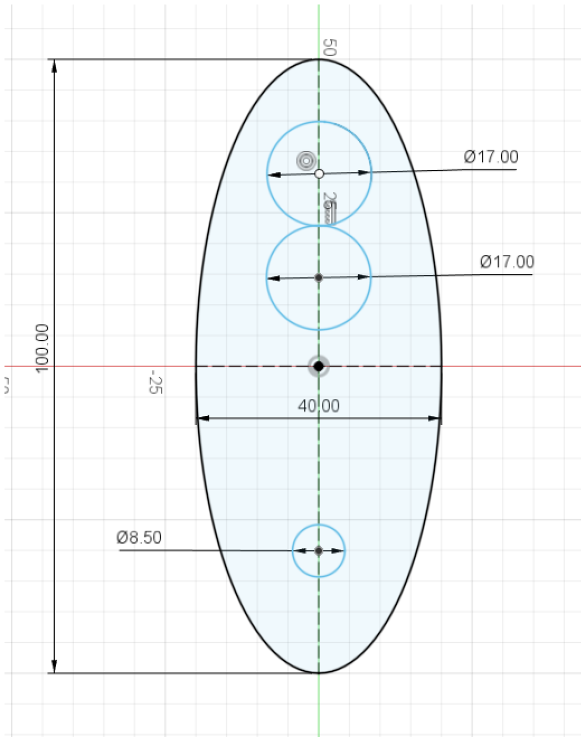

Figure 3.7: McKibben Muscle Mount Print Dimensions

sewn in at the beginning of the 92cm fabric and then every 23cm the next pair was sewn in, until the last 23cm of fabric was sewn into the other side of the first pair at the beginning of the fabric. This final step resulted in a continuous web and created the square shape of the gripper. The stitching done on each side of the McKibben muscle pairs can be seen in Fig. 3.5. The mid-process image in Fig. 3.6 depicts the assembly involved in connecting the muscles and pairs to one continuous length of fabric before being closed together and attached to the “palm” of the gripper. After the gripper digits were sewn together, the digits were mounted to the base frame seen in Fig. 3.1, which consisted of a plane of aluminum composite material that measured 38cm by 38cm. The mounts were 3d printed and the dimensions for the mounts, created in Fusion 360, can be seen in Fig. 3.7. The units for all dimensions were in mm and the mount’s height was printed at 7mm. These were attached to the base frame with M8 nuts and bolt pairs. This base frame serves as a lightweight, relatively rigid palm.

Fig. 3.8 depicts the associated hardware built and used to control the actuation of the gripper in a representative series of experiments. The underlying system for actuating the gripper consisted of two pressure regulators for sourcing the opening and closing of the gripper (SMC

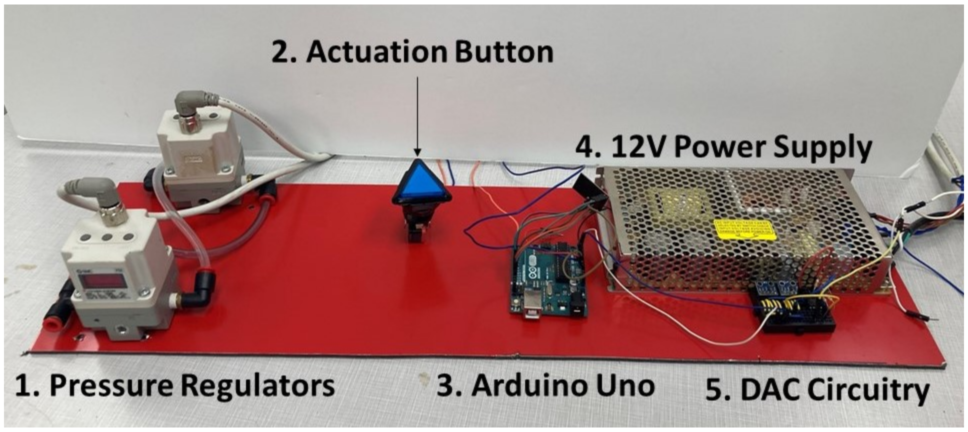

Figure 3.8: Hardware Used to Control Gripper

ITV1050-31N1N4), an Arduino Uno microcontroller to operate the regulators and interpret user input, two Digital-Analog-Converters (DACs) to provide stable, analog signals to the regulators, an air compressor, and relevant DC power supplies for the various components. A simple button interface connected to the microcontroller was used to control the gripper state. There were three states: no pressure to any muscles (partially open, default), pressure applied to the outer-most muscles (fully open), and pressure applied to the inner-most muscles (closed). For both opening and closing, the actuated muscles were supplied with 35PSI.

3.2 Experimental Evaluation

Following the construction process and initial testing of the gripper, a series of experiments in grasping non-cooperative objects were conducted. The key goal for the experiments was to demonstrate the ability of a gripper of the structure discussed above to reliably and gently capture and stabilize non-cooperative targets moving in non-trivial, three-dimensional trajectories. Given a 1-g environment for testing, we elected to grasp two types of flying objects: passively floating (translating and rotating) balloons and actively controlled drones.

It the first test, a series of balloons varying in shape and size were launched towards the gripper. In this test, the base of the gripper was fixed and a variety of spins were imparted to balloons. Spin was introduced by manually tossing the balloons in manners that resulted in the balloons tumbling in unpredictable manners through the air as well as adding small masses to the balloons to alter the center of mass of the balloon. An example of one such toss is seen in Fig. 3.9,

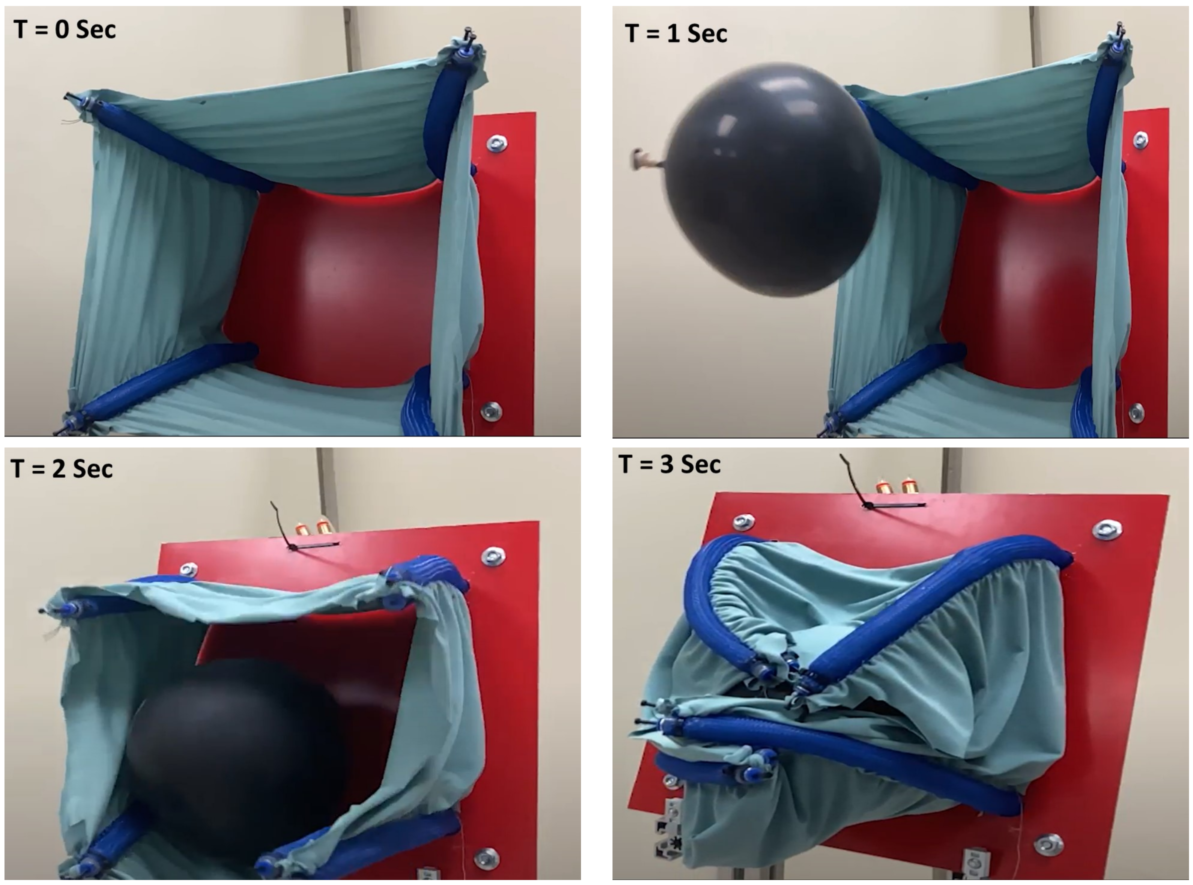

Figure 3.9: Time Lapse for Balloon Grasping

which depicts a time lapse of a balloon capture test over a course of 3 seconds. The spinning balloon, where the spin can be observed by the appearance and disappearance of the balloon stem between t=1s and t=2s, was gently caught in the web and secured by the closing of the gripper. Over the entire series of similar balloon captures, the gripper provided sufficient force to safely restrain all of the balloons without damage.

A second, more challenging, series of tests evaluated whether and how the gripper could successfully capture a flying drone. In these tests, the base of the gripper was actively controlled, and thus there was non-trivial and unmodeled relative motion between the gripper and object (drone) to be captured, with active contributions to this relative motion from both bodies. Fig. 3.10 depicts a representative case, with a time lapse of 13 seconds. In this experiment the drone is flown in an unpredictable and uncooperative manner, while the gripper approaches and captures it successfully. It is important to note the larger time frame of capture here was a result of flying the drone, not

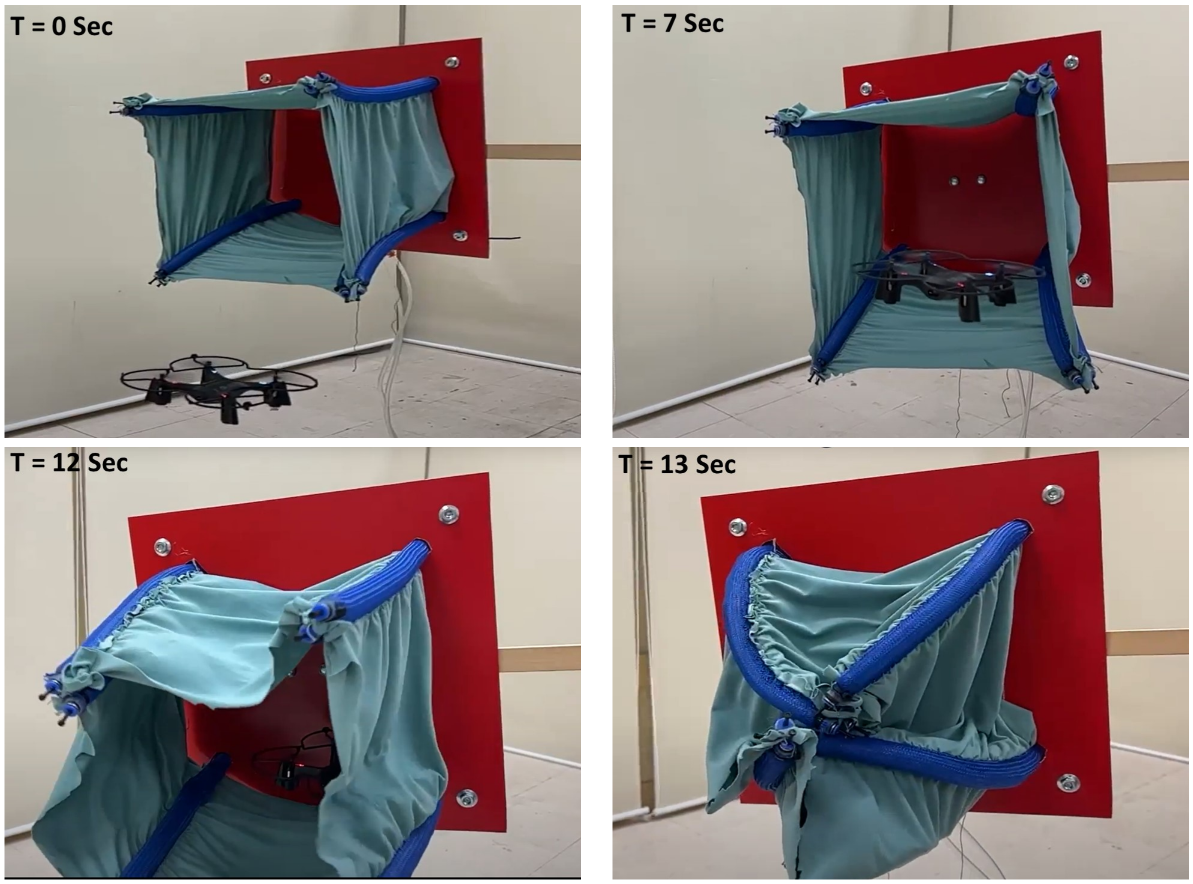

Figure 3.10: Time Lapse for Drone Grasping

the gripper actuation.

The tests with the balloons and the drone were highly successful, with the gripper being able to capture all of the objects without damage to the objects or gripper. The success of the captures, regardless of the exact trajectory of the moving object, highlights the benefit of the compliance of the continuum elements and the web, which provides tolerance for a relatively large margin of error. The lack of damage to gripper or objects also potentially suggests that a web-based gripper can "gently" grasp uncooperative objects, moving in three dimensions, of a variety of shapes, sizes, and trajectories, even though both the underlying hardware and control strategy are very simple.

