August 2020

Novel Soft Palmar Gripper for Chicken Handling

Henry Rutland
Clemson University, hrutlan@g.clemson.edu

Novel Soft Palmar Gripper for Raw Chicken Handling

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
Henry G. Rutland IV
August 2020

Accepted by:
Dr. Ian Walker, Committee Chair
Dr. Adam Hoover
Dr. Apoorva Kapadia

Abstract

This thesis describes the development of a novel concept for a soft gripper with pneumatically articulated fingers and palm used in the pick and place operations of raw chicken to aid with the shortage of human workers that currently perform this task. The gripper was attached to an industrial robot and tested by picking raw chicken parts moving along a conveyor and placing those parts into trays. Four different parts were tested over 250 times each for a total of more than 1000 trials. Over the course of these trials the gripper saw an overall success rate of 63.57%. While this is low, promising results occurred when the pressure in the palm was roughly doubled, yielding success rates around 95%. However, these pressures led to the palm bursting. With a greater investigation in materials and design, a more robust gripper could be achieved.

Dedication

My chosen Katie,

You entered my life a few months before this project and stuck by me to the very end of it. You pushed me forward when I struggled and helped put my thoughts into words. Most of all, you became my beautiful wife, and I get to love you forever.

Acknowledgments

Jesus is my God and savior and without whom my efforts on this project would be nothing. He gave me the strength and courage to carry on when I fell and guided me to the ideas that led to the results found in this thesis.

I am grateful for Dr. Ian Walker and his belief in my abilities and appreciate the opportunity to work in his lab on this project. As my advisor and committee chair, he gave me valuable recommendations and directed the project to its current state.

Thanks to Dr. Adam Hoover and Dr. Apoorva Kapadia for accepting my invitation to be my other two committee members and for the instruction along the course of my education at Clemson University. They provided answers to my questions and tips that helped me complete the project.

Thanks to Chase Frazelle for brainstorming and troubleshooting the difficulties that arose and for spending late nights in the lab helping me.

Thanks to Clemson University for the many opportunities it provides.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This century has seen great technological advancements, especially in the development of robots and their abilities. In particular, the rise of soft and continuum robotics, which feature compliant and conformable bodies, has expanded the capabilities in robotic applications and possible configurations that robots can achieve, beyond those achievable by traditional rigid-link robots [22], [23]. Combining traditional rigid-link and continuum (compliant continuous backbone) technologies also produces interesting results, for example the creation of robotic tongues functioning as grippers for conventional rigid-link robots [4]. Developments in materials and chemistry have led to even more advancements in robotics with the creation of soft, deformable body robotics [25], [19].

These breakthroughs have caused a tremendous surge in adoption of robotics across a variety of industries. However, the food industry has seen a much slower rate of robot integration. Few robots have been introduced to the meat industry [14], [20], and were directed primarily at meat cutting [12]. Early examples of food handling robots were unreliable and impractical [13], especially for dealing with meats. Other robots were limited to specifically shaped and sized meats, such as pork chops [5].

Robots with suction cups were also explored for picking up pork bellies [10], yet they tended to leave unwanted marks. While the deployment of robots in the meat industry is in its infancy, the need for raw chicken handling solutions is continuing to grow.

1.1 Problem Considered

Chicken processing plants have slowly been adding more automated processes to their lines, but still have one bottleneck that has yet to be adequately resolved. Packaging of raw chicken requires people in long coats, hair nets, gloves, and boots standing in front of a conveyor to pick up parts as they pass by and place them into trays before they can be wrapped in plastic. This particular part of the processing plant is one of the last to be automated and requires many manual workers in order to reach the throughput to keep the plant productive. Unfortunately, being a chicken packager is not the most rewarding job, and plant managers struggle to keep enough workers on the line. The number of trays per day is highly dependent on the crew available on a given day. If an automated solution was found, the workers with these jobs could be better utilized in other parts of the plant, and plant managers would be able to better forecast how many trays they can output.

1.2 Related Work

Several companies have attempted to tackle the task of picking and placing poultry. Soft Robotics Inc. has developed a hard-palm gripper with pneumatically actuated fingers. This gripper works well at grasping a variety of objects; however, those objects have to be roughly the same size. The Soft Robotics Inc. gripper's modular design allows it to be reconfigured based on the application. Soft Robotics Inc. gripper has successfully handled breasts [7], legs [9], and drumsticks [8]. Marel's poultry division has taken another approach, and has developed one gripper for cradling pieces from a conveyor and another gripper specifically for handling drumsticks [11]. Both of these grippers feature mostly hard components and are unable to grasp a range of part sizes. Additionally, Omron has also shown some success using an Adept Quattro Robot and has developed two poultry handling grippers. One is a stainless steel parallel gripper that slides under the sides of chicken breasts [17] and another is a drumstick-shaped suction cup for handling drumsticks [18].

1.3 A Helping Hand

The gripper introduced and developed in this thesis was a single tool that can handle a significantly wider variability in the parts that it can grasp and pick up compared to the above previous work. It was inspired by the human hand, specifically fingers connected by a morphable palm. Given the ability to change the effective size of the palm, the fingers had a better grasp. Smaller objects benefit from the palm being more closed. Consider picking up a pencil with a human hand. The hand's fingers are nearly parallel and apply pressure through the fingertips. Larger objects, such as a basketball, require a human hand to stretch out wide in order to "palm" it. The gripper introduced in this thesis had the ability to perform both of these operations. It was manufactured using two types of Smooth-On platinum cure silicone and was pneumatically driven. Not only was the gripper soft and compliant, but it was made from food-grade materials that are easy to clean. These traits are important for handling raw poultry in order to not damage the product and retain sanitary operation.

1.4 Thesis Overview

The chapters that follow describe the journey to the creation of the new gripper, first discussing the hardware and software required to test robot grippers as well as early attempts with gripper concepts and their effectiveness in chapter two. Once the best gripper concept was selected from the early experiments, a systematic procedure for creating the gripper had to be determined, and this process is described in chapter three. Chapter four presents the results of testing the gripper through many iterations of grasping several different types of chicken parts, and chapter five contains conclusions reached about the project as a whole and suggestions for future work.

Chapter 2

System Setup

2.1 Hardware

The empirical system used in this work consisted of a KUKA KR6 R700 sixx robot attached to a rolling cart with a Dorner 2200 series conveyor bolted to one side. A USB camera was mounted above the starting edge of the conveyor (figure 2.1). An Arduino Uno acted as the gripper controller, sending a signal through a twelve bit DAC that wass processed through a non-inverting amplifier with a gain of two (figure 2.2). This signal was then sent to a pressure regulator that provided air to the gripper from a large air compressor and tank. The pressure regulator used was a SMC ITV 1050-31N1N4. The first two gripper designs described below used one pressure regulator since they were fully actuated by a single pressure; the third and fourth designs required two pressure regulators, one for the variable palm and another for the fingers. A laptop received data from the USB webcam and then commanded the robot arm to move towards the chicken part and informed the gripper when to grasp it.

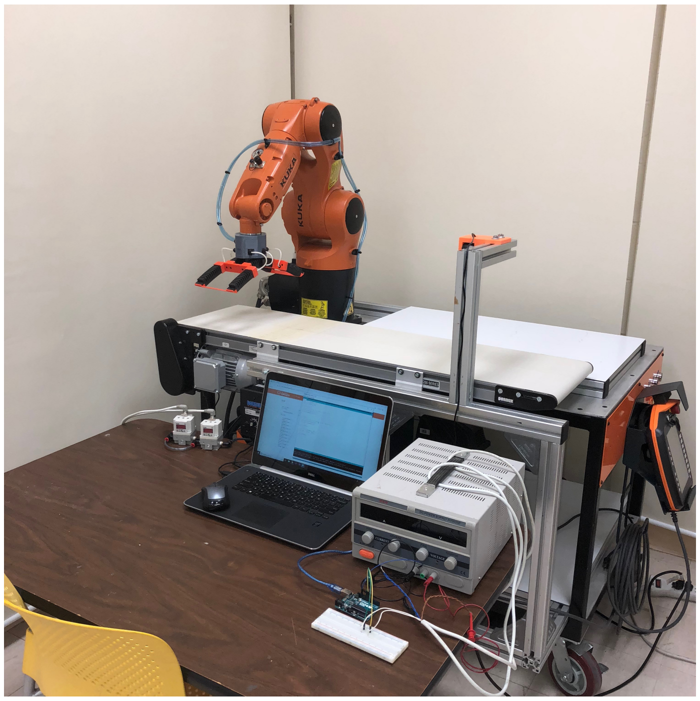
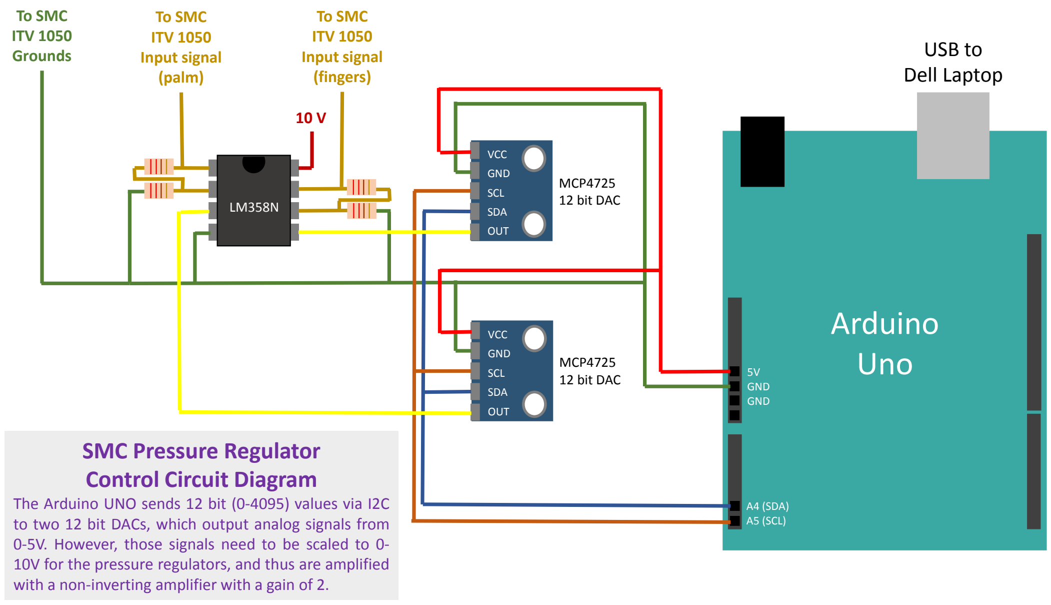


Figure 2.1: Hardware System Setup



SMC Pressure Regulator Control Circuit Diagram

The Arduino UNO sends 12 bit (0-4095) values via I2C to two 12 bit DACs, which output analog signals from 0-5V. However, those signals need to be scaled to 0-10V for the pressure regulators, and thus are amplified with a non-inverting amplifier with a gain of 2.

Figure 2.2: Arduino Pressure Regulator Control Circuit

2.2 Software

In order to move the robot, multiple programs ran simultaneously. The first is kukavarproxy.exe, which was located in the startup folder of the Windows machine that ran in the KUKA KRC4 Compact robot controller. When the robot was booted up, this program automatically ran and functioned as a server. Its purpose was to receive the joint positions and speeds from a client and update them in the robot controller. The second piece of software was written in the KUKA robot language, KRL, and continuously looped on the KUKA teach pendant. This software monitored the updated joint positions and speeds and moved the robot according to these variables. The third software module used the RoboDK library written in Python to create a client. The whole program was written in Python, thus eliminating the overhead of learning KRL. Instructions for setting up this type of system were provided by RoboDK [15], [16].

The gripper control code (appendix A) used in the Arduino UNO was written in C. It continuously monitored a serial line from the laptop that sent the appropriate pressure values, and the Arduino then sent the signal to the pressure regulators. Also, gripper test code (appendix B) was written in Python to verify that the control code worked via the serial connection, while providing a way to tune the palm and finger pressures.

The system software (appendix C) was also written in Python and utilized the Multiprocessing and Queue libraries to transfer data between processes, OpenCV-Python for vision, and RoboDK and Robolink for robot control. Five different processes ran in parallel: the parent process, the planner process, the vision process, the KUKA process, and the Arduino process. A diagram of the system process structure is shown in figure 2.3, which outlines how the different processes were connected.

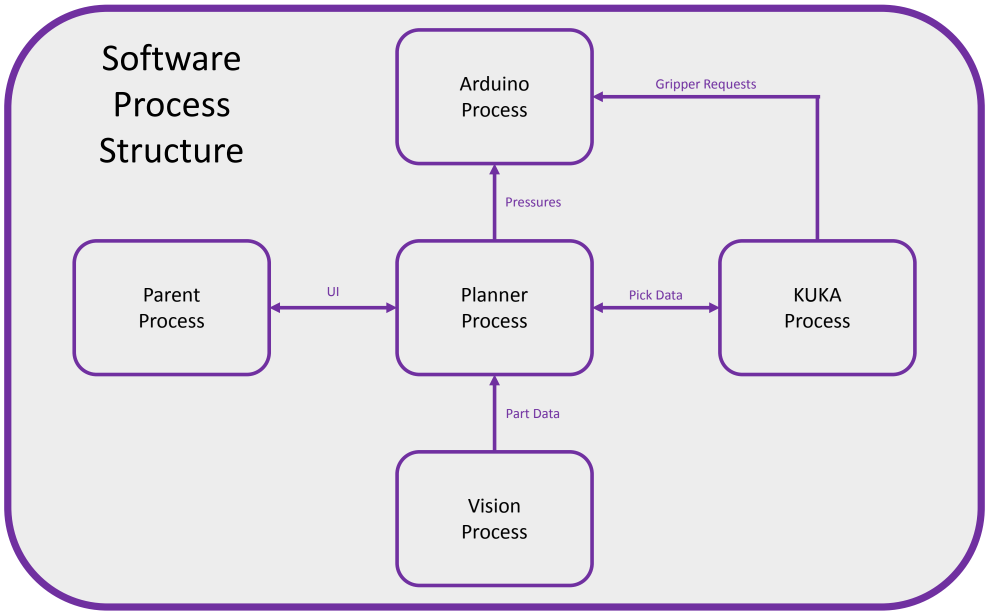


Figure 2.3: Software Process Structure

The parent process created the other processes and the communication lines between those processes via queues. Its purpose was to set processes up and help them shut back down properly. As mentioned earlier, the vision processes used the OpenCV-Python library to detect the raw chicken on the conveyor via color filtering with HSV ranges, find each part's (x,y) position with respect to the KUKA base, find the orientation of each part by fitting an ellipse, and label each part with a timestamp to keep track of it as it is passed through a queue to the planning process. Figure 2.4 demonstrates an example of how the vision system worked with slight alterations. The blue band across the middle was the detection band and was used to improve the processing speed. The band is perpendicular to the direction of travel of the conveyor belt. Only parts with their centroids, the white circles, contained within the band were selected for calculation of their orientations from the fitted, green ellipses.