Chapter 4

Tendon Actuated Web

In the first design realization of the webbed gripper (as reported in Chapter 3), the prototype was actuated via pressurized air. In some applications (for example applications involving Space exploration), it is probable that remote actuation of tendons using electric motors would be a preferred modality. Tendon-based actuation also inherently generates stronger forces than pneumatics. Therefore, we developed a second, more sophisticated, prototype, based on tendon-actuated "tendril" digits.

4.1 Prototype Development

The second, tendon-actuated, webbed gripper was designed with four individual tendril [32] digit subsystems, much like the four actuated pairs of the first design. Each individual tendril subsystem, pictured in Fig. 4.1, was assembled using a carbon fiber backbone (45cm in length), to provide the "continuum" nature of the digit, and a series of tendons that were used to bend the backbone and alter the direction and magnitude of said bend. A single tendon and a jig created for consistent construction can be found in 4.2. The tendrils were constructed with eight spacers and one end cap. The dimensions in mm for these parts can be found in Fig. 4.3. The spacers and end caps had the same dimensions other than the fact that the spacers were printed to have a height of 3mm and the end caps had a height of 10mm. The tendril was constructed by placing the eight spacers and one end cap in the slots of the jig. The backbone material was passed through all of the center holes. Then two smaller carbon fiber tubes were run in parallel through two of the tendon holes to keep the tendon holes aligned. Once all of the components were properly aligned, JB Weld was applied to each side of the spacers and end cap. The first coat of JB Weld was applied to the tendril and allowed to dry for four hours. Then the tendril were flipped over and another coat of JB Weld was applied. The tendrils were left to dry overnight.

The actuator package of each tendril subsystem consisted of two DC motors, in conjunction with two capstans, that were connected to the tendons, along with the appropriate hardware to connect the capstans to the shafts of the DC motors. These are depicted in figure 4.4. Similar to the McKibben muscle pairs of the pneumatic design, one of the DC motors in each tendril subsystem was individually responsible for bending the tendril inward (i.e. closing direction), while the other motor pulled the tendons responsible for bending the tendril outward (opening direction). Note that each tendril was independently controllable. In order to provide increased torsional rigidity to the tendril, each motor was connected to two tendons that ran in parallel along the backbone, for a total of four tendons per backbone. The two tendons on each motor were spooled on the same capstan and at the same rate during motion, providing additional stiffness in the plane perpendicular to the plane of bending. Each individual actuator package measured 22cm$\times$15cm$\times$5cm.

A key innovation in this design was the ability to adjust the relative location of the base don the tendril digits. Given the large footprint of the tendril actuator package, consideration was given for the arrangement of the individual subsystems. To promote reconfigurability and save space, the four tendril subsystems were initially arranged as depicted in Fig. 4.5. The top- and bottom-most actuation packages were aligned so the related tendrils would bend in the same plane, as were the left- and right-most actuation packages. The intersection of the two bending planes occurred at the center of the gripper. In this initial arrangement, the center-to-center distance between neighboring tendrils was 30cm, chosen such that the tendril end-effectors at the maximum angle of bending met in the center point underneath the gripper. The arrangement of the physical tendrils in this configuration (without the web) is depicted in Fig. 4.6.

In this iteration of the gripper, the concept for the web was to create 4 tubes out of a knit fabric (again composed of 96% polyester and 4% spandex) to individually encompass each tendril and then connect the tendril covers with flat panels composed the same fabric. In realizing this concept, we were able to create the web from one continuous piece of fabric, arranging the tendril covers into the same strip of fabric that constitutes the interconnecting panels. The idea was that this would be easier to manufacture and would lead to more uniform connections between the tendrils. The

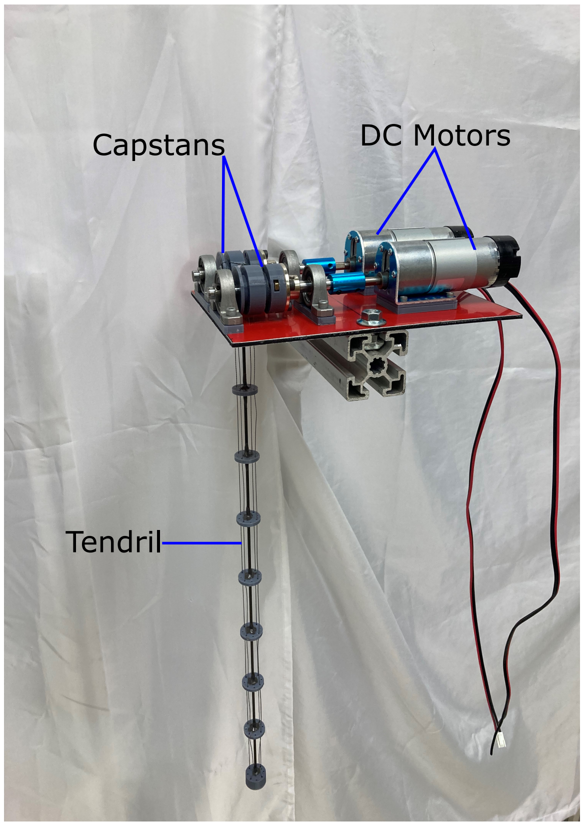

Figure 4.1: Single Tendril System

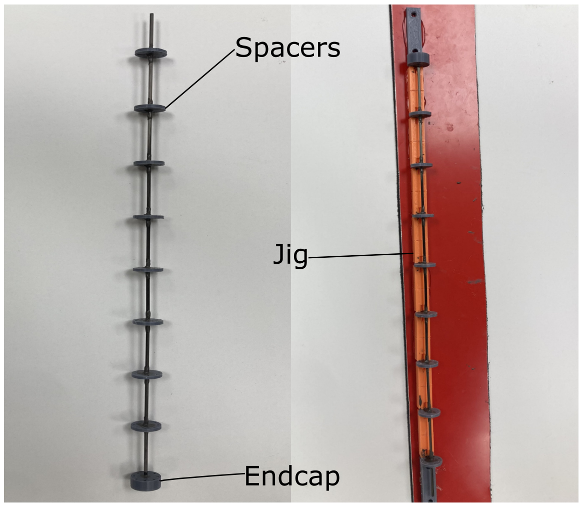

Figure 4.2: Tendril and Construction Jig

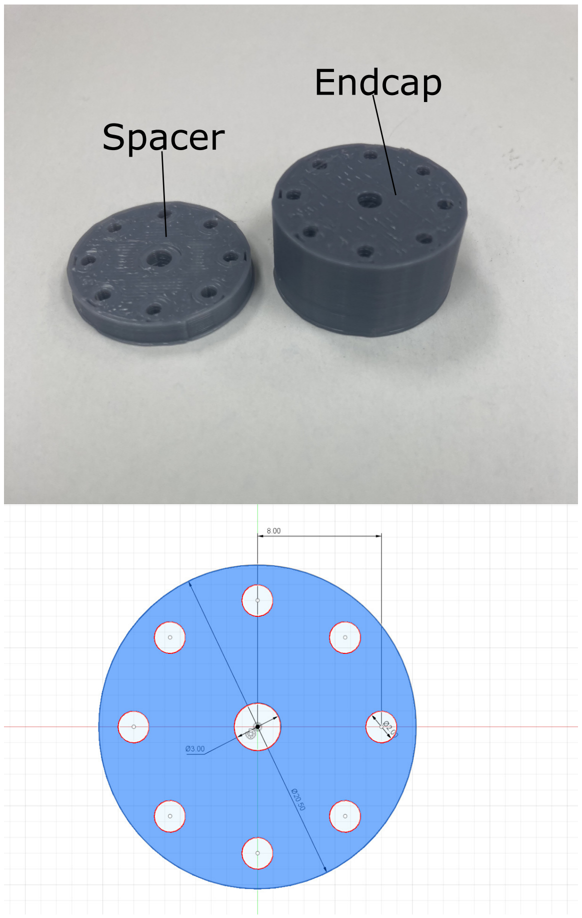

Figure 4.3: Spacer and End Cap Dimensions

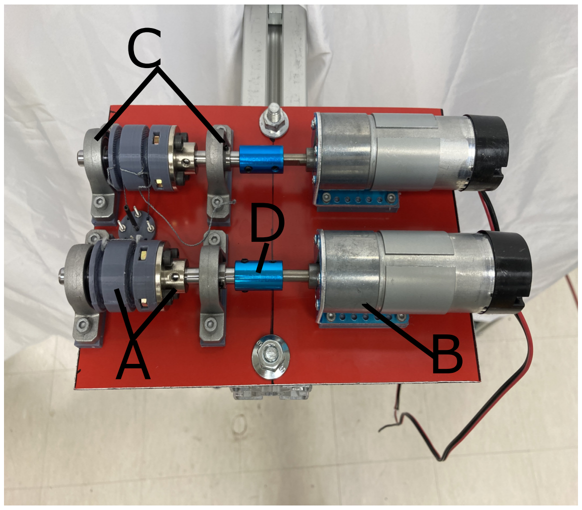

Figure 4.4: Actuation Package: (A) is the capstan and 8mm shaft mount, (B) is the DC Motors, (C) is the 8mm shaft holed pillow blocks, and (D) is the 8mm to 5mm shaft coupler

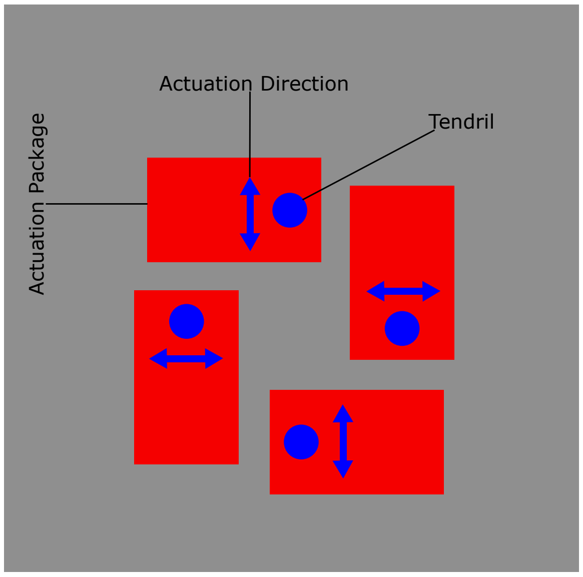

Figure 4.5: Tendril Actuation Package Base Arrangement

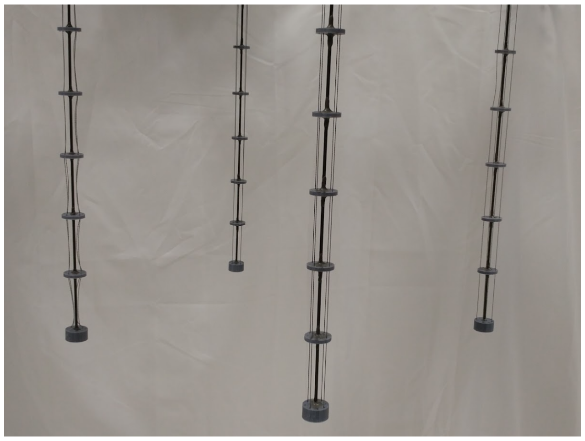

Figure 4.6: Tendril Arrangement With No Web Attached

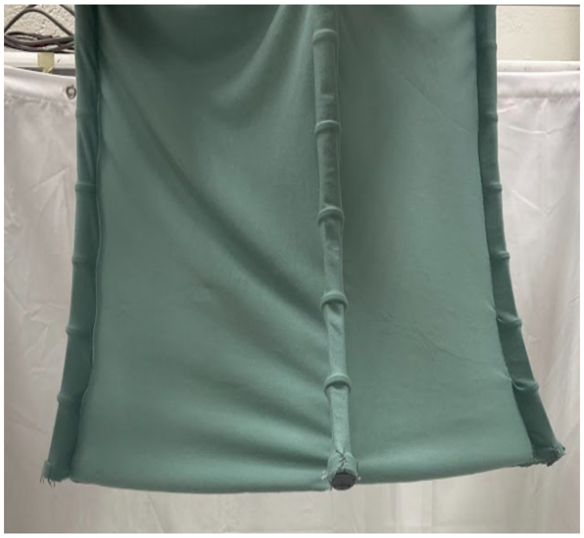
Figure 4.7: Fully Constructed Web

fully constructed web after placement over the tendril digits can be seen in Fig. 4.7.

As a result of an increase in hardware complexity, an Arduino Due was used to handle the control of the 4 tendrils. The pin layout can be seen in Fig. 4.8. In total, 8 motor drivers (MD10C R3) were used to control the 8 individual tendon motors (Andymark NeveRest Classic 60 Gearmotors), powered by a series of 12V DC power supplies.

4.2 Stiffness Control

Continuum robots, and more specifically tendrils, are very susceptible to external forces [32]. Prior to construction of the gripper it was predicted, and this was subsequently confirmed after construction, that the presence of the web would hinder the range of motion of the tendrils. In order to counter this, a stiffness control [25], [18] algorithm was implemented to allow for each tendril to compensate for the external spring-like force of the web.

The intent of the stiffness controller was to counter the passive stiffness of the fabric web. As in [25], we modeled this stiffness as a virtual spring which then enacts a restricting force on the end-effector of the form:

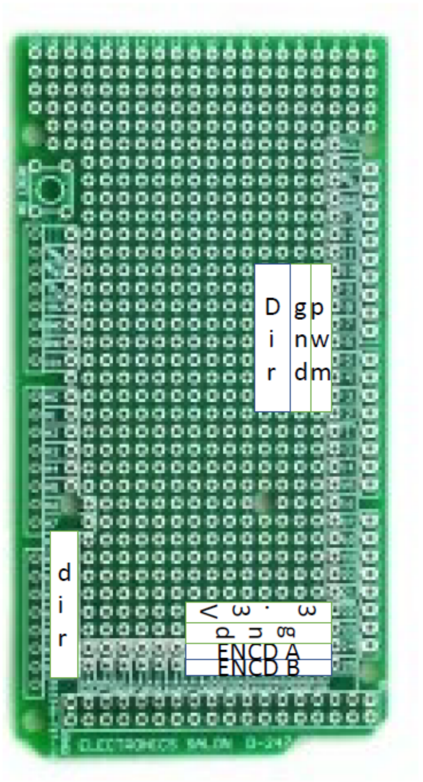

Figure 4.8: Circuit Pin Layout

$$\mathbf{F} = [K]\Delta\mathbf{x} \tag{4.1}$$

$$\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}^d \tag{4.2}$$

where K $\in \mathbb{R}^{n\times n}$ is the stiffness matrix of the material, $\Delta\mathbf{x} \in \mathbb{R}^{n\times 1}$ is the change in effector position with respect to an arbitrary (i.e. goal) position, and $\mathbf{F} \in \mathbb{R}^{n\times 1}$ is the force vector experienced at the end-effector. In practice for rigid-link robots, after calculating the predicted forces at the end-effector, the robot Jacobian can be used to calculate the effective torques needed to exert force $\mathbf{F}$ at the end-effector. In our case, we relate the force to a virtual torque represented by continuum robot kinematics, but calculated using the conventional equation:

$$\tau = [J]^T F \tag{4.3}$$

where the Jacobian matrix J relates end-effector forces to (virtual) joint torques.

In lieu of the rigid-link robot kinematics traditionally used for stiffness control, we utilize the constant-curvature kinematic model for continuum robots presented in [1], reproducing the equations relevant to a planar, fixed length continuum robot below:

$$u = \frac{l_2 - l_3}{d\sqrt{3}} \tag{4.4}$$

$$v = \frac{s(t) - l_1}{d} \tag{4.5}$$

where u and v are rotation values that relate direction and magnitude of bending and s is the length of the continuum backbone. The values l_1, l_2, and l_3 relate absolute tendon lengths for a 3 tendon configuration and d measures the (fixed) radial distance between the backbone and the tendons. In the case of the tendril digits used in the gripper, s is a constant value.

Using the model above, and considering the values u and v as virtual joint values, we derive the Jacobian whose transpose relates the virtual torques about u and v to the tensions in the tendons l_1, l_2, and l_3 using eqs. (4.4),(4.5):

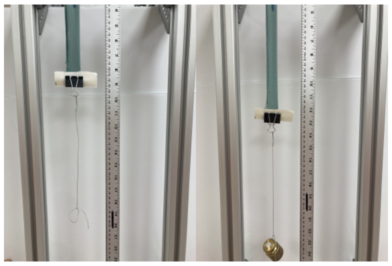
Figure 4.9: Spring Constant Measurement

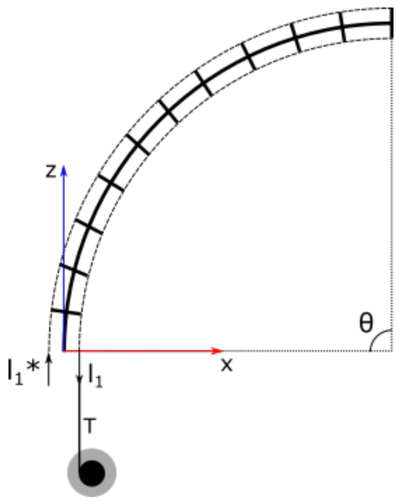

Figure 4.10: Diagram Depiction of Tendril Operation

$$J_{uv} = \begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{\sqrt{3}d} & -\frac{1}{\sqrt{3}d} \\ -\frac{1}{d} & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{l_1} \\ \dot{l_2} \\ \dot{l_3} \end{bmatrix} \tag{4.6}$$

where $\dot{u}$ and $\dot{v}$ are the derivatives of u, v, respectively, with respect to time. Likewise, $\dot{l_1}$, $\dot{l_2}$, and $\dot{l_3}$ are the rates of change for each tendon length over time. In the case of a planar continuum robot (the case for our gripper when simply bending in or out), we only need to use u or v to describe the motion of the robot. Given the simplicity of v, we select this value to define the remainder of the model, simplifying the Jacobian to:

$$J_v = -1/d \tag{4.7}$$

The remainder of the method is as follows: the displacement of the robot end-effector is measured according to the forward kinematics in [1] and multiplied by the measured material stiffness of the web (k), as discussed next.

The stiffness of the web material was measured as depicted in Fig. 4.9. A known weight was attached to the end of the material then the displacement was measured. This gave a stiffness value of g/mm, in the case of our test 2.04 g/mm. This value was be converted to N/m, which would give a value of 20.01 N/m. (It should note that the initial value determined from measurement was not implemented in the final experiments. The stiffness of the material used in the algorithm was a value that was determined empirically, and that value was 0.64 N/m)

The Jacobian J_v converts the external force to tendon tension (T). Tension was converted to torque exerted on the tendon capstan of radius r_c and the compensating torque was combined with the torque output of a standard PD controller τ_{PD}: F

$$F_b = k * \sqrt{\Delta x^2 + \Delta z^2} \tag{4.8}$$

$$T = J_v^T F_b \tag{4.9}$$

$$\tau_{stiff} = T * r_c \tag{4.10}$$

$$\tau_{out} = \tau_{stiff} + \tau_{PD} \tag{4.11}$$

The coordinates underlying the movement of the tendril are depicted in Fig. 4.10. From this figure, the distance the end-effector moves in the x and z place is a result of the amount on tendon, l_1, taken in. This in turn produces tension on the tendon, l_1, and the steps for the final torque applied to the motor have been discussed above.

The tendency of the fabric to stall a torque limited PD controller was measured by incrementally increasing the maximum torque available to the PD controller and then recording how much tendon the motor could retract. The relevant stall data, seen in Fig. 4.11, was collected by dynamically attempting to reach a desired tendon length while its tendril was constrained by the fabric, and then recording the actual amount of tendon that could be retracted. As an example, when the PD controller was limited to a maximum of 0.4 Nm of torque, it was only able to retract 10mm of tendon. This method was repeated until the maximum bending position was reached.

Also depicted in Fig. 4.11 is the output of the stiffness controller, based on tendon retraction and ideal kinematics. As can be seen, the stiffness controller output trends well with the observed

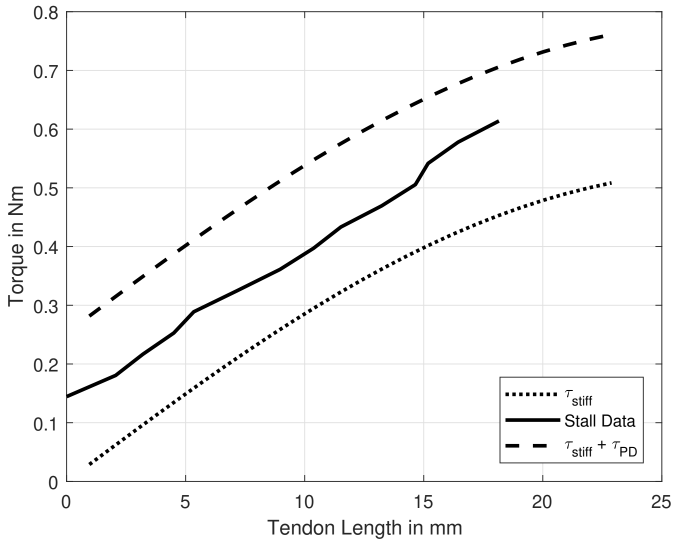

Figure 4.11: Comparing the tendency of tendon motor stall under load with the developed stiffness controller.

minimum torque needed to reach various tendon lengths (as related by the stall data). The addition of a bounded PD controller (i.e. the PD controller is limited to a maximum torque magnitude) clearly enables the combined controller to exert more than enough torque to reliably reach any position in the desired range of motion, providing valuable insight into the viability of stiffness controller. In the remainder of this work, the PD controller was limited to a maximum torque corresponding to a 13.7% duty cycle (35 PWM).

We illustrate the combined controller executing a desirable motion while constrained by the web, testing both with and without the stiffness control active. As seen in Fig. 4.12, the system attempts to retract 15 mm of tendon. It can be seen from the plot of L_1 without the stiffness controller that the bounded PD controller is not able to reach the desired position (it retracts approximately only 6.87 mm of tendon). In contrast, tendon L_2, which is not under tension, is able to smoothly release the desired 15mm. With the stiffness controller active, the L_1 motor is able to match the range of the L_2 motor, despite the building tension and resistance of the fabric.

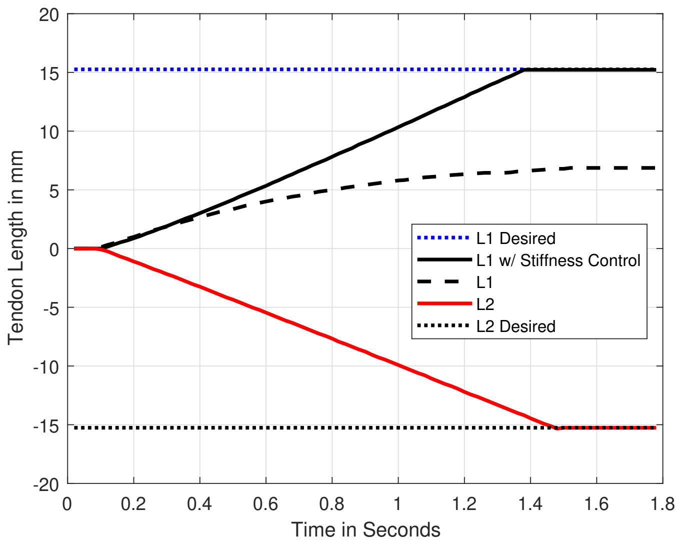

Figure 4.12: Stiffness Control Evaluation

4.3 Experimental Evaluation

Following the implementation of the stiffness controller, we conducted a series of experiments similar to those described in Section 3.2 for the pneumatically actuated gripper. With respect to the tendon-actuated gripper, all of the subsequent experiments were conducted with the gripper initially positioned above the target, with its base parallel to the floor. The various targets were then moved into the gripper's range of motion in a variety of ways.

A key goal of the experiments was to show that the gripper can grasp a range of objects without knowing its precise relative location with respect to, and to not damage, the target. The first set of experiments involved grasping a series of spherical objects with a wide variation in object diameter. The first object we highlight was a red foam ball (diameter of 18 cm), seen in Fig. 4.13 as a time lapse of the grasping motion. Initially the gripper was unactuated and once the ball was in the frame the gripper opened to provide a larger area for the ball to pass through into graspable range of gripper. Once the ball was within the volume of the web, the gripper could begin to actuate into the closed position, securing the object as seen from inside the gripper in the bottom right image in Fig. 4.13.

With the 18cm ball serving as a baseline, the next experiment aimed to capture a significantly smaller, 6.5cm diameter, sphere. The test was conducted similarly to the 18cm ball experiment

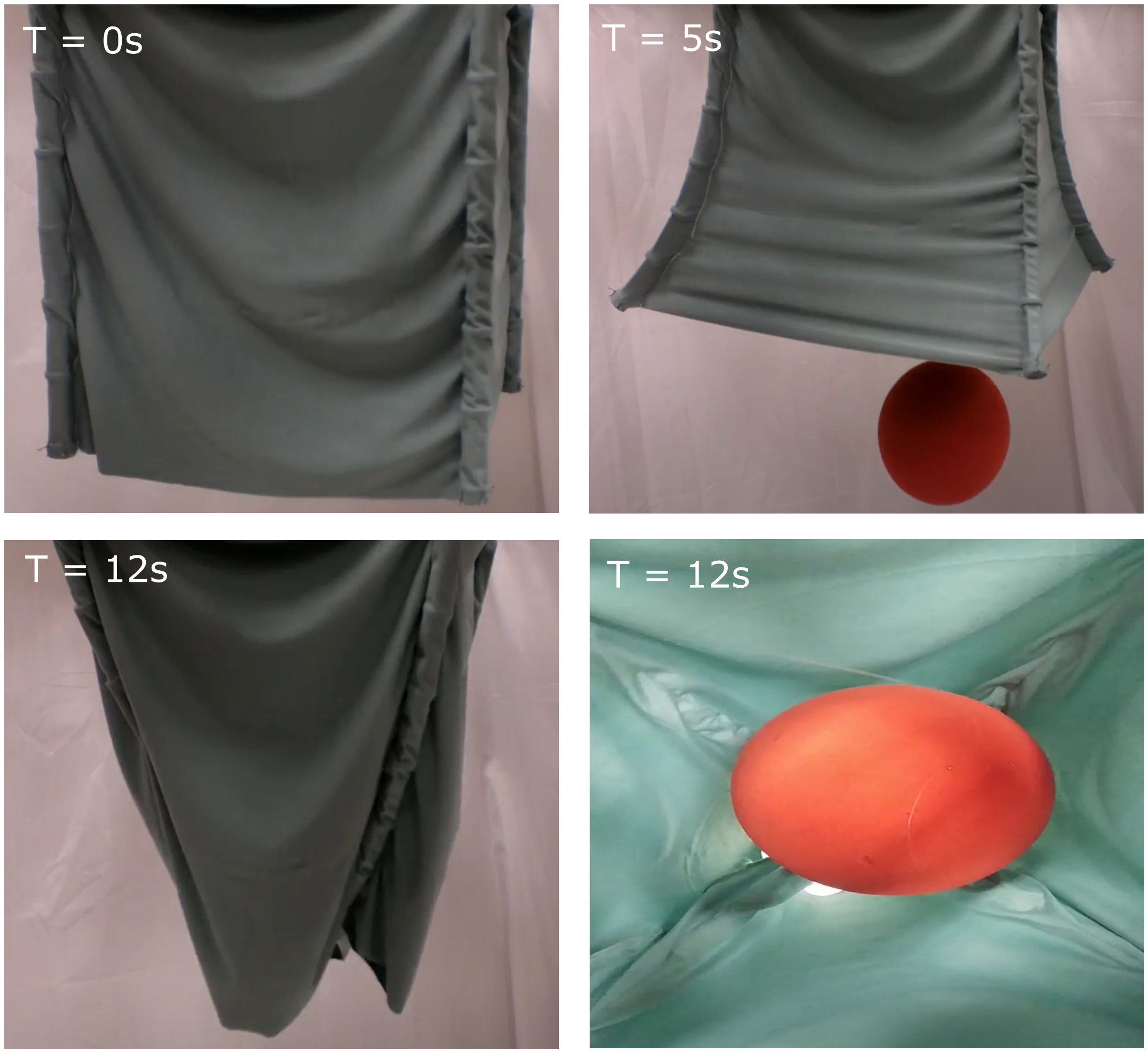

Figure 4.13: Time Lapse for Grasping 18cm ball

and so we only report on the ability to retain the smaller object. As seen in the left image of Fig. 4.14, the presence of the web allowed the gripper to retain the small object, which settled partially underneath one of the tendrils. Without the web, this object would clearly be able to fall through the gaps between the digits. We also performed the same task without the presence of the stiffness controller. The result, seen in the right image of the figure, was the inability to bend the digits of the gripper enough to fully close the web and thus allowed the small object to fall through.