While the parts passed through the detection band, a red rectangle was fit around each part to help eliminate part duplication. Each red rectangle was saved

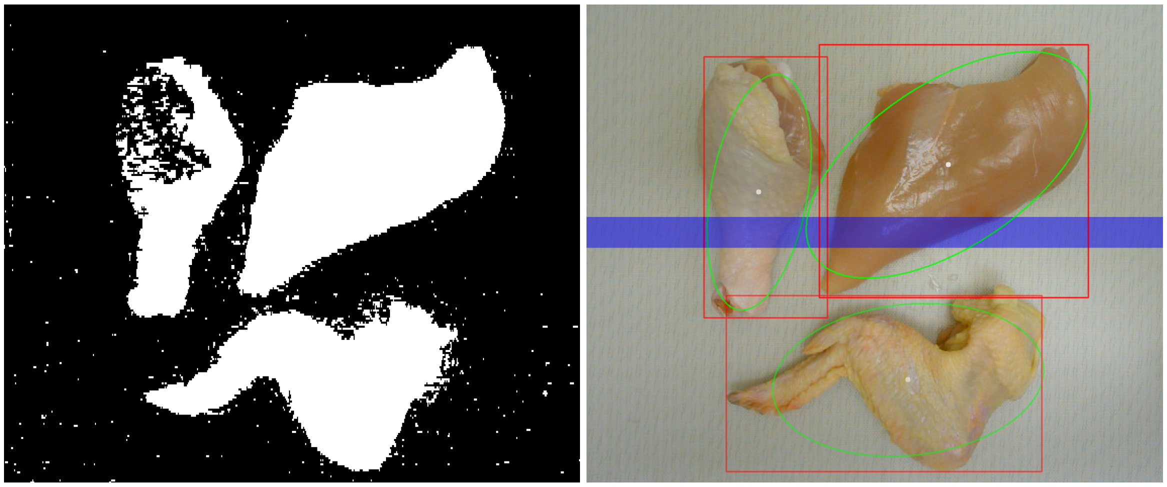

(a) Color Filtered Image (b) Display of Position and Orientation

Figure 2.4: Demonstration of Image Processing

for the next frame taken by the camera. If a centroid in the next frame fell within a previous red rectangle, it was assumed that the part was already recorded in a past frame. The planning process monitored the incoming queue of parts and determined whether the parts were within a reachable range on the conveyor. If they were in picking distance, the planning process added them to the outgoing list. Parts that made it to the outgoing list were evaluated to decide if the KUKA could intercept them before they reached the end of the conveyor. Once a part was thus found to be feasible, the pick position was sent to the KUKA process, and the gripper pressures were sent to the Arduino process. The KUKA process then moved the robot to the specified pick position and notified the Arduino process when to close the gripper. The gripper then grasped the part and moved it to its destination before the KUKA process again notified the Arduino process to open the gripper. Then the robot would then reset to its standby position and wait for the next part.

2.3 Initial Experiments

The first of the grippers considered in this work was created using two common kitchen utensils: a non-slotted spatula and tongs. The spatula was used to slide under the chicken, while the tongs gently grasped from above. This gripper was made from a spatula purchased at Walmart. Its handle was cut off because it was not necessary for the gripper, and holes were added to attach the spatula to a 3D printed part to mount it to the KUKA. A 3D-printed upper bracket and hinge was attached to this mounting part, with a single-action pneumatic cylinder from Grainger attached to the bracket and the rotating jaws of the tongs. The tongs, or jaws, attached to the hinge and rotated down when pressure was applied to the pneumatic cylinder and rotated back up when the pressure was released. Compliant teeth were a part of the tong design in order to help absorb any pressure that could damage the chicken. See figure 2.5.

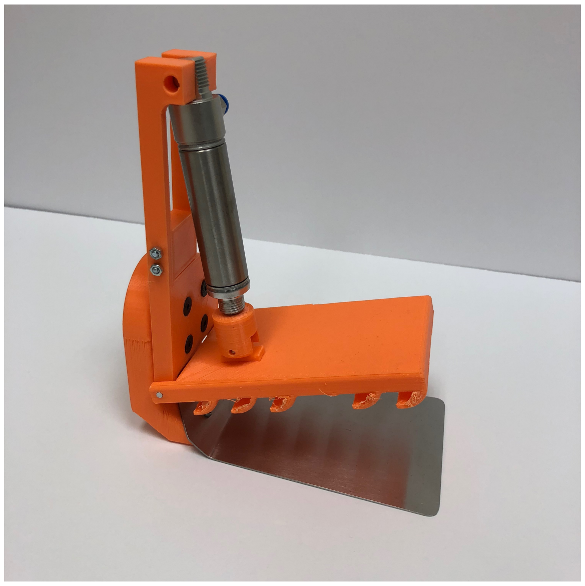


Figure 2.5: Spatula and Tongs Hybrid Gripper

This gripper was tested picking up only chicken tenders and had limited success. During most of the trials, the spatula pushed the chicken out of the way rather than scooping underneath it. The spatula also needed space in front of the chicken to be able to prepare for scooping, which required a larger footprint on the conveyor and meant that the parts cannot be placed close together.

The next gripper considered herein was inspired by the concept of directly using artificial muscles, and had three fingers with a fixed (rigid) palm. Each finger was a short McKibben muscle, which is a rubber tube surrounded by a mesh with a thread sewn down one side. As air is inserted into the tube, the mesh prevents the tube from ballooning, and since the thread is inextensible, this causes the muscle to bend in the direction of the thread. The fixed palm was 3D-printed in PLA, and each finger was fixed 120 degrees from the other two with the threads facing the center of the palm. When a single pressure was applied to all three fingers simultaneously, the gripper closed to grasp an object. The tip of each finger featured a hose clamp to seal the air chambers. However, these clamps could potentially damage the chicken and made it an unattractive tool for the application. Additionally, the mesh presented problems in maintaining sanitary conditions in grasping meat, and therefore the gripper was not tested with chicken. This gripper was used to pick up other foods, such as hard boiled eggs and tomatoes, and was featured in this other paper [6]. See figure 2.6.

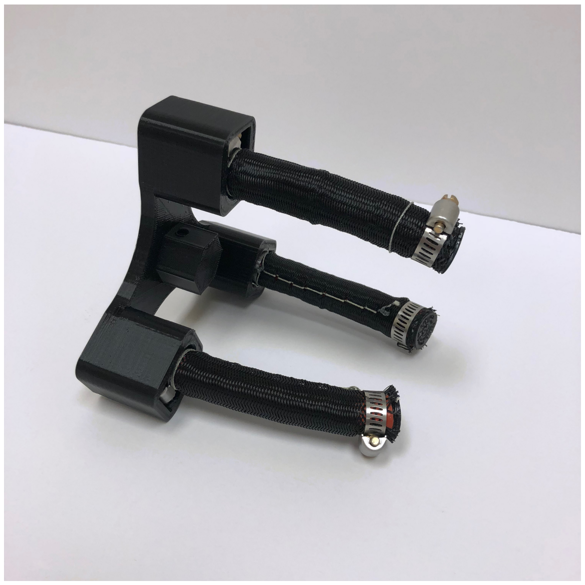


Figure 2.6: McKibben Muscle Three-Fingered Gripper

The tests with the initial finger-and-palm gripper indicated that the design was limited in what it could grasp. Only generally spherical objects were able to be picked up easily using the three fingers, and the design proved impractical for grasping long, thin items. With the fixed palm, only items that were small enough to fit within the opening of the fingers were able to be grasped. Experience with this design showed that a variable palm was desirable in order to have the wide range of motion needed to pick up variably sized items.

The third gripper design took inspiration from the human hand with four fingers separated by an articulated palm. It featured a 3D-printed modular design made of Ninjatek's Ninjaflex with Ninjatek Cheetah filament connection pieces, and consisted of two muscles in the palm. Due to the modular design, the gripper had the ability to have up to 6 fingers. The palm and finger muscles were a series of separated air chambers that were connected by a single airway that led to an inlet. In this gripper, the airway ran along an inextensible layer. When air pressure was applied to the inlet, the chambers expanded, pushing against each other, causing the muscles to bend in one direction. See figure 2.7.

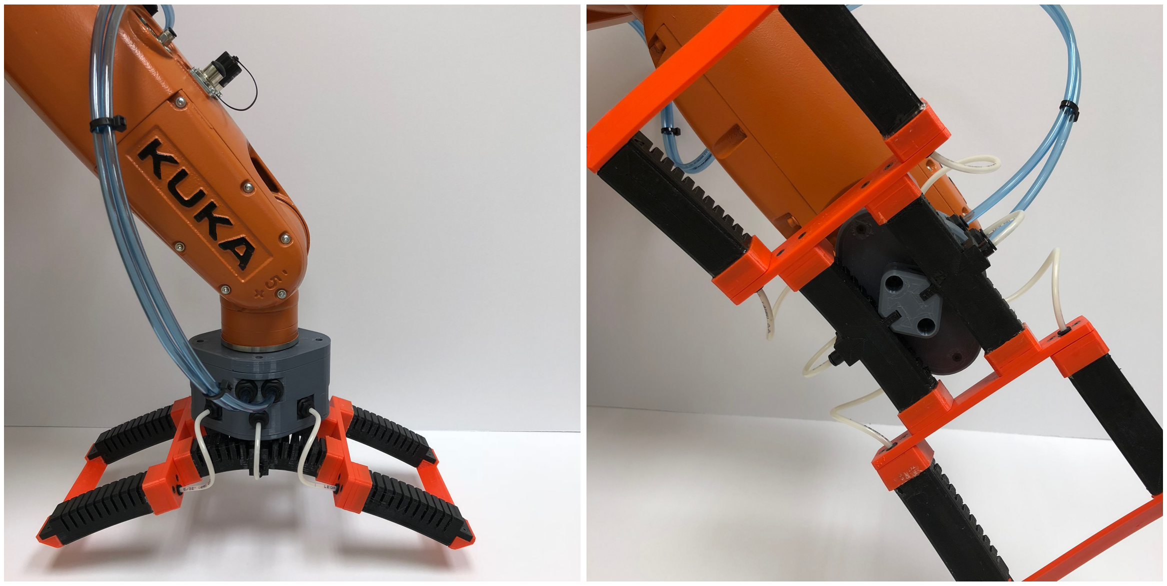
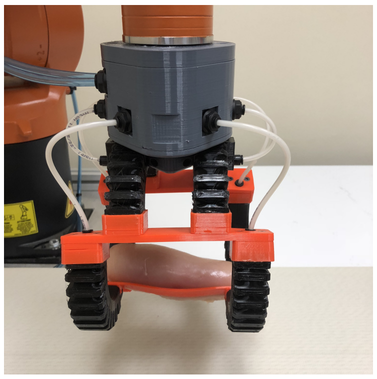
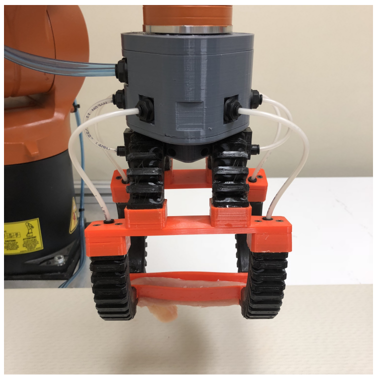
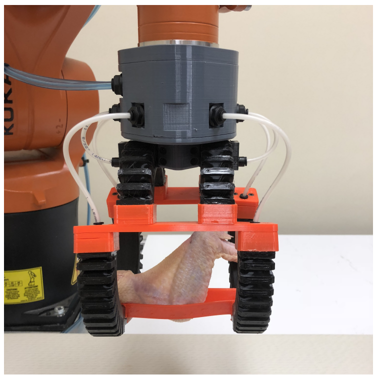


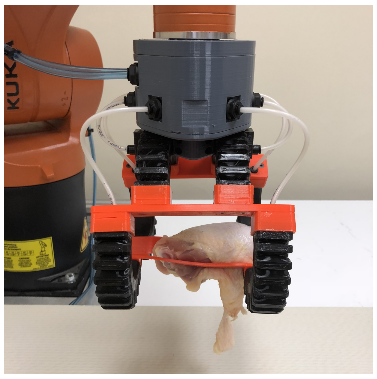
Figure 2.7: 3D Printed Gripper with Fingers and Articulated Palm

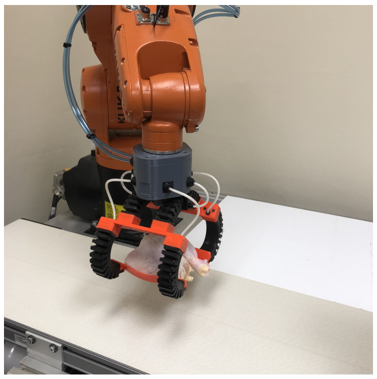
Initial trials were performed using a manually controlled prototype gripper grasping miscellaneous objects, such as white board erasers, circular silicone disks, individual McKibben muscles, and various sized balls to assess the gripper design. During these experiments, the gripper had the most consistent success in picking up the erasers and varying degrees of success with the other items. Following these screenings, manually controlled trials were conducted using chicken breasts, tenderloins, wings, thighs, leg quarters, and legs to test its grasp (figure 2.8). The gripper was successful in picking up and maintaining hold of each piece when beginning from a static position. The next experiment consisted of picking up pieces from the moving conveyor. These trials had limited success and the gripper was only able to consistently pick up chicken legs out of the pieces tested (figure 2.9). Due to the high pressure required, the gripper burst after a short amount of time. One likely reason for this was weakness at the 3D-printed seams.