A further size-related test was to determine if the gripper could manage to grasp and maintain stability of something relatively large in size. In this instance we again used a sphere, measuring

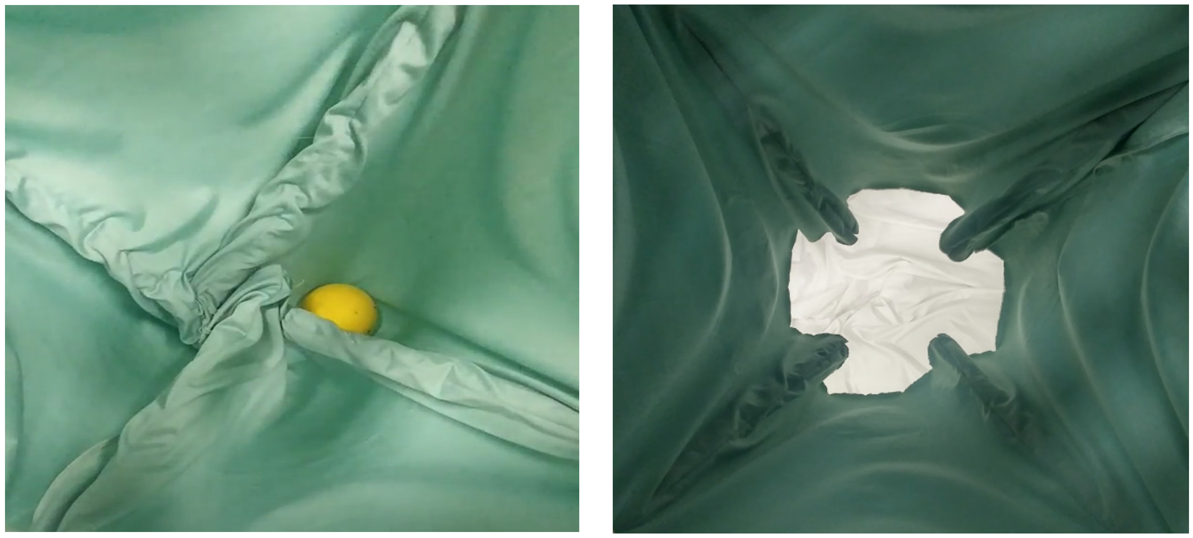

Figure 4.14: Grasping of small object (6.5cm diameter sphere). The web retains the object despite size (left). An absence of stiffness control prevents the gripper from fully closing to capture small objects (right).

30cm in diameter. Similar to the previous tests, the large sphere was moved into the range of the gripper as the gripper was opened to present a larger volume and closed to envelope the object. Images of the approach and capture of the object are seen in Fig. 4.15. The results from grasping these three objects convey the potential advantage of the compliant gripper to open to encompass larger objects and increase tolerance for error as well as capture and maintain a large range of object sizes.

After completing these tests that show the gripper could handle a range of different sizes, a further next experiment was conducted to depict how the gripper could handle non-uniform shapes. In this test, the target was a box measuring 39cm×31cm×18.5cm. It is worth noting that the size of this box was larger than the largest sphere tested. As with the previous tests, the box was brought to the range of the gripper and then the gripper was allowed to close once the box was mostly in the gripper's enclosing volume. As seen in Fig. 4.16, the gripper was still able to secure and stabilize the box, utilizing both the tendrils for supporting the mass and the webbed material for security - a novel grasp mode for robot grippers.

The above test were conducted using the base configuration depicted in Fig. 4.5, i.e. with the tendril digits evenly arranged in the "palm" of the gripper. The following experiments demonstrate the gripper being reconfigured to different base palm positions, illustrating that this is beneficial for

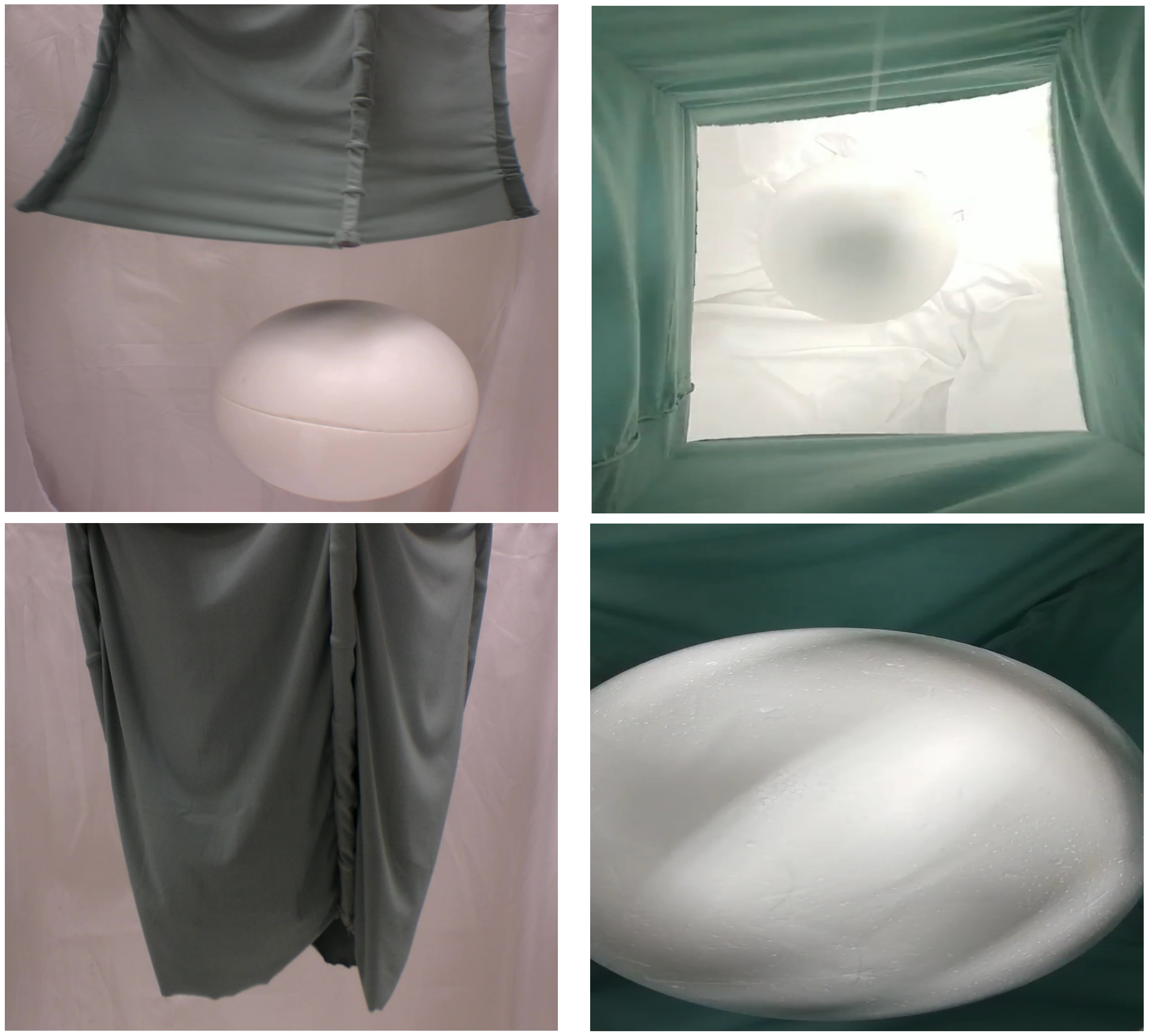

Figure 4.15: Grasping of large, uniform object.

different classes of target being stabilized. The base configuration for the one of these experiments is shown in Fig. 4.17. The bases were configured so that the shape of the web would resemble that of a rhombus. This configuration proved to be beneficial for grasping a large box that was long but quite thin. Images from this experiment are depicted in Fig. 4.18. This experiment shows that the ability to reconfigure the gripper allowed for a successful grasp of a box that would have not been possible with the evenly spaced base configuration.

The next experiment reported herein was an attempt to capture a drone. The drone had a length of 14 cm, a width of 14 cm, and a height of 4.5 cm. In this experiment the drone was

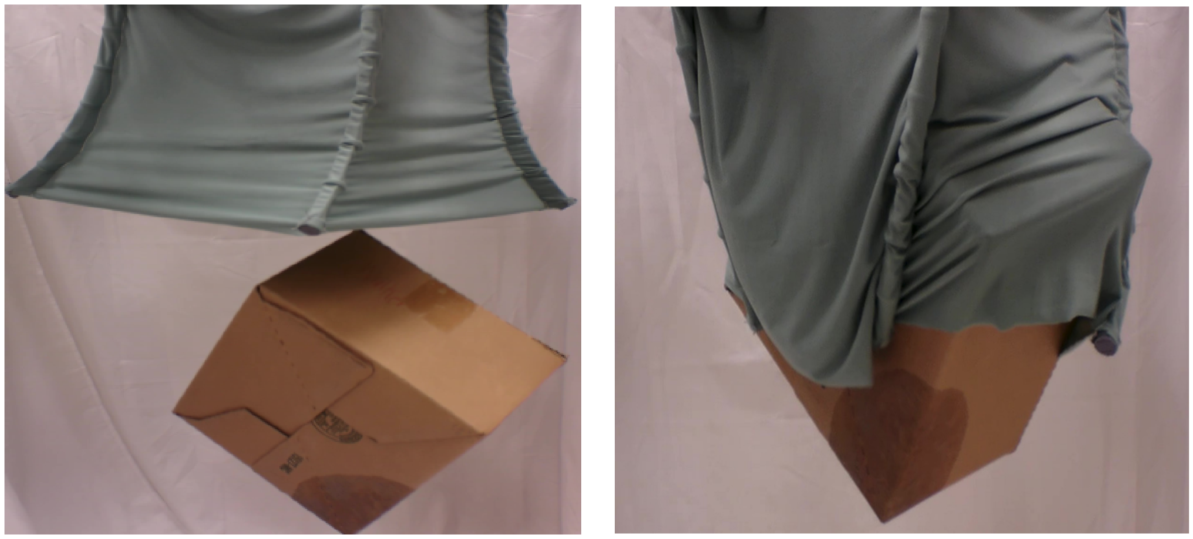
Figure 4.16: Capturing of large, non-uniform object.

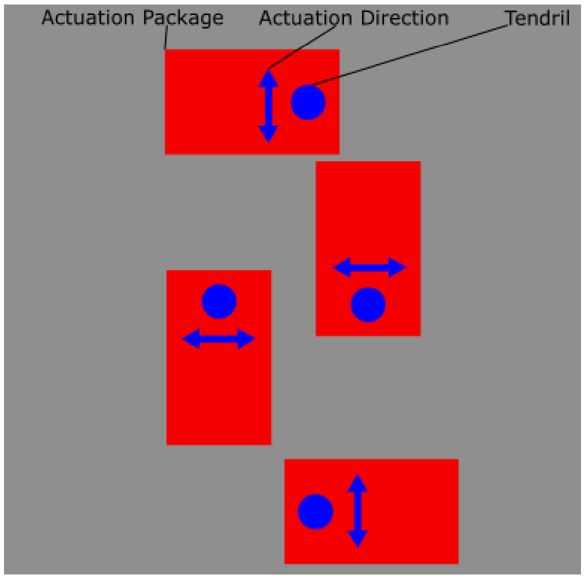

Figure 4.17: Tendril Actuation Package Rhombus Arrangement

flown into the range of motion of the gripper. Once the drone was in range, the gripper closed and captured the drone. Here again the grasp strongly benefited from the web material in retaining the drone between the digits, despite the drone's small size relative to the gripper.