(a) Breast


(b) Tenderloin


(c) Wing


(d) Thigh


(e) Leg Quarter

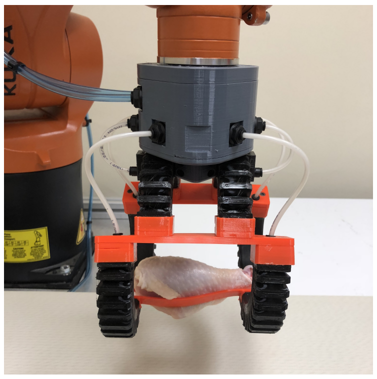

(f) Leg

Figure 2.8: Manual Controlled Grasping of Various Poultry Parts

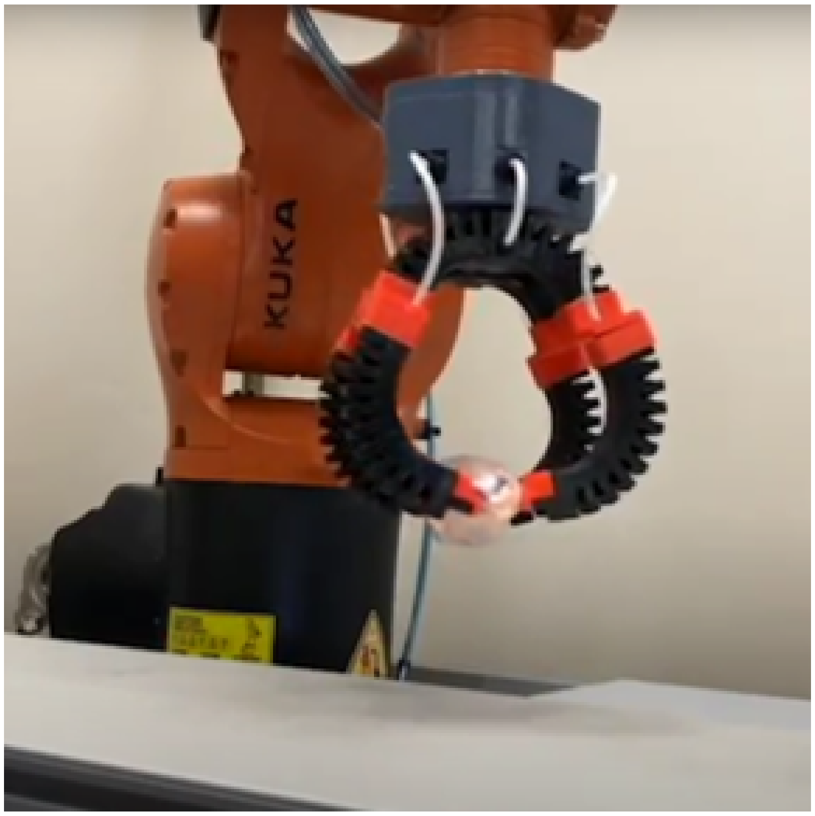


Figure 2.9: Leg Pick and Place Success

2.4 Refining the Gripper

The third gripper had the advantage of requiring limited space on the conveyor belt, unlike the first design that took up significant space on the belt in order for the spatula to scoop the pieces. It featured a variable grasp, meaning it was able to pick up a variety of items with different shapes and sizes, which the fixed palm design had been unable to do. The concept of using chambered muscles was a promising idea, but needed to be refined with a redesign using different materials. This concept led to the final design which was used in culminating experiments. Figure 2.10 shows the newly redesigned silicone gripper (fully detailed in the following chapter), and figure 2.11 shows it grasping the older 3D printed gripper.

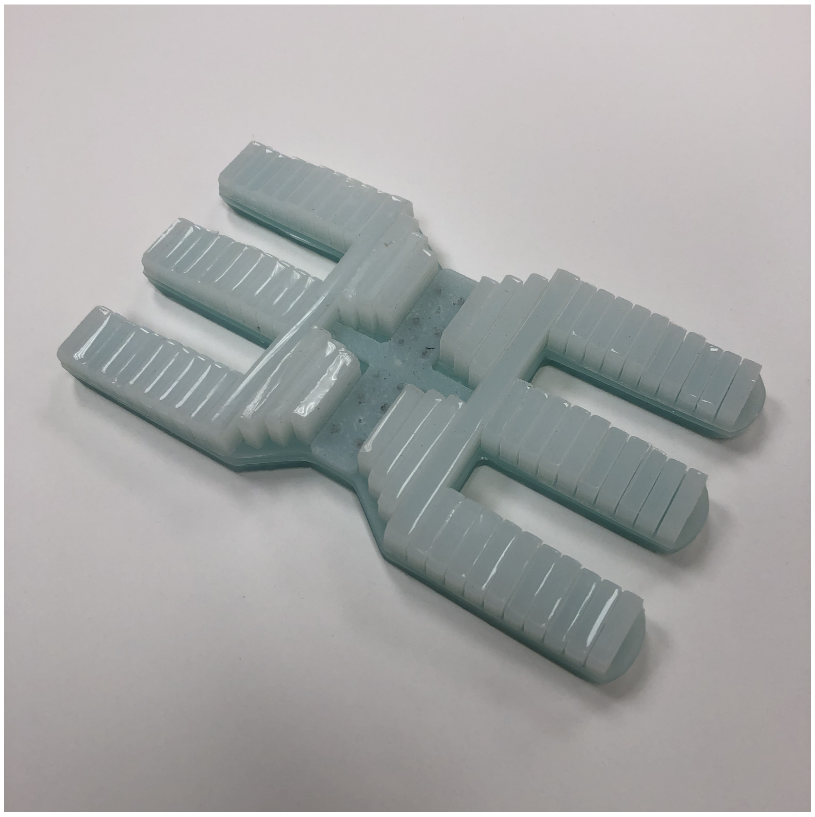


Figure 2.10: Redesigned Silicone Gripper

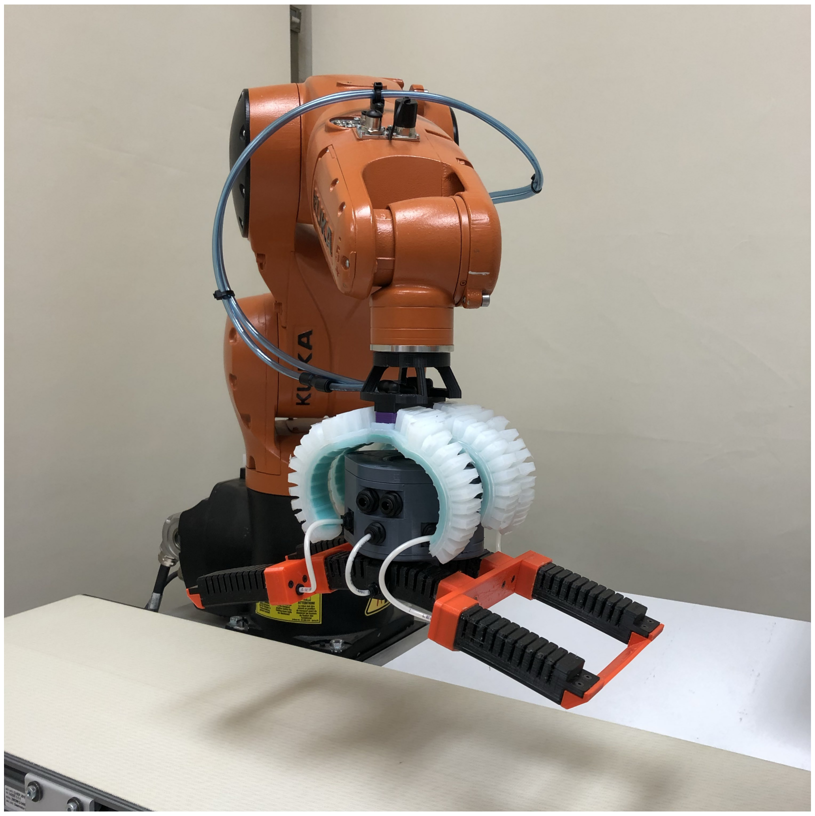


Figure 2.11: Redesigned Silicone Gripper Holding 3D Printed Gripper

Chapter 3

Fabricating a Soft Palmar Gripper

In this chapter, we discuss the design and manufacturing process used to create the soft gripper illustrated in figure 2.11. Based on the lessons learned from the three grippers discussed in chapter 2, we decided to explore a soft gripper made from silicone, with pneumatically actuated fingers and palm. The key innovation was in the inclusion of the soft actuated palm.

In the design of the pneumatic networks, or "PneuNets," that dispersed the pressurized air through the gripper's airways and chambers, we were greatly inspired by the Soft Robotics Toolkit [21]. This is an online resource that explains what PneuNets are, how they work, how they are created, and much more. Using the toolkit, the size, number, and spacing of the air chambers were calculated to play specific roles in the way in which the gripper functioned. Each of the six fingers was designed with smaller chambers closest to the fingertips and increased in two sizes by the base of the finger. All of the chambers had the same spacing and wall thicknesses, so the only difference between them was the height of the chamber. According to the Soft Robotics Toolkit, this should have caused the fingertips to bend slightly later than the base and middle of the fingers since smaller chambers require more pressure to actuate. However, this did not necessarily prove to be the case. The palm profile was reduced compared to the 3D printed gripper discussed in Chapter 2, so the gripper would not be too large. Correspondingly, the palm chambers were designed to be even taller than the fingers, which again in theory would require less pressure to bend more.

3.1 Materials

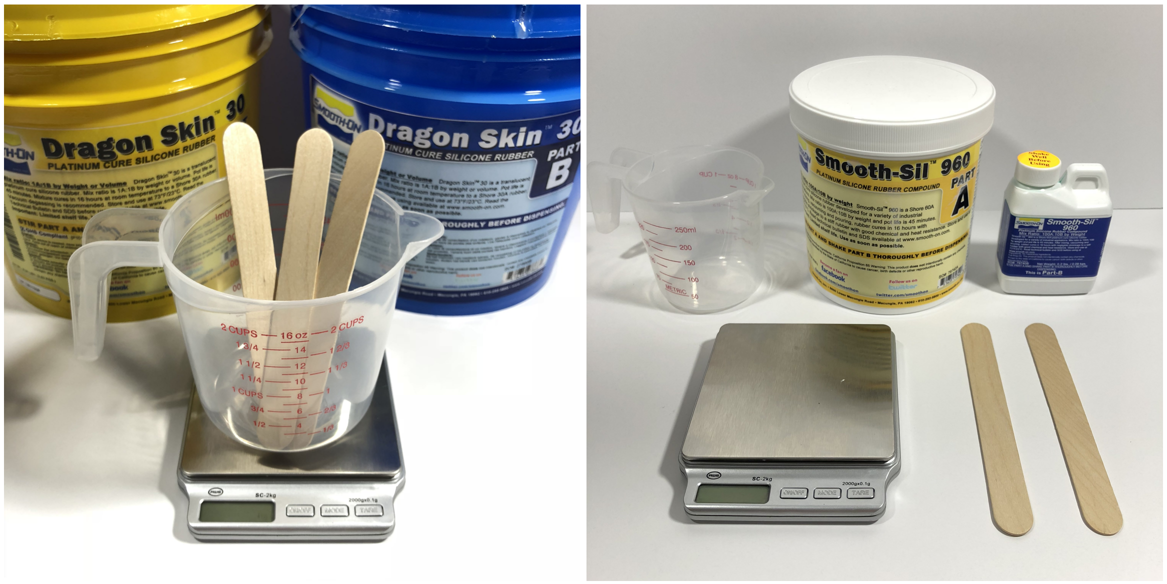
The materials used to produce the soft gripper prototypes introduced in this thesis include two types of platinum cure silicone from Smooth-On: Dragon Skin 30 and Smooth-Sil 960. Dragon Skin 30 has a 30A Shore hardness and made up most of the gripper. Smooth-Sil 960 has a 60A Shore hardness and was used inside the gripper to provide a more inextensible layer. The inextensible layer was used in the grippers to add internal stiffness to promote bending of the gripper when actuated.

A small digital scale, wooden sticks, plastic cups, and a vacuum chamber were used for measuring, mixing, and degassing the silicone before pouring it into 3D printed molds made of PLA and Ninjatek's Cheetah filament. Fasteners for this project included 6mm bolts and nuts for aligning the mold pieces and holding them together, 5mm bolts and nuts for connecting the gripper to the KUKA industrial robot, and 2mm bolts and heat-set inserts for fastening the gripper to its mounting piece. Rubber o-rings were used in conjunction with these 2mm bolts to help seal out contaminants, and rubber grommets were used to help seal the air inlets of the gripper to the air supply tubing. More information on the manufacturing process is provided in the following sections.

3.2 Mixing Silicone

When mixing Dragon Skin 30, a plastic cup was zeroed on the digital scale, and two wooden sticks, one per silicone part, were used to accurately dispense equal parts by weight of part A and part B (figure 3.1a). A third stick was used to thoroughly mix the two parts together. Once mixed, the plastic cup was placed in a vacuum chamber to remove the bubbles introduced during mixing. It is important to leave enough room in the plastic cup so the silicone has room to expand when degassing and collapse back onto itself when the majority of the gas has been sucked out.

Smooth-Sil 960 followed a similar mixing process. However, the ratio of part A to part B is 10:1 by weight (figure 3.1b). One wooden stick was used to help dispense part A into the cup, while part B was poured directly from the bottle. A second wooden stick was used to mix the parts together before them being degassed in the vacuum chamber like the Dragon Skin 30.



(a) Dragon Skin 30 (b) Smooth-Sil 960

Figure 3.1: Smooth-On Platinum Cure Silicone

Each of these silicone mixtures had a working time of 45 minutes, so needed to be poured into the mold before they started to solidify. They then also both required 16 hours to fully cure. A desirable trait that this silicone possessed was that more silicone could be added to the structure even after the cure period, although the bond was much better if the silicone had not yet reached its cure time. Consultation with a Smooth-Sil representative indicated that the best time to bond more silicone was when the initial pour was tacky but did not come off on a finger if touched.

3.3 Pouring the Top Half

The top half of the gripper was made using a two part mold (figure 3.2), with one part forming the internal airways and chambers and another that formed the outside. The molds were bolted together using twelve 6mm bolts and nuts, and twelve 2mm threaded rods were stuck in the middle to form the 2mm mounting bolt holes (figure 3.3). Three hundred twenty grams of Dragon Skin 30 was then poured into the assembled mold until it reached the top of each chamber. Two finger covers were bolted to the top of the mold to squeeze out the excess silicone (figure 3.4). This helped keep the top halves of prototypes more consistent. The palm chambers were left uncovered so more silicone could be added if necessary.

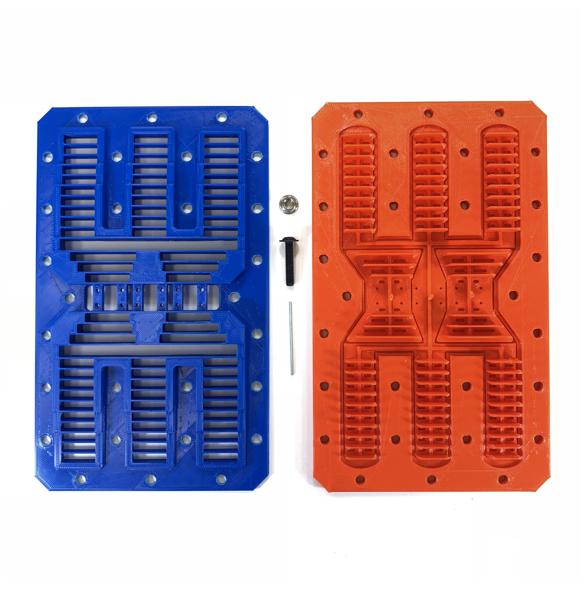


Figure 3.2: Two Part Mold for Gripper Top Half

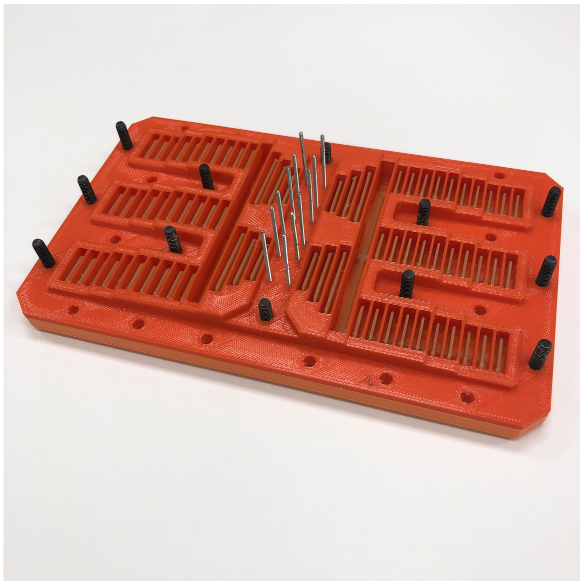


Figure 3.3: Assembled Top Mold

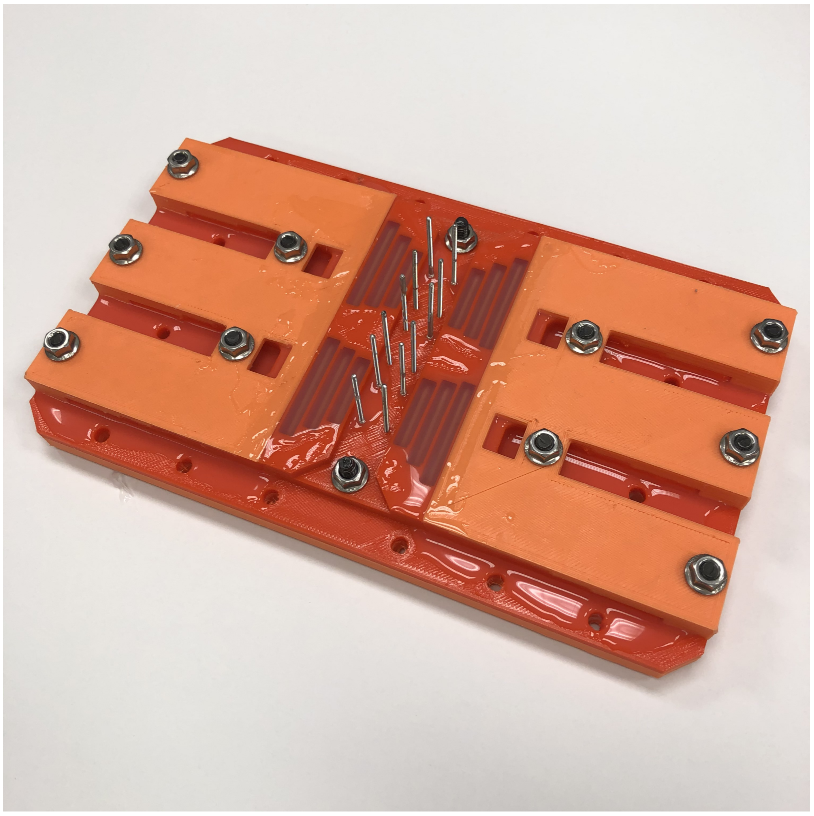


Figure 3.4: Adding the Finger Covers

After the silicone cured for a few hours, the finger covers, threaded rods, nuts and bolts, and the bottom part of the mold that created the airways and chambers were removed, leaving the silicone in the top part of the mold. Stents were pressed into the airway canals to hold them open during the sealing process. This was achieved by pouring 120 grams of Dragon Skin 30, or about a 2mm thick slab, into the sealing mold (figure 3.5). The top with stents was placed into the layer of silicone and slightly weighted down (figure 3.6).

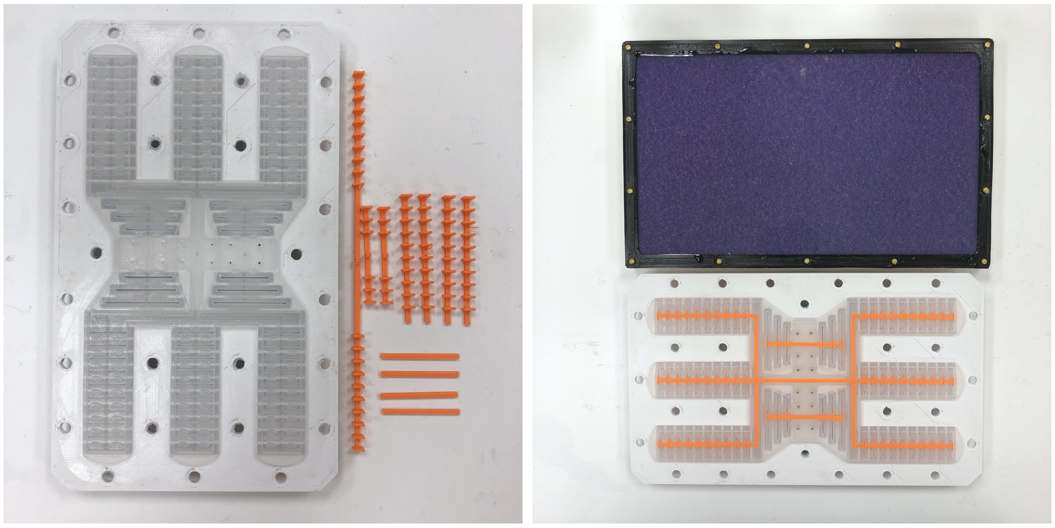

(a) (b)

Figure 3.5: Top Mold with Stents and Think Layer Mold

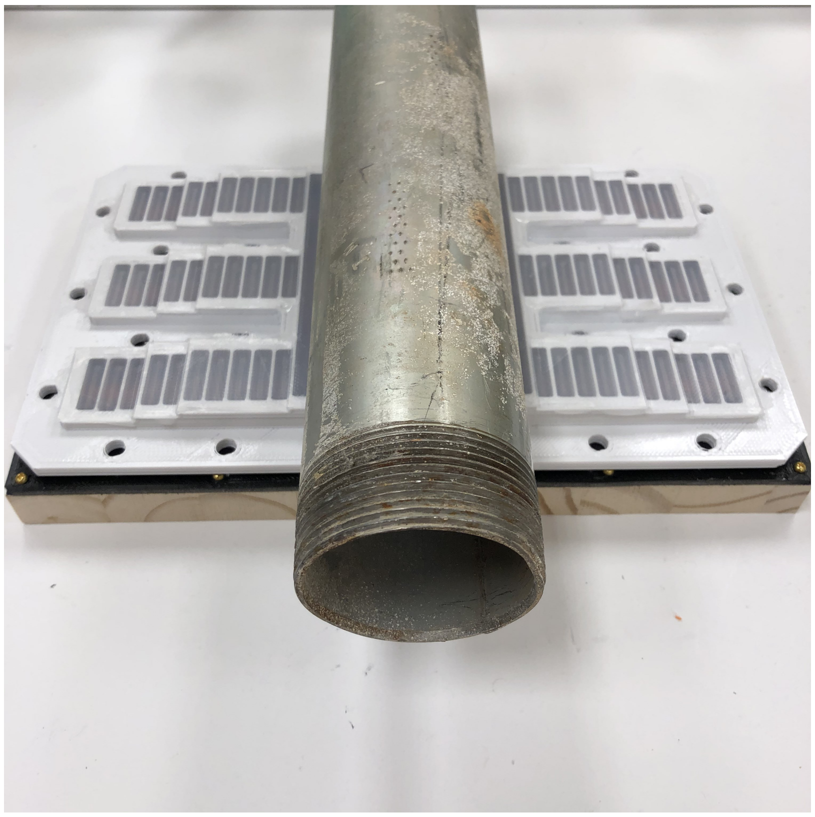


Figure 3.6: Sealing the Gripper

The thin layer of silicone was allowed to cure before the top half of the gripper was released from the molds. The extra silicone around the perimeter of the gripper was cut away, and a knife was also used to cut out along the airways of the gripper to remove the stents (figure 3.7). Ten grams of Dragon Skin 30 was mixed and lightly applied to the stent removal cuts to reseal the gripper. A preliminary pressure test was performed to check if the top half of the gripper was working properly. If it was not, small adjustments were made, or in some cases an entirely new top had to be made. If the top proved functional, it was ready to be secured to the bottom.

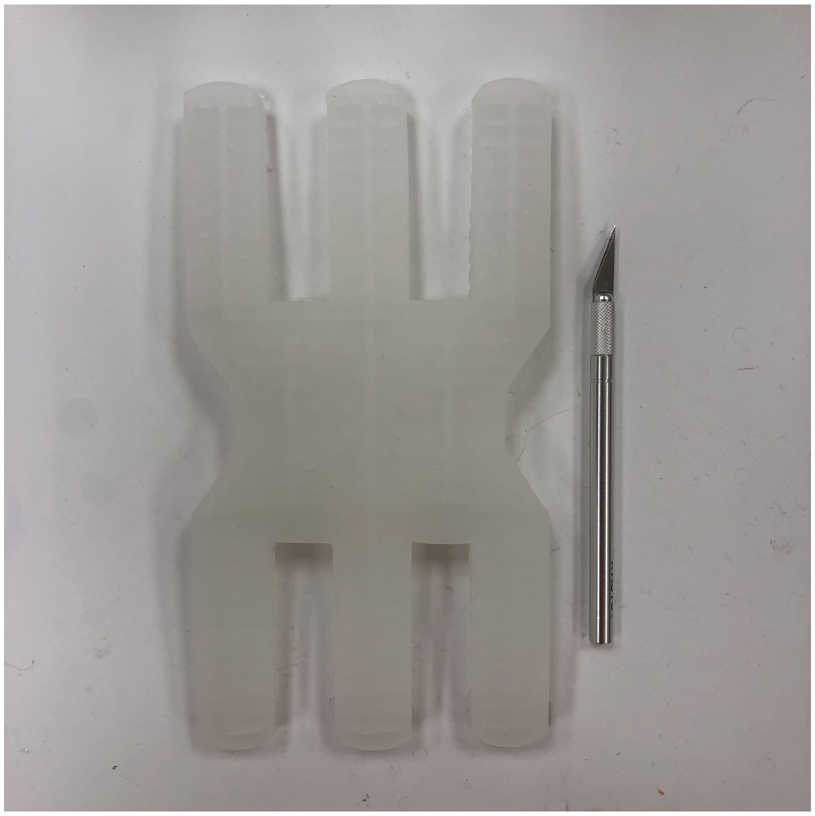
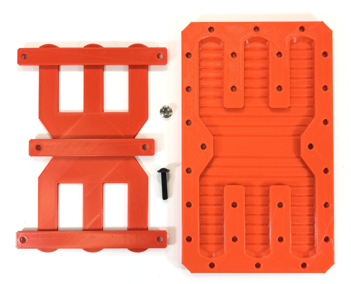


Figure 3.7: Removing the Stents

3.4 Pouring the Bottom Half

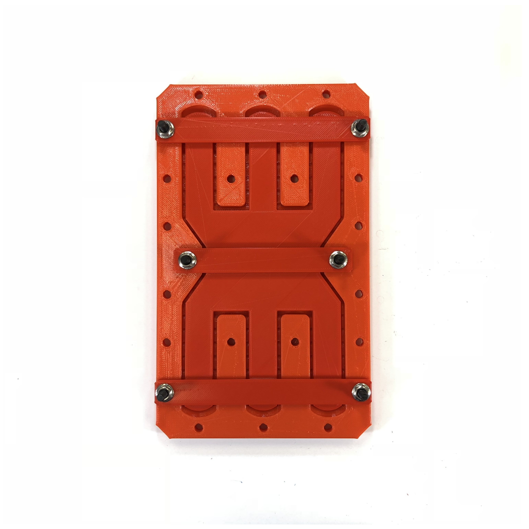
The bottom of the gripper was much simpler to make and rarely failed. It required a bottom piece and an impression piece (figure 3.8a), an embedded piece (figure 3.8b), and a guiding piece (figure 3.8c). Initially, 60 grams of Dragon Skin 30 were mixed and poured into the bottom piece before the impression piece was pushed down into the silicone and bolted in place (figure 3.9a). However, it turned it to be more reliable to clamp the bottom and impression together for best results (figure 3.9b).


(a) Impression and Bottom

(b) Embedded Piece

(c) Guiding Piece

Figure 3.8: Parts Need for the Gripper Bottom Half


(a)

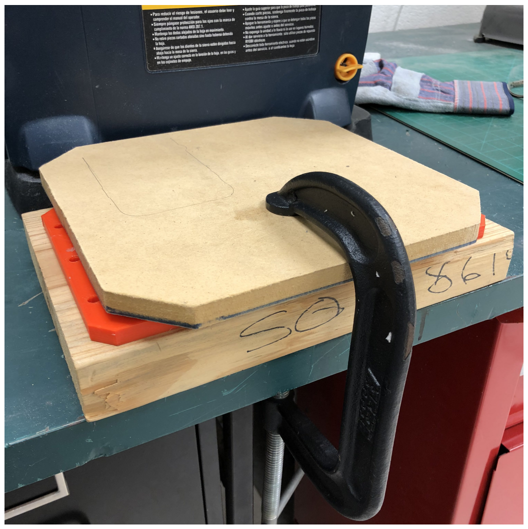
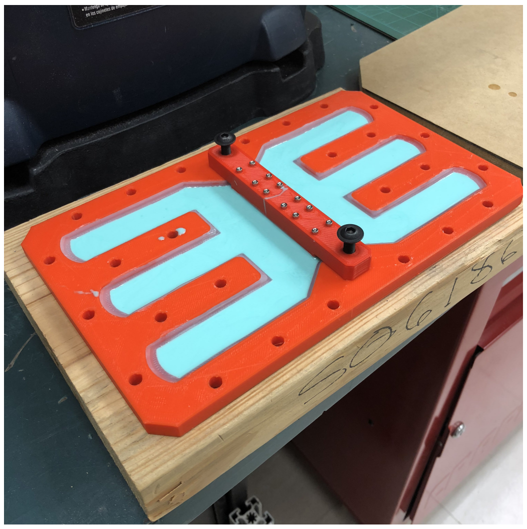

(b)

Figure 3.9: Pouring the Bottom Half

While the bottom was curing for a few hours, the 2mm heat-set inserts were added to the embedded piece holes using a soldering iron, and were screwed into the guiding piece using 2mm bolts with the top of the heat-set inserts facing down, or away from the guide. This was done to ensure that the inserts would not pull through the plastic when fastened. The impression piece was removed once the silicone had cured enough to become solid, and the guiding and embedded pieces were bolted across the middle of the bottom mold. Fifty-five grams of Smooth-Sil 960 was then mixed and poured into the mold (figure 3.10a). However, this section could have been filled with a variety of materials, including more Dragon Skin 30, which would have the advantage of one single pour of 120 grams (figure 3.10b) versus two smaller pours. This section also could house flexible sensors in the future.