The gripper was additionally configured to resemble the shape of a kite. The configuration is depicted in Fig. 4.20. The range of motion in this configuration is depicted in Fig. 4.21. The experiment conducted in this configuration was done on the object found in Fig. 4.22. It is important

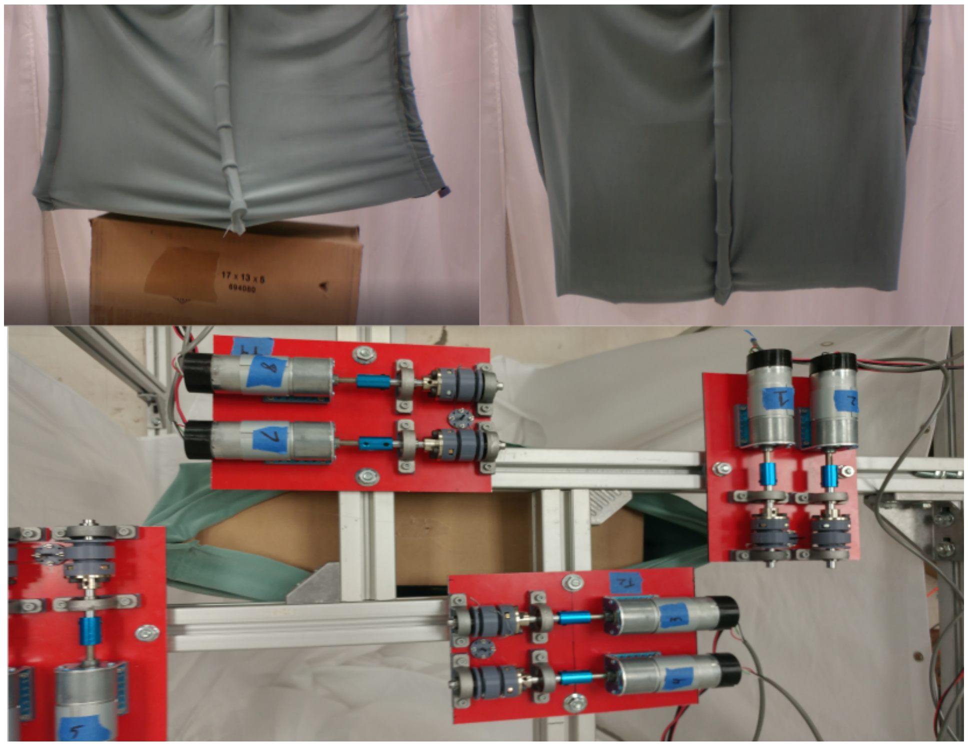

Figure 4.18: Rhombus Arrangement Grip Test

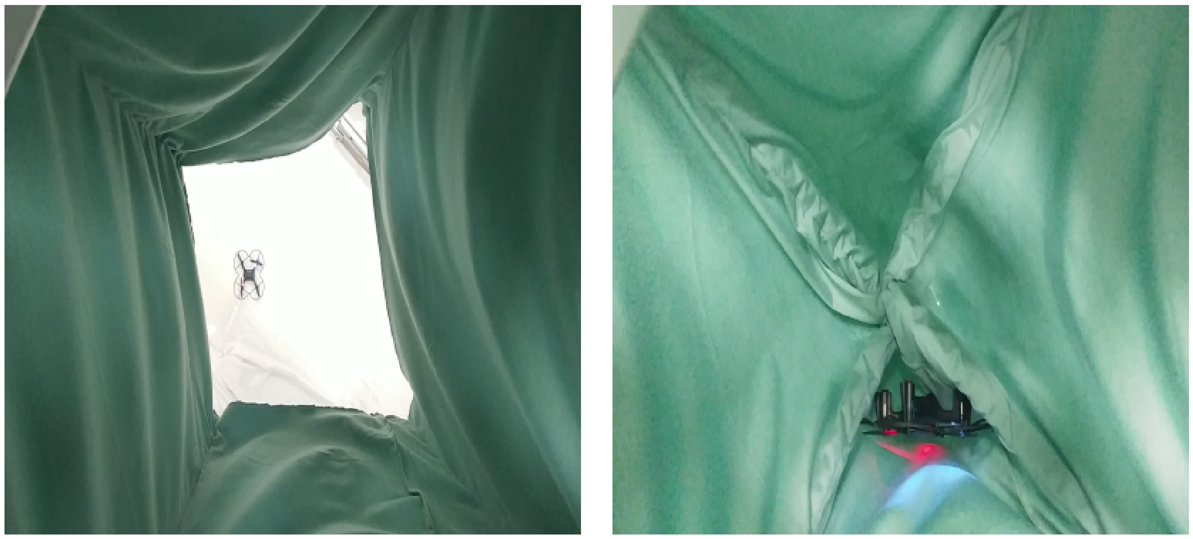

Figure 4.19: Drone Grasping

to note the object size and mass distributing is irregular. From this figure it can be see that the cluster of three tendrils is useful for supporting the side with more mass. While this lone tendril is enough to support the lighter side of the object. This configuration allows for a wider range of

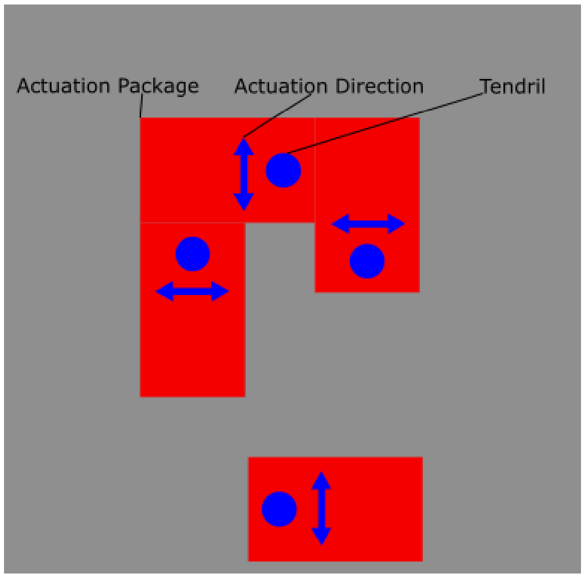

Figure 4.20: Tendril Actuation Package Kite Arrangement

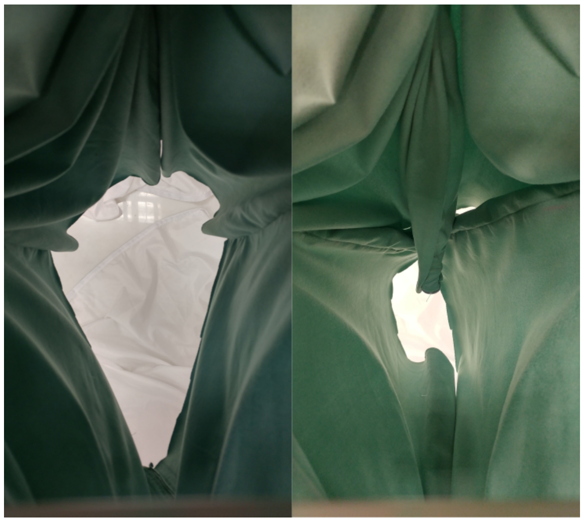

Figure 4.21: Opening and Closing In Kite Configuration

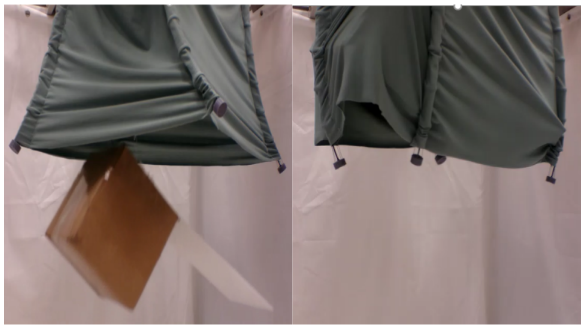

Figure 4.22: Satellite Grasping

objects to be grasped.

Further experiments highlight the fact that the tendrils can be individually actuated, which offers a variety of other ways the gripper can be configured to grasp objects. In one such experiment, detailed in Fig. 4.23, the approach was to actuate pairs of two tendons at a time. The left pair of tendons were actuated to bend in a higher curvature than the right pair, to grasp a drone. Then once the drone was in range, the right pair of tendons were actuated to bend to the same amount as the left pair. This allowed for the drone to pinched by the tendrils and stabilized in a novel grasp. (Note the process is illustrated here without the web, to best allow the motions of the digits to be viewed.)

The final experiment reported herein was conducted in the base (all tendril bases equally separated) configuration. The goal of these experiments was to show the gripper's ability to grasp a target that resembles objects that might possibly be found in space. The target used for the experiment can be found in Fig. 4.24. The results from this experiment can be seen in Fig. 4.25. This experiment once more demonstrates that the gripper has the ability, using its passive web, to capture and stabilize targets that a conventional gripper would likely find challenging without additional local sensing.

Overall, the results of these experiments show that the gripper has the ability to capture a wide range of objects without the need to sense the target's precise location. It also has the ability to gently capture objects that would be damaged by a traditional gripper. The implementation of stiffness control allowed the tendrils to compensate for the external forces of the fabric. These features allow for a wide range of objects to be successfully grasped.

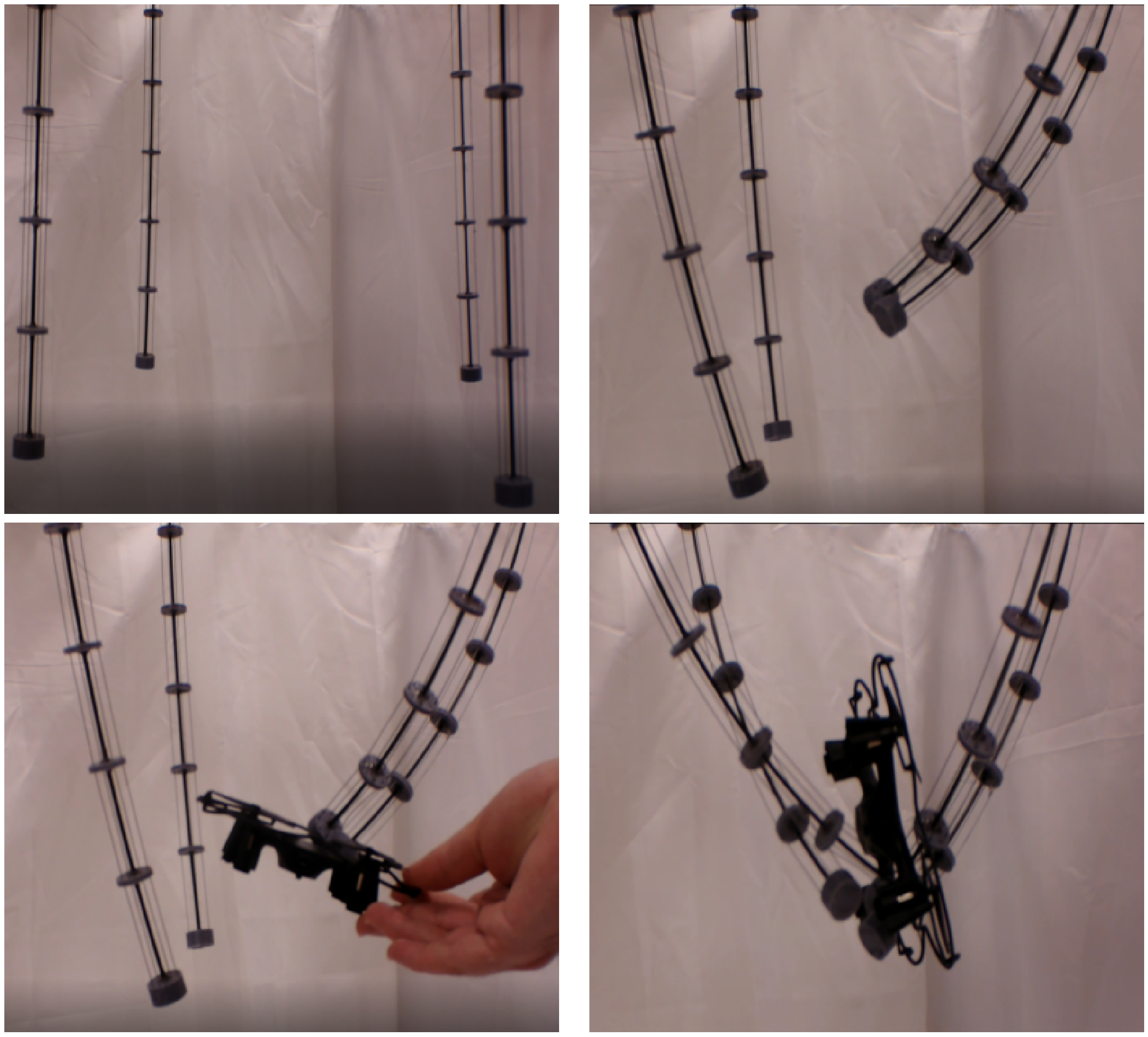

Figure 4.23: Actuating Individual Tendrils Experiment

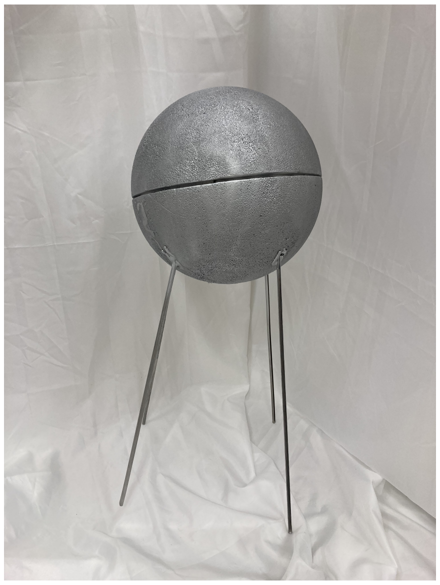
Figure 4.24: Object Similar To What Might Be Found In Space

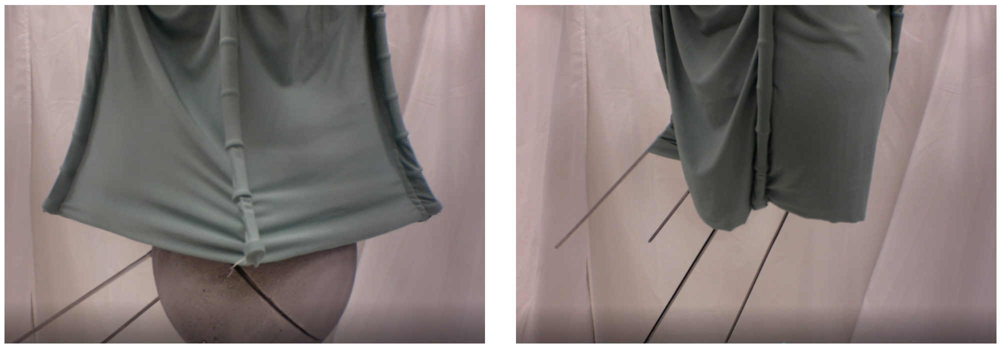
Figure 4.25: Space Object Experiment

Chapter 5

Conclusions and Suggestions for Future Research

5.1 Conclusions

We have discussed the inherent advantages and long term potential of using a flexible web connecting compliant digits within a robot gripper. Incorporation of a soft web enables robust and adaptive grasps which compensate for inaccuracies inevitable in general grasping operations. This is particularly advantageous when there is significant relative motion between object and gripper. The soft web structure also provides advantages when grasping objects that are fragile or small relative to the grasp envelope.

We have introduced and described two new compliant web-based gripper prototypes: one based on pneumatically actuated digits, and one with the digits remotely actuated by tendons. Experiments with the prototypes demonstrate the ability of the underlying design to perform novel and adaptive grasping. The results in this thesis are the first to demonstrate web-based hands with tendon-actuated continuum digits, the first to incorporate independent digit control in such hands, and first to incorporate variable bases for the digits. The empirical results with the prototypes demonstrate the potential of such grippers for a variety of applications, including space-based capture of orbital debris and of non-cooperative targets.

5.2 Suggestions for Future Research

Possible extensions of the research could include consideration of alternative materials for the web, varying the number of digits, active actuation of the digit bases, and provision of a wider range of motion for the digit bases.

Over the course of this project, there were numerous issues encountered in conducting the research, which would benefit from improvements. The first would be choosing a webbing material that is easier to work with, and/or outsourcing the web manufacture to a more experienced professional. The web was sewn by hand by the author, and because the material is difficult to sew with lack of experience the web fitting was imperfect. This resulted in uneven torsion forces being applied to the tendrils, which periodically caused problems in conducting the experiments. If the web had been more uniform then it would have had less of an impact on the performance of the gripper.

Another area that should be improved in future research is the tendril backbone material. Improvements could possibly come from using a larger diameter of carbon fiber, or selecting a less brittle material. The torsioning mentioned above caused backbones to break several times, a problem compounded when grasping objects that caused the backbones to bend significantly. The issue of torsioning could be mitigated by increasing the diameter of the backbone, thus increasing the torsional rigidity. This could also be accomplished by using a backbone material that is less likely to break or fracture under the loads from the experiments.

Another area of improvement would be to incorporate some form of force or tension sensing. This could allow for safety measures to be coded so that the tendons do not experience levels of force or tension that can cause damage to the system.

A further improvement that could be made to this system is to decrease the size of the actuation modules. This could be achieved by using a single motor and capstan combination. This would require more experimentation to ensure the desired range of motion could be achieved. Using higher gearing could also be an approach to using only one motor for actuation.

Perhaps the most interesting change that could allow for more robust grasping would be to increase the number of actuation modules from four to eight, more akin to the vampire squid morphology. This could allow for a larger grip as well as increase the load bearing capability via the extra four tendrils.

Appendices

Appendix A Pneumatically Actuated Web Control Code

```
uint32_t value;

const byte interruptPin = 2;

volatile byte state = 0;


void setup(void) {

  Serial.begin(9600);

  // For MCP4725A0 the address is 0x60 or 0x61

  // 0x60 fingers

  // 0x61 palm

  // Initialize both pressures to zero (0)


  pinMode(interruptPin, INPUT_PULLUP);

  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, FALLING);

  dac.begin(0x61); // palm pressure

  dac.setVoltage(0, false);

  delay(20);

  dac.begin(0x60); // fingers pressure

  dac.setVoltage(0, false);

  delay(20);

  }


void loop(void) {


  if (state == 0){

    value = 0;

    dac.begin(0x61);

    dac.setVoltage(value, false);

    delay(20);

    dac.begin(0x60);
    dac.setVoltage(value, false);
  }


    if (state == 1){
    value = 1260;
    dac.begin(0x60); // left regulator
    dac.setVoltage(value, false);
  }


    if (state == 2){
    value = 0;
    dac.begin(0x60);
    dac.setVoltage(value,false);
    delay(20);
    value = 1260;
    dac.begin(0x61); // right regulator
    dac.setVoltage(value, false);
  }



}

void blink() {
  static volatile unsigned long last_time = 0;
  if (millis()-last_time > 300) {
      last_time = millis();
      state = state + 1;
      Serial.println("Hello");
      if (state > 2){
      state = 0;
      }
```

```
}
```