Once this silicone was cured, the guiding piece was unscrewed from the embedded piece, and the bottom half of the gripper was ready to be added to the top half.


(a)

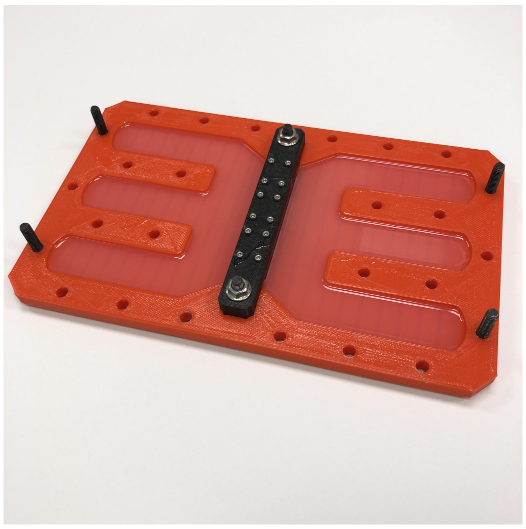

(b)

Figure 3.10: Adding the Impression Layer

3.5 Finishing and Testing

Before the top and bottom halves of the gripper were secured together, more 2mm threaded rods, similar to the ones used in the top half of the gripper were screwed into the heat-set inserts in the embedded piece (figure 3.11). It was important to have the threaded rods in place to help align the two halves and to prevent silicone from migrating into the mounting holes. Forty grams of Dragon Skin 30 were mixed and poured on top of this as a form of glue layer, which needed to be as evenly dispersed as possible. The top half was then aligned with the threaded rods and slid down them to meet the bottom half. They were pressed firmly together, with some silicone squeezing out of the sides. Then the silicone was left to cure.

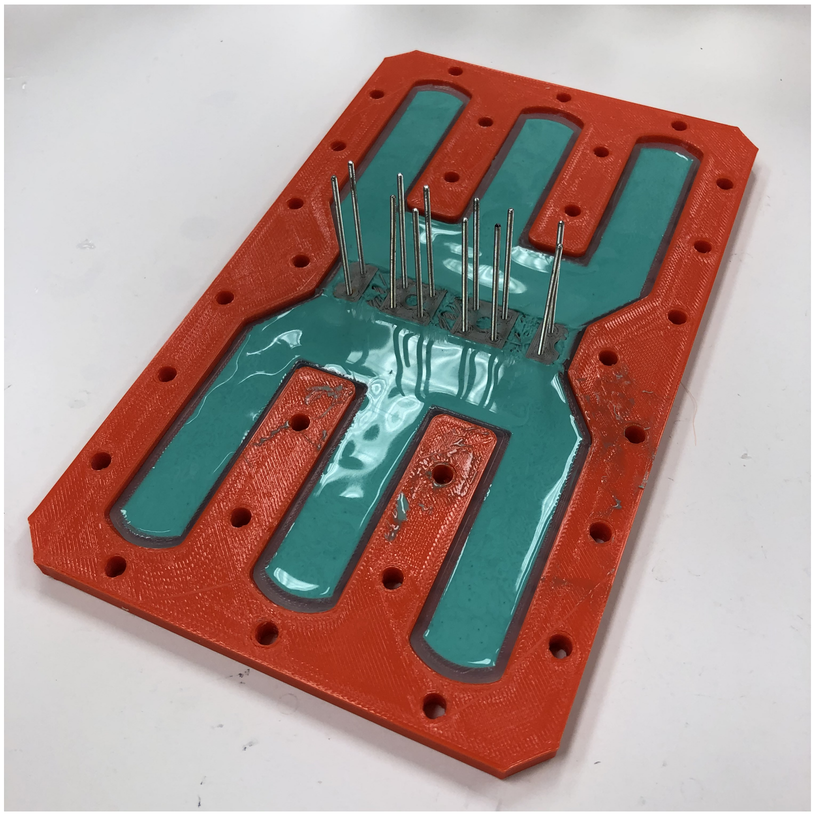


Figure 3.11: Bottom with Threaded Rods Installed

While the gripper was curing, the gripper mount was assembled (figure 3.12). The assembly had three rubber grommets, one in the middle for all six fingers and one on either side for the palm. Additionally, twelve 2mm bolts with rubbers o-rings were inserted into the twelve mount holes. Two 5mm hex nuts were press-fit into the bottom for mounting the gripper to the robot.

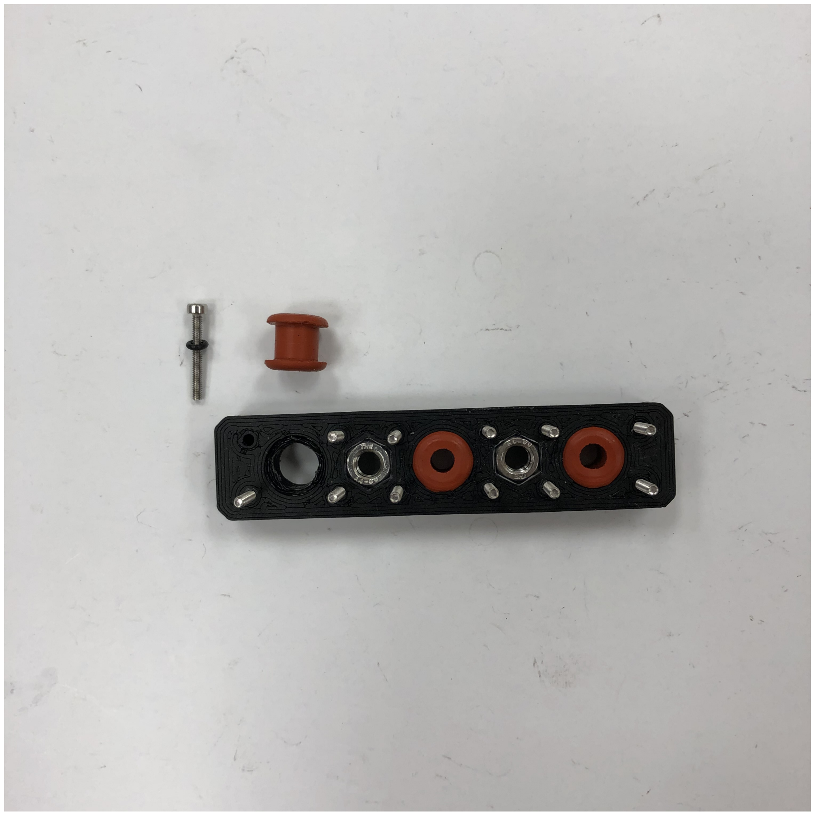
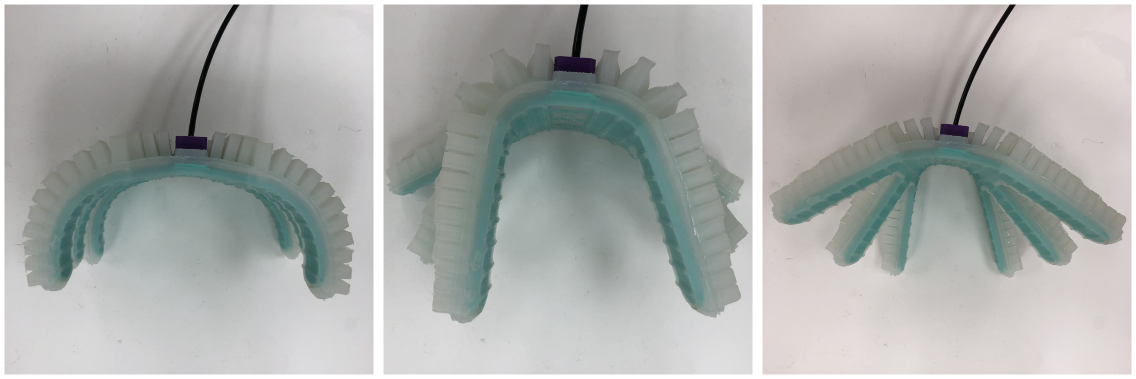


Figure 3.12: Gripper Mount

Once the gripper was fully cured, the threaded rods were unscrewed from the embedded piece and removed. The gripper was also removed from the mold, and any excess silicone was trimmed away. The gripper mount was screwed into the top of the gripper. Then a bike pump was used to check that the gripper actuated properly. See figure 3.13.



(a) Fingers (b) Front Palm (c) Back Palm

Figure 3.13: Testing the Chambers and Airways of a Finished Gripper

The procedures outlined in this chapter were used to create the final version of the gripper, but were the result of much experimentation and multiple silicone gripper iterations. A significant amount of the time spent on the project went towards these silicone gripper manufacturing iterations. Initial attempts were problematic in that the silicone stuck to the molds, which had both been made of PLA, and the mold halves were nearly inseparable. This led to attempts to use mold release spray on both sides of the molds, but this left a sticky residue on the silicone gripper parts.

Early versions were both difficult to remove and didn't seal completely, which led to redesign of the molds in a way that the silicone was easier to remove. This was done by splitting the top mold into more smaller sections; this second iteration also involved designing a gasket using Ninjaflex to seal the molds. Third, instead of reprinting multiple gaskets, the decision was made to use a Cheetah filament mold which was more flexible than PLA, and therefore assisted in sealing the halves. The silicone was naturally easier to remove from the Cheetah filament without assistance from mold release spray. Since it was no longer a challenge to pry the mold pieces apart, the silicone that was in the PLA mold was easy to remove as well.

The last key problem that was that of attaching the multiple silicone parts together after they had been removed from the molds. Originally, a layer of Dragon Skin 30 was poured into the bottom half of the gripper, and the top half was put on directly on top. When too little silicone was used, the palm and finger chambers did not seal properly, but too much silicone clogged the airways. To remedy this, the top was sealed separately first before being added to the bottom. Stents were not used until the same issue arose with sealing the top. Too much or too little silicone caused the gripper not to work. With the use of the stents, the gripper could reliably be sealed and connected together for testing.

Chapter 4

Results

When the gripper manufacturing process was complete, we evaluated the potential of the gripper for application in the chicken part handling process using the experimental setup illustrated in figure 2.1.

Following initial testing to tune the vision system and pressures applied to the gripper, we conducted a total of 1161 trials across four different poultry parts: legs, breasts, wings, and thighs. The samples were selected to provide variability in shape, size, weight, and compliance. Each part type was tested over 250 times with a success counting as a chicken part being grasped from a moving conveyor and placed into a tray. This number of trials was determined to be a reasonable number of trials based on other gripper experiments reported in the literature. A gripper designed for testing the firmness of eggplants reported trials using 234 eggplants [1], while other grippers have been claimed to be successful simply by picking and placing several objects ten times in a row [24], [2] or handling dozens of eggs without breaking them [3]. Given the intended real-world application of the gripper in this thesis we elected to test at the order of the higher number of trials, i.e. at or above 250, for each different chicken part.

In the experiments reported in this thesis, the overall rate of success from these trials was 63.57%. Figure 4.1 shows the comparison between the success rates of each part and contribution to the overall success rate.

Legs and breast pieces were initially tested, with palm pressures of 22 psi and finger pressures of 6 psi (table 4.2). This proved highly successful, with success rates at around 95%. The actively actuated palm proved highly effective in supporting the fingers in obtaining stable grasps. However, the palm pressure proved to be too much for the gripper to withstand, and caused the air chambers to bust after a short period of testing.

Therefore, the palm pressure was reduced to 9 psi while the finger pressure was increased to 12 psi to compensate for the loss in grip force. These lower palm pressures were used to complete the leg and breast piece testing (table 4.3) as well as to conduct all of the wing and thigh trials. The reduced palm pressure increased the dependence on the fingers for obtaining stable grasps, and the success percentage using this mode dropped significantly. A summary of all trials is shown in table 4.1.