Appendix B Tendon Actuated Web Control Code

B.1 Arduino Control Code

```
    //Includes
#include "Arduino.h"
#include <DueTimer.h>
//#include <PrintEx.h>
#include <math.h>
#include "continuumDue_FullOctWeb.h"

//scalar converting spool size and counts per revolution to length
// scaleE2L = pi*{spoolDiameter}/{countsPerRevolution}
const float scaleE2L = 3.14*0.0254/2000; //[rad*m/counts]
#define numMotors 8

////////////////////////////////////////////////////////////////////////
//Definitions (duplicated in continuumDue library, can be deleted here)
////////////////////////////////////////////////////////////////////////
////m(otor)Enc(oder)_{A/B}{motornumber}
//#define mEnc_A1 24
//#define mEnc_B1 25
//#define mEnc_A2 30 //26
//#define mEnc_B2 31 //27
//#define mEnc_A3 36 //28
//#define mEnc_B3 37 //29
//#define mEnc_A4 28 //30
//#define mEnc_B4 29 //31
//#define mEnc_A5 34 //32   chA
//#define mEnc_B5 35 //33   chI
//#define mEnc_A6 40 //34
//#define mEnc_B6 41 //35
```

```
//#define mEnc_A7 26 //36
//#define mEnc_B7 27 //37
//#define mEnc_A8 32 //38
//#define mEnc_B8 33 //39
//#define mEnc_A9 38 //40
//#define mEnc_B9 39 //41
//
////Motor pwm pins for analog write
////pwmM(otor){motorNumber}
//#define pwmM1  3
//#define pwmM2  6
//#define pwmM3  9
//#define pwmM4  5
//#define pwmM5  8
//#define pwmM6 11
//#define pwmM7  4
//#define pwmM8  7
//#define pwmM9 10
//
////direction pins for digital write to motor driver
////dir(ection)M(otor){motorNumber}
//#define dirM1 53
//#define dirM2 50
//#define dirM3 47
//#define dirM4 51
//#define dirM5 48
//#define dirM6 45
//#define dirM7 52
//#define dirM8 49
//#define dirM9 46
//
```

```
////essential for checking encoders
//const int8_t lookup_table[] = {0,-1,1,0,1,0,0,-1,-1,0,0,1,0,1,-1,0};
////////////////////////////////////////

const float secLengthM = 0.1; //[m]
const double db = 0.0452; //radial tendon displacement [m]

//globals
void sendMessage();
String inString = ""; // a string to hold incoming data
boolean stringComplete = false;
float lengths[3] = {0,0,0};
long errors[numMotors];
long setPoints[numMotors/2];
long WindAngle;
long motor5Count_previous;
boolean WindPresent = true;
int currentTime = 0; //[seconds]
int TimeThreshold = 10; // [seconds]
double v;
double tendon_distance=0.00697;
double deltaL1;
double PPR = 1680;
//double setPointM1;
//double setPointM2;
double cap_radius = 0.0102;

volatile long motor1Count = 0;
volatile long motor2Count = 0;
volatile long motor3Count = 0;
volatile long motor4Count = 0;
volatile long motor5Count = 0;
volatile long motor6Count = 0;
volatile long motor7Count = 0;
volatile long motor8Count = 0;
//volatile long motor9Count = 0;

long int counter=0;

//Pre-setup Setups
//PrintEx serial = Serial;

uint32_t pgain = 5000;
uint32_t dgain = 3;

//Declaring robot object(contains motor information, only used for initialization and safety reasons
//Robot robot;
//declaring robot motors
//DCMotor motorName(PWMpin,dirOutPin,pgain,dgain);
DCMotor motor1(pwmM1,dirM1,pgain,dgain,&motor1Count);
DCMotor motor2(pwmM2,dirM2,pgain,dgain,&motor2Count);
DCMotor motor3(pwmM3,dirM3,pgain,dgain,&motor3Count);
DCMotor motor4(pwmM4,dirM4,pgain,dgain,&motor4Count);
DCMotor motor5(pwmM5,dirM5,pgain,dgain,&motor5Count);
DCMotor motor6(pwmM6,dirM6,pgain,dgain,&motor6Count);
DCMotor motor7(pwmM7,dirM7,pgain,dgain,&motor7Count);
DCMotor motor8(pwmM8,dirM8,pgain,dgain,&motor8Count);


//DCMotor *motor5  = robot.getMotor(5,pwmM5,dirM5,pgain,dgain);
//DCMotor *motor6  = robot.getMotor(6,pwmM6,dirM6,pgain,dgain);
//DCMotor *motor7  = robot.getMotor(7,pwmM7,dirM7,pgain,dgain);
```

```
//DCMotor *motor8  = robot.getMotor(8,pwmM8,dirM8,pgain,dgain);
//DCMotor *motor9  = robot.getMotor(9,pwmM9,dirM9,pgain,dgain);



void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  Serial.println("hello world");

  Timer2.attachInterrupt(sendMessage);

  inString.reserve(200);

  pinMode(dirM1, OUTPUT);
  pinMode(pwmM1, OUTPUT);
  pinMode(dirM2, OUTPUT);
  pinMode(pwmM2, OUTPUT);
  pinMode(dirM3, OUTPUT);
  pinMode(pwmM3, OUTPUT);

  ////Extra motor controller declarations
  pinMode(dirM4, OUTPUT);
  pinMode(pwmM4, OUTPUT);
  pinMode(dirM5, OUTPUT);
  pinMode(pwmM5, OUTPUT);
  pinMode(dirM6, OUTPUT);
  pinMode(pwmM6, OUTPUT);

  pinMode(dirM7, OUTPUT);
  pinMode(pwmM7, OUTPUT);
  pinMode(dirM8, OUTPUT);
  pinMode(pwmM8, OUTPUT);
//  pinMode(dirM9, OUTPUT);
//  pinMode(pwmM9, OUTPUT);

  digitalWrite(dirM1, HIGH);
  digitalWrite(dirM2, HIGH);
  digitalWrite(dirM3, HIGH);

  ////Extra direction pin declarations
  digitalWrite(dirM4, HIGH);
  digitalWrite(dirM5, HIGH);
  digitalWrite(dirM6, HIGH);

  digitalWrite(dirM7, HIGH);
  digitalWrite(dirM8, HIGH);
//  digitalWrite(dirM9, HIGH);

  /////////////////////////////
  ////ENCODER DECLARATION//////
  /////////////////////////////

  pinMode(mEnc_A1, INPUT_PULLUP);      // sets pin A as input
  digitalWrite(mEnc_A1, HIGH);  // turn on pullup resistors
  pinMode(mEnc_B1, INPUT_PULLUP);      // sets pin B as input
  digitalWrite(mEnc_B1, HIGH);  // turn on pullup resistors
  attachInterrupt(digitalPinToInterrupt(mEnc_A1), motor1Interrupt, CHANGE);
  attachInterrupt(digitalPinToInterrupt(mEnc_B1), motor1Interrupt, CHANGE);

  pinMode(mEnc_A2, INPUT_PULLUP);      // sets pin A as input
  digitalWrite(mEnc_A2, HIGH);  // turn on pullup resistors
  pinMode(mEnc_B2, INPUT_PULLUP);      // sets pin B as input
  digitalWrite(mEnc_B2, HIGH);  // turn on pullup resistors
  attachInterrupt(digitalPinToInterrupt(mEnc_A2), motor2Interrupt, CHANGE);
  attachInterrupt(digitalPinToInterrupt(mEnc_B2), motor2Interrupt, CHANGE);

  pinMode(mEnc_A3, INPUT_PULLUP);      // sets pin A as input
  digitalWrite(mEnc_A3, HIGH);  // turn on pullup resistors
  pinMode(mEnc_B3, INPUT_PULLUP);      // sets pin B as input
  digitalWrite(mEnc_B3, HIGH);  // turn on pullup resistors
  attachInterrupt(digitalPinToInterrupt(mEnc_A3), motor3Interrupt, CHANGE);
  attachInterrupt(digitalPinToInterrupt(mEnc_B3), motor3Interrupt, CHANGE);

  ////Extra encoder declarations
  pinMode(mEnc_A4, INPUT_PULLUP);      // sets pin A as input
  digitalWrite(mEnc_A4, HIGH);  // turn on pullup resistors
  pinMode(mEnc_B4, INPUT_PULLUP);      // sets pin B as input
  digitalWrite(mEnc_B4, HIGH);  // turn on pullup resistors
  attachInterrupt(digitalPinToInterrupt(mEnc_A4), motor4Interrupt, CHANGE);
  attachInterrupt(digitalPinToInterrupt(mEnc_B4), motor4Interrupt, CHANGE);
//
  pinMode(mEnc_A5, INPUT_PULLUP);      // sets pin A as input
  digitalWrite(mEnc_A5, HIGH);  // turn on pullup resistors
  pinMode(mEnc_B5, INPUT_PULLUP);      // sets pin B as input
  digitalWrite(mEnc_B5, HIGH);  // turn on pullup resistors
  attachInterrupt(digitalPinToInterrupt(mEnc_A5), motor5Interrupt, CHANGE);
  attachInterrupt(digitalPinToInterrupt(mEnc_B5), motor5Interrupt, CHANGE);

//  pinMode(42,INPUT_PULLUP);
//  attachInterrupt(digitalPinToInterrupt(42),indexInterrupt,RISING);
//
  pinMode(mEnc_A6, INPUT_PULLUP);      // sets pin A as input
  digitalWrite(mEnc_A6, HIGH);  // turn on pullup resistors
  pinMode(mEnc_B6, INPUT_PULLUP);      // sets pin B as input
  digitalWrite(mEnc_B6, HIGH);  // turn on pullup resistors
  attachInterrupt(digitalPinToInterrupt(mEnc_A6), motor6Interrupt, CHANGE);
  attachInterrupt(digitalPinToInterrupt(mEnc_B6), motor6Interrupt, CHANGE);
//
  pinMode(mEnc_A7, INPUT_PULLUP);      // sets pin A as input
  digitalWrite(mEnc_A7, HIGH);  // turn on pullup resistors
  pinMode(mEnc_B7, INPUT_PULLUP);      // sets pin B as input
  digitalWrite(mEnc_B7, HIGH);  // turn on pullup resistors
  attachInterrupt(digitalPinToInterrupt(mEnc_A7), motor7Interrupt, CHANGE);
  attachInterrupt(digitalPinToInterrupt(mEnc_B7), motor7Interrupt, CHANGE);
//
  pinMode(mEnc_A8, INPUT_PULLUP);      // sets pin A as input
  digitalWrite(mEnc_A8, HIGH);  // turn on pullup resistors
  pinMode(mEnc_B8, INPUT_PULLUP);      // sets pin B as input
  digitalWrite(mEnc_B8, HIGH);  // turn on pullup resistors
  attachInterrupt(digitalPinToInterrupt(mEnc_A8), motor8Interrupt, CHANGE);
  attachInterrupt(digitalPinToInterrupt(mEnc_B8), motor8Interrupt, CHANGE);
//
//  pinMode(mEnc_A9, INPUT_PULLUP);      // sets pin A as input
//  digitalWrite(mEnc_A9, HIGH);  // turn on pullup resistors
//  pinMode(mEnc_B9, INPUT_PULLUP);      // sets pin B as input
//  digitalWrite(mEnc_B9, HIGH);  // turn on pullup resistors
//  attachInterrupt(digitalPinToInterrupt(mEnc_A9), motor9Interrupt, CHANGE);
//  attachInterrupt(digitalPinToInterrupt(mEnc_B9), motor9Interrupt, CHANGE);

Timer2.start(200000); // Calls every 100ms

  delay(1000);
//  digitalWrite(14,HIGH);
```

```
}

//Do not introduce delays into main loop, it will mess with the PD control
void loop() {//runs continuously
  counter++;
  currentTime = millis()/1000;
  deltaL1 = v * tendon_distance;
 // setPointM1 = deltaL1*PPR/(2*PI*cap_radius);



//  //there is a better way to do this, but updating errors
  errors[0] = ((int)setPoints[0]) - motor1Count;
  errors[1] = ((int)setPoints[0]) - motor2Count;
  errors[2] = ((int)setPoints[1]) - motor3Count;
  errors[3] = ((int)setPoints[1]) - motor4Count;
  errors[4] = ((int)setPoints[2]) - motor5Count;
  errors[5] = ((int)setPoints[2]) - motor6Count;
  errors[6] = ((int)setPoints[3]) - motor7Count;
  errors[7] = ((int)setPoints[3]) - motor8Count;
//  errors[1] = setPoints[1] - motor2Count;
//  errors[2] = setPoints[2] - motor3Count;
//  errors[3] = setPoints[3] - motor4Count;

  //send error values to update function for PD control
  if (setPoints[0] < 0){
      motor1.update(errors[0],0);
      motor2.update(errors[1],2);
  }
  if (setPoints[1] < 0){
      motor3.update(errors[2],0);
      motor4.update(errors[3],2);
```

```
}
if (setPoints[2] < 0){
    motor5.update(errors[4],0);
    motor6.update(errors[5],2);
}
if (setPoints[3] < 0){
    motor7.update(errors[6],0);
    motor8.update(errors[7],2);
}









if (setPoints[0] > 0){
    motor1.update(errors[0],2);
    motor2.update(errors[1],0);
}
if (setPoints[1] > 0){
    motor3.update(errors[2],2);
    motor4.update(errors[3],0);

}
if (setPoints[2] > 0){
    motor5.update(errors[4],2);
    motor6.update(errors[5],0);
}
if (setPoints[3] > 0){
    motor7.update(errors[6],2);
```

```
        motor8.update(errors[7],0);
    }







    if (setPoints[0] == 0){
        motor1.update(errors[0],0);
        motor2.update(errors[1],0);
    }
    if (setPoints[1] == 0){
        motor3.update(errors[2],0);
        motor4.update(errors[3],0);

    }
    if (setPoints[2] == 0){
        motor5.update(errors[4],0);
        motor6.update(errors[5],0);
    }
    if (setPoints[3] == 0){
        motor7.update(errors[6],0);
        motor8.update(errors[7],0);
    }

//  if (setPointM1 > 0){
//      motor1.update(errors[0],2);
//      motor2.update(errors[1],0);
//      motor3.update(errors[2],2);
//      motor4.update(errors[3],0);
//      motor5.update(errors[4],2);
//      motor6.update(errors[5],0);
//      motor7.update(errors[6],2);
//      motor8.update(errors[7],0);
//  }
////  if (setPointM1 < 0){
////      motor1.update(errors[0],0);
////      motor2.update(errors[1],0);
////      motor3.update(errors[2],0);
////      motor4.update(errors[3],0);
////      motor5.update(errors[4],0);
////      motor6.update(errors[5],0);
////      motor7.update(errors[6],0);
////      motor8.update(errors[7],0);
////  }
////  if (setPointM1 > 0){
////      motor1.update(errors[0],0);
////      motor2.update(errors[1],0);
////      motor3.update(errors[2],0);
////      motor4.update(errors[3],0);
////      motor5.update(errors[4],0);
////      motor6.update(errors[5],0);
////      motor7.update(errors[6],0);
////      motor8.update(errors[7],0);
////  }
//   if (setPointM1 == 0){
//      motor1.update(errors[0],0);
//      motor2.update(errors[1],0);
//      motor3.update(errors[2],0);
//      motor4.update(errors[3],0);
//      motor5.update(errors[4],0);
//      motor6.update(errors[5],0);
//      motor7.update(errors[6],0);
//      motor8.update(errors[7],0);
//  }
//  motor1.update(errors[0],0);
//  motor2.update(errors[1],1);
//  motor3.update(errors[2]);
//  motor4.update(errors[2]);

}

//device output
void sendMessage(){

//  serial.printf( "%d %d %d %d\n", motor1Count,motor2Count,motor3Count,motor4Count);
  Serial.print(motor1Count);
  Serial.print("\t");
  Serial.print(motor2Count);
  Serial.print("\t");
  Serial.print(motor3Count);
  Serial.print("\t");
  Serial.print(motor4Count);
  Serial.print("\t");
  Serial.print(motor5Count);
  Serial.print("\t");
  Serial.print(motor6Count);
  Serial.print("\t");
  Serial.print(motor7Count);
  Serial.print("\t");
  Serial.print(motor8Count);
  Serial.println("\t");
  //Serial.print(setPoint)
}


int charCount=0;
String dummy = "";
void serialEvent(void){
  int i;
  while (Serial.available() > 0) {
    int inChar = Serial.read();
    inString += (char)inChar;

    // if you get a newline, print the string, then the string’s value:
    if ((char)inChar == ’,’){
      charCount++; //Serial.print(count);
    }
    if (inChar == ’\n’) {
      charCount++;
      if(charCount==(numMotors/2)){
        for(i=0;i<charCount;i++){
          int index = inString.indexOf(’,’);
          dummy = inString.substring(0,index);
          setPoints[i] = dummy.toFloat();
          if (setPoints[i] > 450){
            setPoints[i] = 450;
          }
          if (setPoints[i] < -400){
            setPoints[i] = -400;
          }
          inString = inString.substring(index+1,inString.length());
          dummy = "";
        }
        inString = "";// clear the string for new input:
        charCount = 0;
        //updateState();
//        displayState();
      }
      else{
        inString = "";// clear the string for new input:
        charCount = 0;
      }
    }
  }
}

//////////////////////////
////ENCODER HANDLING//////
//////////////////////////
void motor1Interrupt() {// pins 26 & 27, PD1 & PD2
  static uint8_t m1_enc_val = 0;
  m1_enc_val = m1_enc_val << 2;
  m1_enc_val = m1_enc_val | (((PIOD->PIO_PDSR & 0x02))|((PIOD->PIO_PDSR & 0x04) >> 2));
  motor1Count = motor1Count - lookup_table[m1_enc_val & 0b1111];
}

void motor2Interrupt() {// pins 28 & 29, PD3 & PD6
  static uint8_t m2_enc_val = 0;
  m2_enc_val = m2_enc_val << 2;
  m2_enc_val = m2_enc_val | (((PIOD->PIO_PDSR & 0x08) >> 2)|((PIOD->PIO_PDSR & 0x40) >> 6));
  motor2Count = motor2Count - lookup_table[m2_enc_val & 0b1111];
}

void motor3Interrupt() { // pins 30 & 31, PD9 & PA7
```

```
  static uint8_t m3_enc_val = 0;
  m3_enc_val = m3_enc_val << 2;
  m3_enc_val = m3_enc_val | (((PIOD->PIO_PDSR & 0x0200) >> 8)|((PIOA->PIO_PDSR & 0x80)>>7));
  motor3Count = motor3Count - lookup_table[m3_enc_val & 0b1111];
}

void motor4Interrupt() {// pins 32 & 33, PD10 & PC1
  static uint8_t m4_enc_val = 0;
  m4_enc_val = m4_enc_val << 2;
  m4_enc_val = m4_enc_val | (((PIOD->PIO_PDSR & 0x0400)>>9)|((PIOC->PIO_PDSR & 0x02) >> 1));
  motor4Count = motor4Count - lookup_table[m4_enc_val & 0b1111];
}

void motor5Interrupt() {// pins 34 & 35, PC2 & PC3
  static uint8_t m5_enc_val = 0;
  m5_enc_val = m5_enc_val << 2;
  m5_enc_val = m5_enc_val | (((PIOC->PIO_PDSR & 0x04) >> 1)|((PIOC->PIO_PDSR & 0x08) >> 3));
  motor5Count = motor5Count - lookup_table[m5_enc_val & 0b1111];
}

void motor6Interrupt() { // pins 36 & 37, PC4 & PC5
  static uint8_t m6_enc_val = 0;
  m6_enc_val = m6_enc_val << 2;
  m6_enc_val = m6_enc_val | (((PIOC->PIO_PDSR & 0x10) >> 3)|((PIOC->PIO_PDSR & 0x20) >> 5));
  motor6Count = motor6Count - lookup_table[m6_enc_val & 0b1111];
}

void motor7Interrupt() {// pins 38 & 39, PC6 & PC7
  static uint8_t m7_enc_val = 0;
  m7_enc_val = m7_enc_val << 2;
  m7_enc_val = m7_enc_val | (((PIOC->PIO_PDSR & 0x40) >> 5)|((PIOC->PIO_PDSR & 0x80) >> 7));
  motor7Count = motor7Count - lookup_table[m7_enc_val & 0b1111];
}

void motor8Interrupt() {// pins 40 & 41, PC8 & PC9
  static uint8_t m8_enc_val = 0;
  m8_enc_val = m8_enc_val << 2;
  m8_enc_val = m8_enc_val | (((PIOC->PIO_PDSR & 0x100) >> 7)|((PIOC->PIO_PDSR & 0x200) >> 9));
  motor8Count = motor8Count - lookup_table[m8_enc_val & 0b1111];
}

//void indexInterrupt() {
//  motor5Count = 0;
//  }
```