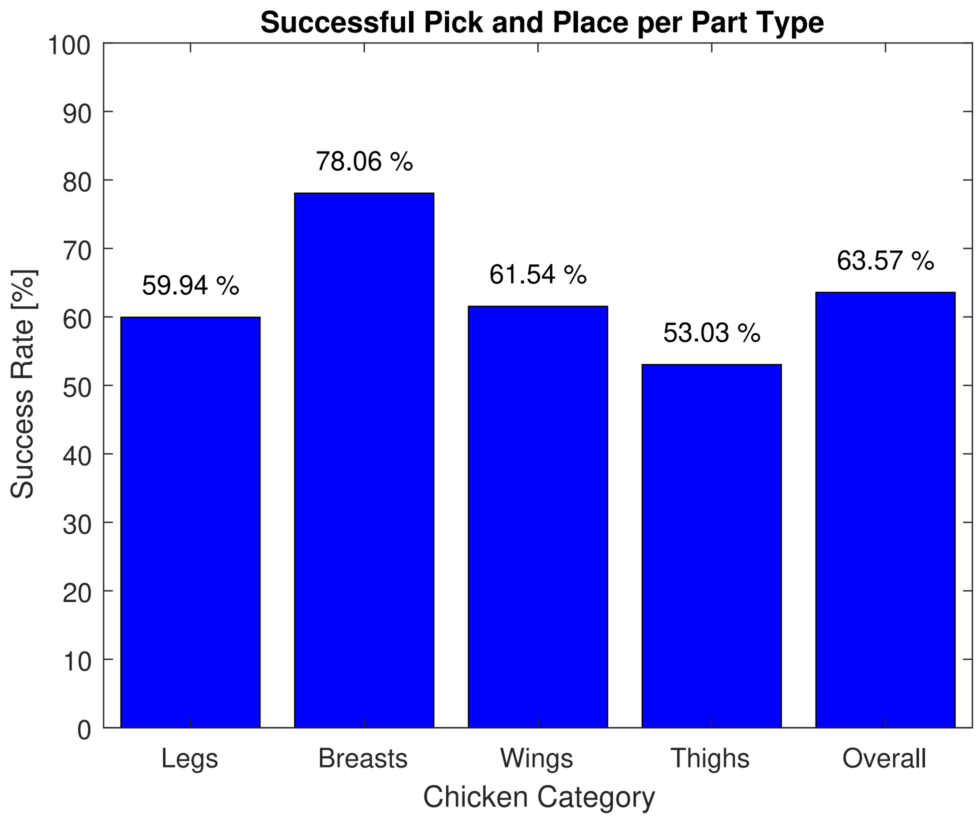


Figure 4.1: Rate of Successful Picks and Places per Part

Part Type	**Total Trials**	**Success**	**Fail**	**Rate**
Legs	327	196	131	59.94 %
Breasts	310	242	68	78.06 %
Wings	260	160	100	61.54 %
Thighs	264	140	124	53.03 %
Overall	1161	738	423	63.57 %

Table 4.1: Experimental Results Summary

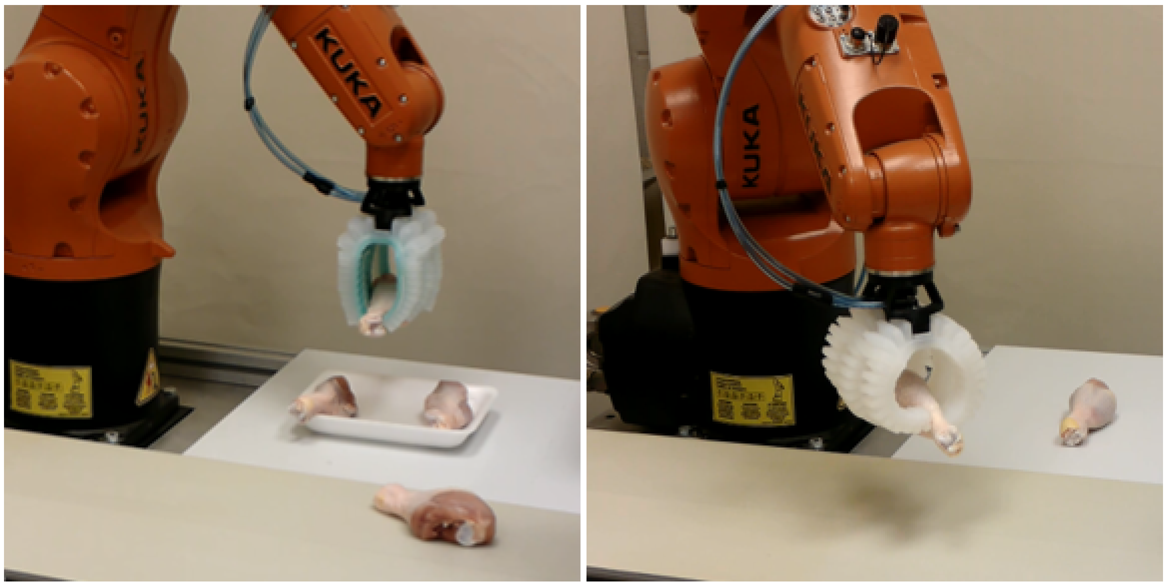
Part Type	**Trials**	**Success**	**Fail**	**Rate**
Legs	37	35	2	94.59 %
Breasts	170	162	8	95.29 %

Table 4.2: High Palm Pressure Experiments

Part Type	**Trials**	**Success**	**Fail**	**Rate**
Legs	290	161	129	55.52 %
Breasts	140	80	60	57.14 %

Table 4.3: Low Palm Pressure Experiments

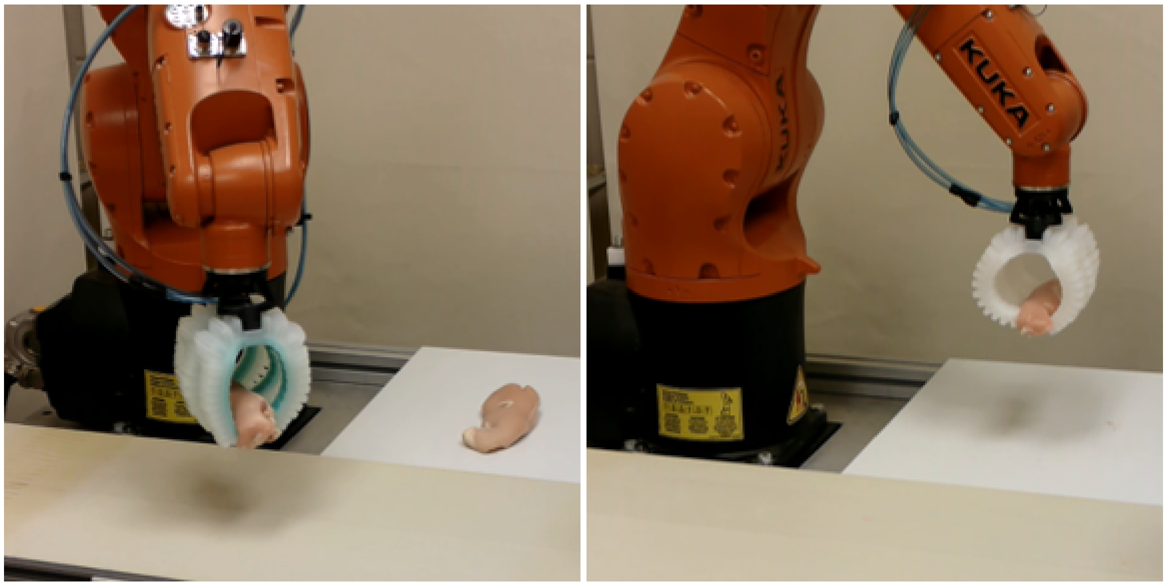
A total of 327 legs, or drumsticks, were tested with 196 of those legs being picked, yielding a success rate of 59.94%. Initially, the gripper was better tuned to grasping legs, but this required the higher palm pressures. Of the 327 legs, 37 were tested with the higher pressures and resulted in 35 of those legs being successful picks and places, which gave a 94.59% success rate. Two hundred and ninety legs were tested at the lower pressure, but only 161 were successful. This gave a success rate of 55.52%.



(a) High Palm Pressure (b) Low Palm Pressure

Figure 4.2: Successful Leg Picks

Three hundred and ten boneless skinless breasts were tested. Two hundred and forty-two were successful, yielding a 78.06% success rate. One hundred and seventy were tested at the higher palm pressure, and 162 were picked and placed. The higher pressures were successful 95.29% of the time. One hundred and forty were tested at the lower palm pressure with a success rate of 57.14%, or 80 picks and places.



(a) High Palm Pressure (b) Low Palm Pressure

Figure 4.3: Successful Breast Picks

Wings were tested two hundred and sixty times at the lower palm pressure with one hundred and sixty successes, resulting in a success rate of 61.54%.

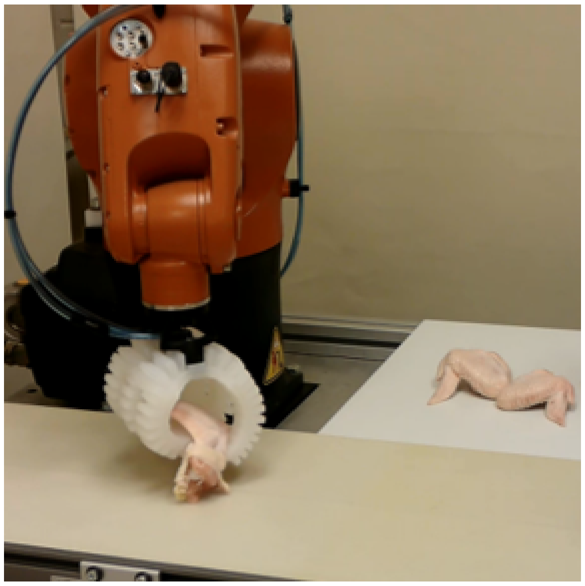


Figure 4.4: Successful Wing Pick Using Low Palm Pressure

Two hundred sixty-four skinless thighs were also tested with the lower palm pressure with one hundred and forty successes. This yielded a success rate of 53.03%.

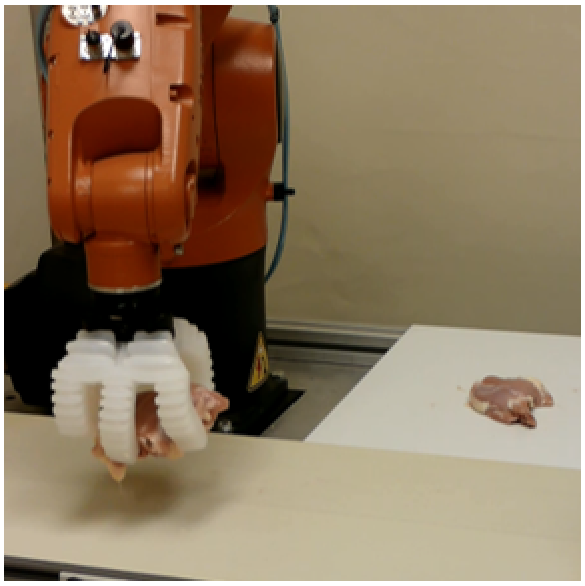


Figure 4.5: Successful Thigh Pick Using Low Palm Pressure

Summarizing these experiments, it can be seen that the gripper had reasonable overall success with picking and placing a variety of chicken parts, and demonstrated a very high success rate when the palm was actuated to reach the higher pressures found to provide a firm but delicate grasp. However, the gripper's palm could not sustain the high pressures for long periods of time. The longest run the gripper had at the higher palm pressure was about 180 trials before bursting. This was only a small portion of the trials reported in this thesis, and many less than the gripper would have to perform in a chicken processing plant. Bursting at higher palm pressures was experienced with multiple grippers, with the bursting consistently occurring in the palm.

Compensating for the drop in palm grasping force by increasing the finger pressure was partially successful, allowing a single gripper to conduct the bulk of the (greater than a thousand) trials reported here. However, this mode of operation was not ideal since at the lower palm pressures the gripper tended to 'fumble' more when contacting the pieces and roll them up into the fingers rather than gracefully grasp them. This led to numerous drops and failed grasps, significantly reducing the overall success rate.

Additionally, errors in the planning algorithm for intercepting chicken parts on the conveyor caused a number of misses due to the gripper not aligning above the part properly. These failures were not due to a gripper malfunction, but were due to software limitations. In summary, via the experiments reported in this Chapter the core design concept underlying the gripper was validated, and the hardware prototypes achieved generally positive results, though the material and software limitations inhibited greater success.

Chapter 5

Conclusions and Discussion

This thesis introduced a new soft gripper with a novel actively actuated palm for handling raw chicken pieces and packaging them into trays. The underlying motivation for the work was in response to a need for consistency in production volume from poultry processing plants which robots can potentially provide, given the unpredictability in availability of human workers. A successful gripper design must grasp and place pieces across a variety of shapes and sizes, as well as not damage the poultry in handling.

Three prototype grippers were explored, and the lessons learned were incorporated into a final working design. These early designs consisted of a spatula and tongs in an ultimately unsuccessful attempt to scoop the chicken pieces; a three-fingered fixed-palm gripper with artificial, or McKibben, muscles, that had a limited grasp and used unsanitary materials; and a 3D-printed four-fingered design with an articulated palm that had air chambers connected throughout that bent under pressure and was made mostly of 3D-printed Ninjaflex and Cheetah filament.

The most successful concept emerging from these designs was that of using a variable palm with flexible fingers that had a wide grasp, and this was carried into a final design. The problem with the third gripper had been the relative inflexibility of the materials, which led to the concept of using silicone instead of 3D-printed materials. This final design was manufactured by pouring silicone into 3D-printing molds to create a gripper with a variable palm and 6 fingers. A practical advantage of this design is that the silicone surface which contacts the raw chicken is of food grade material that is easy to clean.

The final gripper was evaluated in grasping and placing experiments, with the gripper mounted to an industrial robot arm and the chicken parts moving on a conveyor belt, and identified using a vision system. The experiments used sets of four different types of chicken parts, with over a thousand trials conducted overall. The experiments confirmed that the inclusion of the novel actuated palm was enabling in adapting grasps across the range of chicken parts handled.

To our knowledge, the soft robot gripper introduced in this thesis is the first with an independently actuated palm. It is also the first to demonstrate grasping multiple types of chicken parts, in thousands of trials, using a single gripper. However, while the silicone-based gripper showed moderate success in the experiments, there remains room for improvement, as discussed in the following section.

5.1 Future Work

The key focus for making improvements towards a deployable gripper design would be to modify the palm so that it is able to fully actuate without loss of structural integrity over large numbers of grasps. Palm bursting was the main limiting factor in the experiments, requiring lower grasp pressures (and corresponding reduction in success rate) to achieve gripper durability. This will require investigation of different materials as well as an exploration of various airway and chamber designs, starting with testing a variety of silicones and silicone combinations to find a more robust solution and eliminating sharp turns and edges in the airways and chambers to prevent possible tear points. Another adjustment in materials would be to use a different or additional material in the inextensible layer, such as paper or mesh, to investigate if this could potentially help the fingers and palm to bend more at lower pressures and impart a greater grasping force without risking damage to the gripper.

Additionally, the method for manufacturing grippers should be simplified as much as possible without losing the fundamental gripper concept of a pneumatically actuated variable palm and fingers. In order to make a consistently successful gripper, alternative design and manufacturing procedures should be tested.

In addition to improvements in production and materials, another refinement that could me made is to add force and flex sensors to the embedded layer in order to provide closed loop control of the grasp of the gripper on the chicken pieces. A further design change could involve moving the embedded piece to the outside of the gripper instead embedding it in each new gripper, a process which requires additional time and resources. In the poultry processing application, a blue conveyor belt could be used to provide the best contrast from the poultry parts and improve how the software detects and directs the robot system to pick up pieces.

With soft and continuum and soft robotics on the rise, there is great opportunity for robots to make an impact in every aspect of society. This thesis has explored several aspects of this emerging field, with the aim of making a difference in the food industry in the way that chicken is processed and packaged. Although there is still more work to be done, the gripper design introduced in this thesis has significant potential to contribute towards a more efficient future.

Appendices

Appendix A   Arduino Control Code

```
#include <Wire.h>
#include <Adafruit_MCP4725.h>

Adafruit_MCP4725 dac;
char data[3];
uint32_t value;

void setup(void) {
  Serial.begin(9600);

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

  // 0x60 fingers
  // 0x61 palm

  // Initialize both pressures to zero (0)
  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) {
  //Look for data on the serial connection
  while (Serial.available() >= 3)
  {
    for (int i = 0; i < 3; i++)
    {
      data[i] = Serial.read();
    }
    value = ((int)data[1]*100) + (int)data[2];

    switch (data[0])
    {
      case 'p': //palm section
        dac.begin(0x61);
        break;
      case 'f': //finger section
        dac.begin(0x60);
        break;
    }
    dac.setVoltage(value, false);
    delay(20);
  }
}
```

Appendix B Gripper Test Code

```
import serial
import struct
import time
import math

#Setup USB Communication
arduino = serial.Serial('com5',9600,timeout=1)
time.sleep(5)

while True:
    #Reset the Pressure Values
    arduino.write(b'p')
    arduino.write(struct.pack('>B',0))
    arduino.write(struct.pack('>B',0))
    time.sleep(0.05)
    arduino.write(b'f')
    arduino.write(struct.pack('>B',0))
    arduino.write(struct.pack('>B',0))

    time.sleep(5)

    # set the finger pressure
    arduino.write(b'f')
    # upper 2 digits, thousands and hundreds places
    arduino.write(struct.pack('>B',4))
    # lower 2 digits, tens and ones places
    arduino.write(struct.pack('>B',0))

    time.sleep(0.25)

    # set the palm pressure
    arduino.write(b'p')
    # upper 2 digits, thousands and hundreds places
    arduino.write(struct.pack('>B',1))
    # lower 2 digits, tens and ones places
    arduino.write(struct.pack('>B',50))

    time.sleep(10)
```

Appendix C Python System Software

```
# Henry G. Rutland IV
# Graduate Research
# Pick and Place

##############################################################################
# PYTHON SCRIPT NAME:                                                        #
# Talon_v2.py                                                                #
#                                                                            #
# PURPOSE:                                                                   #
# Use machine vision to detect raw chicken parts moving down a 12" wide by  #
# 4' long conveyor. Command a KUKA KR 6 R700 sixx robot to pick up the      #
# moving parts using the developed gripper and place the parts on the table.#
##############################################################################

# Communal Packages
from multiprocessing import Queue
from multiprocessing import Process
from datetime import datetime
from time import sleep

# Communal Classes
class NewPart:
    def __init__(self):
        self.params(0,0,0,0,0,0,0,0,0,0,0)
    def params(self,ID,timeS,xPos,yPos,theta,speed,length,width,zPos,palm,fing):
        self.ID = ID
        self.timeS = timeS
        self.xPos = xPos
        self.yPos = yPos
        self.theta = theta
        self.speed = speed
        self.length = length
        self.width = width
        self.zPos = zPos
        self.palm = palm
        self.fing = fing

class NewRequest:
    def __init__(self):
        self.msg(0,'')
    def msg(self,ID,state):
        self.ID = ID
        self.state = state

class NewPose:
    def __init__(self):
        self.pose(0,0,0,0,0,0,0)
    def pose(self,ID,X,Y,Z,A,B,C):
        self.ID = ID
        self.X = X
        self.Y = Y
        self.Z = Z
        self.A = A
        self.B = B
        self.C = C

# Communal Time Function to Return Current Time in Milliseconds
def millis():
    ct = datetime.now()
    #ms = (ct.day * 24 * 60 * 60 + ct.second) * 1000 + ct.microsecond / 1000.0
    ms = (ct.hour * 3600 + ct.minute * 60 + ct.second) * 1000 + ct.microsecond / 1000.0
    return ms

###############################################################################
# FUNCTION NAME:                                                              #
# vision()                                                                    #
#                                                                             #
# PURPOSE:                                                                    #
# The vision process continuously examines the conveyor for new blocks to    #
# send to the motion planning process via the visQ mutliprocessing queue.    #
# It takes about 15-20 frames per second, and each frame is color filtered  #
# to only leave red, orange, yellow, green, and blue blocks in a binary      #
# image, with the blocks appearing as white blobs and everything else as     #
# black. From the binary image, the centroid of each blob is determined and #
# checked to see if it is within a certain band of pixels. If the blob is   #
# found within the band, its centroid is converted to the x,y world          #
# coordinate system of the KUKA robot, and global time stamp at which the   #
# centroid was found is also saved. Finally, both the x,y coordinates and   #
# the time stamp are sent via a queue to the motion planning process. To    #
# ensure that blobs are not replicated, a box is drawn around the blob and  #
# centroid within the band, so if the centroid of a blob is found to be     #
# within a box drawn in the previous frame, it is supposed that it was a    #
# blob detected and added to the list of obstacles in a previous frame.     #
#                                                                             #
# PARAMETERS:                                                                 #
# visQ is the only input parameter and allows communication between the     #
# vision and motion planning processes.                                       #
###############################################################################
def vision(qVisPlan):
    try:
        print('vision() process started properly . . .')

        # Packages
        import cv2 as cv
        import numpy as np

        # Convert the Pixel Coordinate to KUKA World Coordinate Grid
        def px2cd(pX,pY,t):
            gX = (pX / 2.0) + 517
            gY = (pY / 2.0) - 585
            return [gX,gY,t]

        # Part ID for tracking parts among processes
        partID = 0

        ##### CAMERA CALIBRATION INFORMATION:
        #
        #   # Frame size of image: 640 X 480 pixels
        #   # Frame width (measured): 320mm
        #   # Frame height (measured): 240mm
        #
        #   # Top of Frame relative to KUKA: 517mm in X coordinate
        #   # Right of Frame relative to KUKA: -265mm in Y coordinate
        #
        ##### Approx Top Left of Frame Coordinates: (X)517mm, (Y)-585mm

        # Connect to Webcam
        cap = cv.VideoCapture(1)

        # Set the HSV Lower and Upper Ranges
        hsvLwr = np.array([0,0,0])
        hsvUpr = np.array([129,52,255])

        # Initialize the Bounding Boxes and Obstacle Lists
        boxes = list()
        parts = list()

        # Begin Capturing the Video
        while True:
            # Check if the Planner is Done
            #if not visQ.empty():
            #    if visQ.get() == 'Done':
            #        break

            # Grab the Current Frame
            ret, frame = cap.read()

            # Copy the Current Frame
            crop = frame.copy()
            overlay = frame.copy()

            # Crop the Current Frame
            crop[0:480, 0:23] = 255
            crop[0:480, 617:640] = 255

            # Convert the Frame to HSV
            hsv = cv.cvtColor(crop, cv.COLOR_BGR2HSV)

            # Mask/Filter Out Everythig but the Colored Blocks
            mask = cv.inRange(hsv, hsvLwr, hsvUpr)
            mask = cv.bitwise_not(mask)
```

```
# Display the Filtered Image
cv.imshow('Filtered',mask)

# Copy the thresholded image
im_floodfill = mask.copy()

h, w = mask.shape[:2]
newMask = np.zeros((h+2, w+2), np.uint8)

cv.floodFill(im_floodfill, newMask, (0,0), 255)

im_floodfill_inv = cv.bitwise_not(im_floodfill)
im_out = mask | im_floodfill_inv

# Display the FloodFill Image
cv.imshow('FloodFill',im_out)

# Initialize the Current Boxes and New Boxes
ctbox = boxes.copy()
boxes = list()

# Find the Contours in the Image
im2, contours, hierarchy = cv.findContours(im_out, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
for c in contours:
    M = cv.moments(c)

    # Find the areas
    area = cv.contourArea(c)
    #print(area)

    if area > 2700:
        # Find the bounding rectangle
        #bX,bY,bW,bH = cv.boundingRect(c)
        # Find the Angle of Rotation and the Major and Minor axis lengths
        #(cX,cY),(ma,MA),angle = cv.fitEllipse(c)
        #ellipse = (cX,cY),(ma,MA),angle
        #(cX,cY),(ma,MA),angle = cv.minAreaRect(c)

        #cv.ellipse(frame,(cX,cY),(MA,ma),angle,0,180,255,-1)
        #print(cX,cY,MA,ma,angle)

        # Calculate the Moment Coordinates
        if M["m00"] == 0:
            den = 0.0001
        else:
            den = M["m00"]

        cY = int(M["m01"] / den)
        if cY > 210 and cY < 270:
```

```
# Find the bounding rectangle
bX,bY,bW,bH = cv.boundingRect(c)
# Find the Angle of Rotation and the Major and Minor axis lengths
(cY,cX),(ma,MA),angle = cv.fitEllipse(c)
ellipse = (cY,cX),(ma,MA),angle

print('ma',ma)
print('MA',MA)
print('angle',angle)


#cX = int(M["m10"] / den)
print('center',cX,cY)
cX = int(cX)
cY = int(cY)
cv.ellipse(frame,ellipse,(0,255,0),2)
cv.circle(frame, (cY, cX), 5, (255, 255, 255), -1)
#cv.rectangle(frame, (cX-30,cY-60),(cX+30,cY+30),(0,0,255),2)
cv.rectangle(frame, (bX,bY),(bX+bW,bY+bH),(0,0,255),2)

# Append the list of new boxes for the next frame
#boxes.append((cX-30,cY-60,cX+30,cY+30))
boxes.append((bX,bY,bX+bW,bY+bH))
#print('box',bX,bY,bX+bW,bY+bH)

# Append the Obstacles List if a New Obstacle is found
newPart = False
if ctbox != list():
    for box in ctbox:
        if (cX > box[0] and cX < box[2] and cY > box[1] and cY < box[3]):
            newPart = False
            break
        else:
            newPart = True
else:
    newPart = True

if newPart:
    # Update the part ID
    partID = partID + 1

    # Create a new part
    part = NewPart()
    part.ID = partID
    part.timeS = millis()
    part.xPos = (cX / 2.0) + 517
    #part.yPos = (cY / 2.0) - 585
    part.yPos = (cY / 2.0) - 585
    print('CENTER',part.xPos,part.yPos)
    part.theta = angle
    part.speed = 120
```

```
                    part.length = MA/2
                    part.width = ma/2 - 40
                    #print('ma',ma)
                    #print('MA',MA)
                    qVisPlan.put(part)

            # Draw the Detection Band
            cv.rectangle(overlay, (0, 210), (640, 270), (255, 0, 0), -1)
            cv.addWeighted(overlay, 0.2, frame, 0.8, 0, frame)

            # Dispay the Current Frame
            cv.imshow('Current',frame)

            if cv.waitKey(30) & 0xFF == 27:
                break

        cap.release()
        cv.destroyAllWindows()

    except KeyboardInterrupt:
        print('\nVision Process Exiting . . .')
        return

###############################################################################
# FUNCTION NAME:                                                              #
# planner()                                                                   #
#                                                                             #
# PURPOSE:                                                                    #
# The planner process receives the new found chicken parts from the vision   #
# process via the visQ mulitprocessing queue and keeps track of all of the   #
# them as they move down the conveyor. This process also uses the            #
# force-based approach of collision avoidance to plan the motions of the     #
# KUKA as it moves up the conveyor. It sends the new positions and speeds    #
# to the KUKA via the kukaQ mulitprocessing queue.                            #
#                                                                             #
# PARAMETERS:                                                                 #
# visQ and kukaQ are the two input parameters, which are the                 #
# multiprocessing queues in which the new found obstacles are sent to the    #
# planner process from the vision process via the visQ queue, and the new    #
# KUKA positions and speeds are sent from the planner process to the kuka    #
# process via the kukaQ multiprocessing queue.                               #
###############################################################################
def planner(qParPlan,qVisPlan,qPlanArd,qPlanKuk,kukPos,kukSpd):
    try:
        print('planner() process started properly . . .')

        # Packages
        import math
        import numpy as np
        from numpy import linalg as LA
```

```
# Initialize the Incoming and Outgoing Parts Lists
partsIn = list()
partsOut = list()

# Continue Planning New Positions until loop broken by Parent
while True:
    # Get any new parts from the vision process
    while not qVisPlan.empty():
        partsIn.append(qVisPlan.get())

    # Process the Incoming Parts List
    for part in partsIn:
        #print('part',part)
        # update the part positions
        tNow = millis()
        part.xPos = part.xPos - (tNow - part.timeS) / 1000 * part.speed
        part.timeS = tNow

        # Move parts to Outgoing only if beyond certain X Position on the conveyor
        if part.xPos <= 550:
            # Calculate the part parameters
            part.width = part.width
            #part.zPos = (-5/28 * part.width/2 + 340)
            #part.palm = (part.width/2 - 140)/(0 - 140) * 80
            #part.fing = (-1/3 * part.palm + 200/3)

            # Convert the Pressure to 12bit Values
            #part.palm = int(part.palm / 130 * 4095)
            #part.fing = int(part.fing / 130 * 4095)

            part.zPos = 300
            part.palm = 550
            part.fing = 550


            # Add the part to the Outgoing Parts List
            partsOut.append(part)

            # Remove the part from the Incoming Parts List
            partsIn.remove(part)

    # Grab the oldest part (FIFO basis)
    if not partsOut == list():
        part = partsOut.pop(0)

        # update the part position
        tNow = millis()
        part.xPos = part.xPos - (tNow - part.timeS) / 1000 * part.speed
        part.timeS = tNow

        # Consider only parts in reach
```

```
if part.xPos >= 50:
    # Pick Position Evaluation Decrement in mm
    dx = 1

    # Distance ahead of Part in mm
    offset = 75

    # Create a New Pick Position
    pick = NewPose()
    pick.ID = part.ID

    # Starting Pick Evaluation Point
    pick.X = part.xPos - offset
    #print('pick.X',pick.X)

    while True:
        dist = LA.norm(np.array([pick.X,part.yPos])-np.array([kukPos.X,kukPos.Y]))
        rate = dist / (dx / part.speed)
        #print('dist',dist)
        #print('rate',rate)
        if rate <= kukSpd:
            break
        else:
            dx = dx + 1
            pick.X = part.xPos - offset - dx
            #print('pick.X',pick.X)

    if pick.X < 0 or part.ypos < -585 or part.ypos > -265:
        print('Part out of reach!')
        continue

    # Send the Finger and Palm pressures to Arduino Process
    qPlanArd.put(part)
    sleep(0.05)

    #pick.X = part.xPos - (dist/kukSpd * part.speed) - offset
    pick.Y = part.yPos
    pick.Z = part.zPos
    if part.theta > 90:
        pick.A = 270 - part.theta
    else:
        #pick.A = 90 - part.theta
        pick.A = part.theta
    #pick.A = 180 - part.theta
    pick.B = 0
    pick.C = 180

    # Send the pick position to the KUKA process
    qPlanKuk.put(pick)
    sleep(0.05)
    #print('pick',pick)
```

```
                # Wait on KUKA to finish; (blocking)
                while not qPlanKuk.empty():
                    # Do nothing
                    sleep(0.05)
                # Wait for Ready from KUKA process
                while qPlanKuk.empty():
                    # Do nothing
                    sleep(0.05)
                if qPlanKuk.get() == 'Ready':
                    print('Ready')
                else:
                    # Error!
                    print('An Error Occurred in KUKA Process')
                    return
            else:
                # Part removed by partsOut.pop()
                print('Part dropped!')

    except KeyboardInterrupt:
        print('\nPlanner Process Exiting . . .')
        return

##############################################################################
# FUNCTION NAME:                                                             #
# arduino()                                                                  #
#                                                                            #
# PURPOSE:                                                                   #
# The kuka process is simply meant to move the KUKA to its commanded        #
# positions at the given speeds and times. Once it receives a new position  #
# and speed from kukaQ, the KUKA moves to that position and notifies the    #
# planner process once it has finished the given command via the same kukaQ #
# multiprocessing queue.                                                     #
#                                                                            #
# PARAMETERS:                                                                #
# kukQ is the multiprocessing queue in which this process named kuka gets   #
# its updated positions and speeds from.                                     #
##############################################################################
def arduino(qPlanArd,qKukArd):
    try:
        print('arduino process() started properly . . .')

        # Packages
        import serial
        import struct
        import math

        # Setup serial comm with Arduino
        arduino = serial.Serial('com5',9600,timeout=1)
        sleep(1)
        print('Arduino Comm Ready . . .')
```

```
# Create a Dictionary of Part IDs and Pressure Outputs
press = dict()

# Continue Running until loop broken by Planner
while True:
    # Add to the Pressures Dictionary if needed
    while not qPlanArd.empty():
        part = qPlanArd.get()

        # Check if the process should end
        if part == 'Done':
            break
        # Otherwise add the part to the dictionary
        press[part.ID] = (part.palm, part.fing)

    # Poll for Requests from the KUKA Process
    if not qKukArd.empty():
        # Get the current KUKA request
        request = qKukArd.get()

        # Look up the part ID
        grip = press[request.ID]
        print('grip',grip)

        # Decide what to do
        if request.state == 'place':
            # Make both pressures zero (0)
            arduino.write(b'p')
            arduino.write(struct.pack('>B',0))
            arduino.write(struct.pack('>B',0))
            arduino.write(b'f')
            arduino.write(struct.pack('>B',0))
            arduino.write(struct.pack('>B',0))

            # Delete the part from the dictionary
            del press[request.ID]

        elif request.state == 'pick':
            # Write the finger pressure for the given part ID
            arduino.write(b'f')
            arduino.write(struct.pack('>B',math.floor(grip[1]/100)))
            arduino.write(struct.pack('>B',(grip[1]%100)))
            sleep(0.2)

            # Write the palm pressure for the given part ID
            arduino.write(b'p')
            arduino.write(struct.pack('>B',math.floor(grip[0]/100)))
            arduino.write(struct.pack('>B',(grip[0]%100)))

            #sleep(0.1)
            ## Write the finger pressure for the given part ID
```

```
                #arduino.write(b'f')
                #arduino.write(struct.pack('>B',math.floor(grip[1]/100)))
                #arduino.write(struct.pack('>B',(grip[1]%100)))

    except KeyboardInterrupt:
        print('\nArduino Process Exiting . . .')
        return

#############################################################################
# FUNCTION NAME:                                                            #
# kuka()                                                                    #
#                                                                           #
# PURPOSE:                                                                  #
# The kuka process is simply meant to move the KUKA to its commanded        #
# positions at the given speeds and times. Once it receives a new position  #
# and speed from kukaQ, the KUKA moves to that position and notifies the    #
# planner process once it has finished the given command via the same kukaQ #
# multiprocessing queue.                                                    #
#                                                                           #
# PARAMETERS:                                                               #
# kukaQ is the multiprocessing queue in which this process named kuka gets  #
# its updated positions and speeds from.                                    #
#############################################################################
def kuka(qPlanKuk,qKukArd,kukPos,kukSpd):
    try:
        print('kuka process() started properly . . .')

        # Packages
        #from robolink import *  # API to communicate with RoboDK
        #from robodk import *    # Basic matrix operations
        from robolink import Robolink
        from robolink import ROBOTCOM_READY
        from robodk import KUKA_2_Pose
        import numpy as np
        from numpy import linalg as LA

        # KUKA Robot Network Parameters
        ROBOT_IP = '172.31.1.147'
        ROBOT_PORT = 7000
        RDK = Robolink()

        # Select the Robot
        robot = RDK.Item('KUKA KR 6 R700 sixx')
        if not robot.Valid():
            print('No robot in the station. Load a robot first, then run this program.')
            pause(5)
            raise Exception('No robot in the station!')
        print('Robot selected properly . . .')

        # Set the KUKA Connection Parameters
        robot.setConnectionParams(ROBOT_IP,ROBOT_PORT,'/','anonymous','')
```

```
# Connect to the KUKA
success = robot.Connect()

# Check the Connection Status
status, status_msg = robot.ConnectedState()
if status != ROBOTCOM_READY:
    # Stop if the connection did not succeed
    print(status_msg)
    raise Exception('Failed to connect: ' + status_msg)

# Define the Robot Move Increment
#move_speed = 5

# Set the Initial Robot Speed
robot.setSpeed(kukSpd,360)

# Set the Robot Standby position
standby = [kukPos.X, kukPos.Y, kukPos.Z, kukPos.A, kukPos.B, kukPos.C]
STANDBY = KUKA_2_Pose(standby)
robot.MoveL(STANDBY)

# Continue Running until loop broken by Planner
while True:
    # Get next Part from Planner
    if not qPlanKuk.empty():
        pick = qPlanKuk.get()
        sleep(0.05)

        if not pick == 'Done':

            # Move to the X and Y of Pick Position
            pose = [pick.X,pick.Y,kukPos.Z,kukPos.A,pick.B,pick.C]
            POSE = KUKA_2_Pose(pose)
            robot.MoveL(POSE)

            # Rotate the Gripper for Grasping
            pose = [pick.X,pick.Y,kukPos.Z,pick.A,pick.B,pick.C]
            POSE = KUKA_2_Pose(pose)
            robot.MoveJ(POSE)

            # Lower the Gripper
            pose = [pick.X,pick.Y,pick.Z,pick.A,pick.B,pick.C]
            POSE = KUKA_2_Pose(pose)
            robot.MoveL(POSE)

            # Send request to Arduino Process
            request = NewRequest()
            request.ID = pick.ID
            request.state = 'pick'
            qKukArd.put(request)
```

```
                sleep(0.5)

                # Lift the part
                pose = [pick.X,pick.Y,(pick.Z + 100),pick.A,pick.B,pick.C]
                POSE = KUKA_2_Pose(pose)
                robot.MoveL(POSE)

                # Move to place
                pose = [400,0,(pick.Z + 100),kukPos.A,kukPos.B,kukPos.C]
                POSE = KUKA_2_Pose(pose)
                robot.MoveL(POSE)

                # Lower the part
                pose = [400,0,pick.Z,kukPos.A,kukPos.B,kukPos.C]
                POSE = KUKA_2_Pose(pose)
                robot.MoveL(POSE)

                # Send request to Arduino Process
                request.state = 'place'
                qKukArd.put(request)
                sleep(0.5)

                # Go back to Standby
                robot.MoveL(STANDBY)

                # Signal Planner for new Part
                qPlanKuk.put('Ready')
                sleep(0.05)

                # Wait for Planner to Read Ready Message
                while not qPlanKuk.empty():
                    sleep(0.05)

    except KeyboardInterrupt:
        print('\nKUKA Process Exiting . . .')
        return

###############################################################################
# FUNCTION NAME:                                                              #
# __main__                                                                    #
#                                                                             #
# PURPOSE:                                                                    #
# This is the parent process to three other processes that each serve        #
# different purposes. One handles the computer vision necessary to detect    #
# blocks that are coming down the conveyor and send the x,y coordinates      #
# as well as the time stamp for that object to the second child process,    #
# which handles the motion planning calculations. The motion planning        #
# process keeps track of the block obstacles as well as calculates the new  #
# positions and velocities for the KUKA, which are sent to the third child  #
# process. The third process handles the movements of the KUKA and lets the #
# motion planning process know when it isat the new position and is ready   #
# for the next position.                                                    #
##############################################################################
if __name__ == "__main__":
    try:
        # Shared KUKA Standby/Start Position
        kukPos = NewPose()
        kukPos.X = 400
        kukPos.Y = -425
        kukPos.Z = 300
        kukPos.A = 90
        kukPos.B = 0
        kukPos.C = 180
        # Shared KUKA Speed
        kukSpd = 300

        # Create Multiprocessing Queues
        qParPlan = Queue()  # data comm between parent and plan processes
        qVisPlan = Queue()  # data comm between vision and plan processes
        qPlanArd = Queue()  # data comm between plan and arduino processes
        qPlanKuk = Queue(maxsize=1) # data comm between plan and kuka processes
        qKukArd = Queue(maxsize=1)  # data comm between kuka dn arduino processes

        # Create New Processes
        visP = Process(target=vision, args=(qVisPlan,))
        planP = Process(target=planner, args=(qParPlan,qVisPlan,qPlanArd,qPlanKuk,kukPos,kukSpd))
        kukaP = Process(target=kuka, args=(qPlanKuk,qKukArd,kukPos,kukSpd))
        ardP = Process(target=arduino, args=(qPlanArd,qKukArd))

        # Begin Running the Processes
        visP.start()
        planP.start()
        kukaP.start()
        ardP.start()

        # Wait for the processes to end
        planP.join()
        print('planner() ended properly . . .')
        kukaP.join()
        print('kuka() ended properly . . .')
        visP.join()
        print('vision ended properly . . .')

    except KeyboardInterrupt:
        print('\nParent Waiting . . .')
        kukaP.join()
        ardP.join()
        planP.join()
        visP.join()
        print('\nParent Process Exiting . . .')
```