B.2 C++ Function File

```
    #if (ARDUINO >= 100)
#include "Arduino.h"
#else
#include "WProgram.h"
#endif
#include <Wire.h>

#include "continuumDue_FullOctWeb.h"
#include <math.h>

void DCMotor::run(uint8_t cmd) {
switch (cmd) {
case FORWARD:
this->setPin(dirPin, HIGH);  // take low first to avoid 'break'
digitalWrite(dirPin, HIGH);
//Serial.print(dirOutpin);
```

```
//Serial.println("FORWARD");
break;
case BACKWARD:  // take low first to avoid 'break'
this->setPin(dirPin, LOW);
digitalWrite(dirPin, LOW);
//Serial.println("BACKWARD");
break;
case RELEASE:
this->setPin(dirPin, LOW);
break;
}
}

DCMotor::DCMotor(void) {
PWMpin = dirPin = m_pgain = m_dgain = 0;
count = 0;
m_prevError = 0x80000000;
}

DCMotor::DCMotor(uint8_t pwm, uint8_t dir, uint32_t pgain, uint32_t dgain, volatile long* motorCount
{
PWMpin = pwm;
dirPin = dir;
m_pgain = pgain;
m_dgain = dgain;
count = motorCount;
m_prevError = 0x80000000;
}

void DCMotor::setPin(uint8_t pin, boolean value) {
digitalWrite(pin, value);
```

```
}

double DCMotor::getLength() {
double countPerRev = 1680.0;
double length = (double)(*(this->count)/countPerRev)*(2*M_PI*0.0102);
return length;
}

void DCMotor::setSpeed(uint8_t speed) {
//Serial.print("S  ");
//Serial.print(speed);
//Serial.print("\t");
analogWrite(PWMpin, speed);
}

void DCMotor::update(int32_t errorL, uint8_t use_force_control)
{
long int vel;
int correction;
int setpoint = errorL + (*(this->count));
    int8_t scalar = (setpoint > 0)?1:-1;
//Serial.print(errorL);
//Serial.print("\t");
if (use_force_control == 1)
{
correction = this->update_force_backbone();
     correction = correction * scalar; // will delete
}
if(use_force_control == 2)
   {
     correction = this->update_force_fabric();//+ this->update_force_backbone();
     correction = correction * scalar; // will delet
   }
else
{
correction = 0;
}



if (m_prevError != 0x80000000)
{
vel = (long int)(errorL * m_pgain + (errorL - m_prevError) * m_dgain) >> 8;
//Serial.print(vel);
//Serial.print("\t");

//m_pos += velocity;
if (vel > MOTOR_MAX)
{
vel = MOTOR_MAX + correction;
}
else if (vel < MOTOR_MIN)
{
vel = MOTOR_MIN + correction;
}
}



 //Serial.println(vel);
//Serial.print("\t");
if (vel >= 0) {
run(BACKWARD);
setSpeed((uint8_t)abs(vel));
```

```
}
else {
run(FORWARD);
setSpeed((uint8_t)abs(vel));
}

m_prevError = errorL;
}

int DCMotor:: update_force_fabric(){
double s=0.45; //needs to be updated
    //double _tendon_distance=0.00784; // needs to be updated v
double _tendon_distance=0.00697;
    double l_1 = s + this->getLength();
    double u=0.0;
    double v = 0.0;
    v = (s - l_1) / _tendon_distance;
    // _pwm_adjustment = pwm_adjustment;

    double theta = sqrt(pow(u,2)+pow(v,2));
    double x = -(((cos(theta)-1)/pow(theta,2))*s*v);
        //double y = (((cos(theta)-1)/pow(theta,2))*_s_u_v_parameters.s*_s_u_v_parameters.u);
    double z = (sin(theta)/theta)*s;
    double delz = s - z;
    double kb = 0.64;
    double Fb = kb*sqrt(pow(x,2)+pow(delz,2));
    double Juv = -(1.0/_tendon_distance);
    double tension = Fb*Juv;
    double torque = -(0.0102*tension);
    double current = 2.833*torque;
    double resistance = 2.3;
    double voltage = current*resistance;
    int pulse = (int)((voltage*255.0)/12.0);

    if (pulse>255)
        pulse=255;
    else
        if (pulse<-255)
        pulse=-255;

    return pulse;
}

int DCMotor:: update_force_backbone(){
    double s=0.45; //needs to be updated
//    //double _tendon_distance=0.00784; // needs to be updated v
//    double _tendon_distance=0.00697;
//    double l_1 = s + this->getLength();
//    double u=0.0;
//    double v = 0.0;
//    v = (s - l_1) / _tendon_distance;
//    // _pwm_adjustment = pwm_adjustment;
//
//    double theta = sqrt(pow(u,2)+pow(v,2));
    double pwm_theta = 0.1207*abs((*(this->count)));
    int pulse = (int)pwm_theta;
//    double x = -(((cos(theta)-1)/pow(theta,2))*s*v);
//        //double y = (((cos(theta)-1)/pow(theta,2))*_s_u_v_parameters.s*_s_u_v_parameters.u);
//    double z = (sin(theta)/theta)*s;
//    double delz = s - z;
//    double kb = 0.64;
//    double Fb = kb*sqrt(pow(x,2)+pow(delz,2));
//    double Juv = -(1.0/_tendon_distance);
//    double tension = Fb*Juv;
//    double torque = -(0.0102*tension);
//    double current = 2.833*torque;
//    double resistance = 2.3;
//    double voltage = current*resistance;
//    int pulse = (int)((voltage*255.0)/12.0);

    if (pulse>255)
        pulse=255;
    else
        if (pulse<-255)
        pulse=-255;

    return pulse;
}
```

B.3 Header File

```
    #ifndef _continuumDue_FullOctWeb_h_
#define _continuumDue_FullOctWeb_h_

#include <inttypes.h>
#include <Wire.h>
#include "Arduino.h"

#define FORWARD  1
#define BACKWARD 2
#define RELEASE  3

#define RIGHTM 1
#define LEFTM  2
```

```
#define LEFT_AXIS       1
#define RIGHT_AXIS      0

#define MOTOR_MAX       30
#define MOTOR_MIN       -30

//m(otor)Enc(oder)_{A/B}{motornumber}
#define mEnc_A1 26
#define mEnc_B1 27
#define mEnc_A2 28 //26
#define mEnc_B2 29 //27
#define mEnc_A3 30 //28
#define mEnc_B3 31 //29
#define mEnc_A4 32 //30
#define mEnc_B4 33 //31
#define mEnc_A5 34 //32
#define mEnc_B5 35 //33
#define mEnc_A6 36 //34
#define mEnc_B6 37 //35
#define mEnc_A7 38 //36
#define mEnc_B7 39 //37
#define mEnc_A8 40 //38
#define mEnc_B8 41 //39
//#define mEnc_A9 38 //40
//#define mEnc_B9 39 //41

//Motor pwm pins for analog write
//pwmM(otor){motorNumber}
#define pwmM1  3
#define pwmM2  4
```

```
#define pwmM3  5

#define pwmM4  6

#define pwmM5  7

#define pwmM6  8

#define pwmM7  9

#define pwmM8  10

//#define pwmM9 10


//direction pins for digital write to motor driver

//dir(ection)M(otor){motorNumber}

#define dirM1 69

#define dirM2 68

#define dirM3 67

#define dirM4 66

#define dirM5 65

#define dirM6 64

#define dirM7 63

#define dirM8 62

//#define dirM9 46


//essential for checking encoders

const int8_t lookup_table[] = { 0,-1,1,0,1,0,0,-1,-1,0,0,1,0,1,-1,0 };


class DCMotor

{

public:

volatile long* count;

DCMotor(void);

DCMotor(uint8_t pwm, uint8_t dir, uint32_t pgain, uint32_t dgain, volatile long* motorCount);

void setSpeed(uint8_t);

void run(uint8_t);
void update(int32_t error, uint8_t use_force_control);

double getLength();

    int update_force_fabric();

int update_force_backbone();


private:

uint8_t dirPin = 2;

uint8_t PWMpin=3;

uint32_t m_pgain = 0;

uint32_t m_dgain = 0;

int32_t m_prevError = 0x80000000;

void setPin(uint8_t pin, boolean val);

};


#endif
```

Bibliography

[1] Thomas F Allen, Levi Rupert, Timothy R Duggan, Gabriel Hein, and Kevin Albert. Closed-form non-singular constant-curvature continuum manipulator kinematics. In *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*, pages 410–416. IEEE, 2020.

[2] D. D. Arachchige, Y. Chen, I. D. Walker, and I. S. Godage. A novel variable stiffness soft robotic gripper. In *IEEE International Conference on Automation Science and Engineering (CASE)*, page 2222–2227, 2021.

[3] L. Beddow, H. Wurdemann, and D. Kanoulas. A caging inspired gripper using flexible fingers and a movable palm. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7195–7200, 2021.

[4] A. Bicchi. Hands for dexterous manipulation and robust grasping: A difficult road toward simplicity. *IEEE Transactions on Robotics and Automation*, 16(6):652–662, 2000.

[5] J. Burgner-Kars, D.C. Rucker, and H. Choset. Continuum robots for medical applications: A survey. *IEEE Transactions on Robotics*, 31(6):1261–1280, 2015.

[6] S. Cai, C. Tang, L. Pan, G. Bao, W. Bai, and Q. Yang. Pneumatic webbed soft gripper for unstructured grasping. *International Journal of Agricultural and Biological Engineering*, 14(4):145–151, 2021.

[7] A. Carambia, C.G. Frazelle, and I.D. Walker. Continuum robot grippers exploiting soft webs. In *IEEE International Conference on Automation Sciences and Engineering (CASE) (in review)*, 2022.

[8] K.C. Galloway, K.P. Becker, B. Phillips, J. Kirby, S. Licht, D. Tchernov, R.J. Wood, and D.F. Gruber. Soft robotic grippers for biological sampling on deep reefs. *Soft Robotics*, 3(1):23–33, 2016.

[9] P. Gonthina. *Novel Parallel Continuum Robot Grippers and their Modeling.* M.S. Thesis, Department of Electrical and Computer Engineering, Clemson University, 2018.

[10] Phanideep S Gonthina, Apoorva D Kapadia, Isuru S Godage, and Ian D Walker. Modeling variable curvature parallel continuum robots using euler curves. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1679–1685. IEEE, 2019.

[11] V. Ho and S. Hirai. Design and analysis of a soft-fingered hand with contact feedback. *IEEE Robotics and Automation Letters*, 2(2):491–498, 2017.

[12] H.J. Hoving and B.H. Robison. Vampire squid: detritivores in the oxygen minimum zone. *Proceedings of the Royal Society B: Biological Sciences*, 279.1747:4559–4567, 2012.

[13] J. Hughes, U. Culha, F. Giardina, F. Guenther, A. Rosendo, and F. Iida. Soft manipulators and grippers: a review. *Frontiers in Robotics and AI*, 3:69, 2016.

[14] I. Kato. *Mechanical Hands Illustrated.* Survey Japan, Reprinted from 1977 Japanese edition, 1982.

[15] D.M. Lane, J.B.C. Davies, G. Robinson, D.J. O'Brien, J. Sneddon, E. Seaton, and A. Elfstrom. The amadeus dextrous subsea hand: Design, modeling, and sensor processing. *IEEE Journal of Oceanic Engineering*, 24(1):96–111, 1999.

[16] J. Li, Z. Teng, J. Xiao, A. Kapadia, A. Bartow, and I.D. Walker. Autonomous continuum grasping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4569–4576, 2013.

[17] J. Li and J. Xiao. Determining "grasping" configurations for a spatial continuum manipulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4207–4214, 2011.

[18] Y. Li and I. Kao. A review of modeling of soft-contact fingers and stiffness control for dextrous manipulation in robotics. In *IEEE International Conference on Robotics and Automation*, pages 3055–3060, 2001.

[19] Y. Liu, Q. Bi, and Y. Li. Development of a bio-inspired soft robotic gripper based on tensegrity structures. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7398–7403, 2021.

[20] C.P. Mark and S. Kamath. Review of active space debris removal methods. *Space Policy*, 47:194–206, 2019.

[21] C. Melchiorri and M. Kaneko. Robot hands. In *Springer Handbook of Robotics, 2nd Ed., B. Siciliano and O. Khatib, Eds*, volume Chapter 19, pages 463–480, 2016.

[22] T. Puiu. Colugo (flying lemur): the most accomplished and cutest mammalian glider. *ZME Science*, 2015.

[23] G.C. Robson. Notes on the cephalopoda.—iv. on octopus ægina, gray; with remarks on the systematic value of the octopod web. *Journal of Natural History*, 1(5):641–646, 1928.

[24] D. Sahrhage and J. Lundbeck. *A history of fishing.* "Springer Science & Business Media", 2012.

[25] J.K. Salisbury. Active stiffness control of a manipulator in cartesian coordinates. In *IEEE Conference on Decision and Control*, pages 95–100, 1980.

[26] Michael Sfakiotakis, Asimina Kazakidi, Avgousta Chatzidaki, Theodoros Evdaimon, and Dimitris P Tsakiris. Multi-arm robotic swimming with octopus-inspired compliant web. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 302–308. IEEE, 2014.

[27] N.R. Sinatra, C.B. Teeple, D.M. Vogt, K.K. Parker, D.F. Gruber, and R.J. Wood. Ultragentle manipulation of delicate structures using a soft robot gripper. *Science Robotics*, 4:1–11, 2019.

[28] F. J. Tarasoff, A. Bisaillon, J. Pierard, and A.P. Whitt. Locomotory patterns and external morphology of the river otter, sea otter, and harp seal (mammalia). *Canadian Journal of Zoology*, 50(7):915–929, 1972.

[29] D. Trivedi, C. D. Rahn, W. M. Kier, and I. D. Walker. Soft robotics: Biological inspiration, state of the art, and future research. *Applied Bionics and Biomechanics*, 5(3):99–117, 12 2008.

[30] I. D. Walker, H. Choset, and G. Chirikjian. Snake-like and continuum robots. In *Springer Handbook of Robotics*, chapter 20, pages 481–498. Springer, 2nd edition, 2016.

[31] R. J. Webster and B. A. Jones. Design and kinematic modeling of constant curvature continuum robots: A review. *International Journal of Robotics Research*, 29(13):1661–1683, 2010.

[32] Michael B Wooten and Ian D Walker. Vine-inspired continuum tendril robots and circumnutations. *Robotics*, 7(3):58, 2018.