Bibliography

[1] C. Blanes, C. Ortiz, M. Mellado, and P.Beltran. Assessment of eggplant firmness with accelerometers on a pneumatic robot gripper. *Computers and Electronics in Agriculture*, 113.

[2] E. Brown, N. Rodenberg, J. Amend, A. Mozeika, E. Steltz, M. R. Zakin, H. Lipson, and H. M. Jaeger. Universal robotic gripper based on the jamming of granular material. *PNAS National Academy of Sciences of the United States of America*, 107(44):18809–19914, November 2010.

[3] H. Choi and journal = International Journal of Machine Tools Manufacture volume = 46 M. Koc, title = Design and feasibility tests of a flexible gripper based on inflatable rubber pockets.

[4] C. Cohen, B. Hiott, A. D. Kapadia, and booktitle = Proceedings of SPIE Conference 9836, Micro- and Nanotechnology Sensors, Systems, and Applications VIII, SPIE Defense and Security Conference location = Baltimore, MD month = April year = 2016 in Space: Continuum Surfaces for Robotic Grasping and Manipulation".

[5] FANUC. Fanuc m-430ia high speed meat handling robot. http://www.autocells.com/fanuc-m430ia-high-speed-meat-handling-robot.

[6] Z. Hawks, M. S. Islam, M. Lastinger, H. Rutland, M. Trout, I. D. Walker, and K. E. Green. Robots in the home: A novel suite of interactive devices for assisting with disease prevention and meal preparation. In *Proceedings of IEEE Global Humanitarian Technology Conference*, pages 715–718, October 2019.

[7] Soft Robotics Inc. Delicate and secure handling of chicken breast. https://www.youtube.com/watch?v=IO7JNu6rwhM, 2017.

[8] Soft Robotics Inc. Delicate and secure handling of raw chicken drumsticks. https://www.youtube.com/watch?v=iIXLCzDjaAQ, 2017.

[9] Soft Robotics Inc. High speed packaging of chicken legs. https://www.youtube.com/watch?v=e1ReBWvLVPQ, 2017.

[10] T. B. Jorgensen, M. M. Pedersen, N. W. Hansen, and B. R. Kruger. A flexible suction based grasp tool and associated grasp strategies for handling meat. In *Proceedings 4th International Conference on Mechatronics and Robotics Engineering*, February 2018.

[11] Marel. Robobatcher gets a firm grip on styling. https://marel.com/articles/robobatcher-gets-a-firm-grip-on-styling/, 2018.

[12] A. Matthieu, S. Franck, S. Laurent, S. Kevin, G. Grigore, and M. Youcef. Robotic solutions for meat cutting and handling.

[13] A. Pattersson, S. Davis, J. O. Gary, T. J. Dodd, and T. Ohlsson. Design of a magnetorheological robot gripper for handling of delicate food products with varying shape. *Journal of Food Engineering*, 98.

[14] M. Pellegrini. Rise of robotics in the meat processing industry. https://www.provisioneronline.com/articles/104778-rise-of-robotics-in-the-meat-processing-industry, May 2017.

[15] RoboDK. Kuka robots. https://robodk.com/doc/en/Robots-KUKA.htmlKUKA, n.d.

[16] RoboDK. Robot drivers. https://robodk.com/doc/en/Robot-Drivers.htmlRobotDrivers, n.d.

[17] Omron Robotics and Inc. Safety Technologies. Usda accepted high speed adept quattro robot handling raw chicken. https://www.youtube.com/watch?v=6eMcI6k3NA, 2009.

[18] Omron Robotics and Inc. Safety Technologies. Primary secondary food packaging with omron adept robotics. https://www.youtube.com/watch?v=mZUL6BdgINM, 2015.

[19] J. Shintake, V. Cacucciolo, D. Floreano, and H. Shea. Soft robotic grippers. 30(29), July 2018.

[20] T. Simonite. Robots start to grasp food processing. https://www.technologyreview.com/s/537646/robots-start-to-grasp-food-processing/, May 2015.

[21] Soft Robotics Toolkit. Pneunets bending actuators. https://softroboticstoolkit.com/book/pneunets-bending-actuator.

[22] I. D. Walker. Continuous backbone ”continuum” robot manipulators: A review. *ISRN Robotics*, July, 2013.

[23] I. D. Walker, H. Choset, and G. Chirikjian. “snake-like and continuum robots”. In *Handbook of Robotics*. Springer, 2016.

[24] Z. Wang, Y. Torigoe, and S. Hirai. A prestressed soft gripper: Design, modeling, fabrication, and tests for food handling. *IEEE ROBOTICS AND AUTOMATION LETTERS*, 2(4):1909–1916, October 2017.

[25] G. M. Whitesides. Soft robotics. *Angewandte Chemie International Edition*, 57(16):4258–4273, 2018.