ME 460 Mechanical Engineering Senior Design

A Technical Report for

Collaborative Robotic Arm with Integration of Vision Recognition, Mechanical Gripper, and Safety Systems

Submitted by:

Jose Bonilla
Mechanical Engineering, Class of 2018
jbonilla@vols.utk.edu

Christopher Mobley
Mechanical Engineering, Class of 2018
cmobley4@vols.utk.edu

Benjamin Terry
Mechanical Engineering, Class of 2018
bterry7@vols.utk.edu

Jasmine Worlds
Mechanical Engineering, Class of 2018
jworlds@vols.utk.edu

Mechanical, Aerospace & Biomedical Engineering Department
The University of Tennessee
Knoxville, TN 37996-2210

TABLE OF CONTENTS

ABSTRACT 1

INTRODUCTION 1

Background 1

Scope and Initial Goals 2

INITIAL RESEARCH AND ANALYSES 2

Vision Recognition 2

Safety System 3

Mechanical Gripper 3

System Enclosure 4

FUNCTIONS AND REQUIREMENTS 4

Mechatronic Parts 4

Software and Programming 12

CONCLUSIONS AND RECOMMENDATIONS 14

Mechanical Parts 14

Software and Programming 15

REFLECTION AND LESSONS LEARNED 18

Benjamin Terry 19

NEXT STEPS (Fall 2018) 19

Appendix A: Work Breakdown 20

Schedule with Milestones (Fall 2017) 20

Schedule with Milestones (Spring 2018) 20

Appendix B: Bill of Materials 23

Appendix C: CAD Drawings 24

Appendix D: Visual Recognition Code Line (C++) 26

track_marker.cpp 26

calibration.cpp 30

Appendix E: AUBO Script Communication Proof of Concept 38

TestFile1.aubo 38

Appendix F: Arduino Code for integrated Grip and Trip Wire 39

bterry7_ME450_AUBOStrong_MainAll.ino 39

Appendix G: Circuit Diagrams for Grip System and Trip Laser Safety System 41

ABSTRACT

The purpose of this project was to integrate an AUBO-i5 collaborative robot with both a gripper and camera vision system into a conveyor tracking application. The manipulator's goal is to pick and then precisely place parts from a conveyor belt. Moreover, the AUBO-i5 is a collaborative system so an additional objective is to integrate a trip laser safety system for reduced speeds and safety zones within the work cell designed. Students were tasked with designing and then building a system enclosure that could achieve the previously mentioned objectives. Work on this project resulted in the design and creation of software for robot vision and control as well as a safety system for the robot's active area. This project was broken down into two primary functional areas: mechatronics and computer programming.

The mechatronic area consisted of the mechanical gripper, various sensors, the safety system, and the system enclosure. For the mechanical gripper, the selection was AUBO provided pneumatic system and a purchased, servo driven grip. AUBO also independently purchased an advanced ROBOTIQ gripper. The servo driven grip was ultimately used, however, due to computer communication issues. This grip was to incorporate force sensors for better control. The safety system was to be composed of laser lights/dots, a system of mirrors, and photoresistors. All of this, especially the mirrors which would define the safety areas edge, was to be housed nicely by some form of system enclosure.

The programming can be broken into two relatively distinct elements. First was the vision recognition. This used the OpenCV library and relied on Aruco markers placed on the object. Said library and markers would allow the computer to find the (x,y,z) of the object from the camera. The second major programming task was communication. The camera had to feed information to the computer, which communicated with the AUBO robotic arm as well as account for information from external sensors (safety system, force sensor) and control the mechanical gripper.

INTRODUCTION

The basis of this project is to demonstrate the potential uses of the Aubo Robotics Aubo-i5 robot. Specifically, the initial task involves the integration of a vision recognition system, which has remained the core objective of the project. Supplementing this primary objective are a number of smaller goals and tasks which include the integration of a proximity detection/LIDAR/safety system and a mechanical grip mechanism. An additional task of this project was to incorporate a dynamic component. Moving objects were expected to be picked and then placed to a specific waypoint with attention to orientation. Ideally, this operation would be accomplished in the most optimized manner.

Background

Aubo Robotics is an existing collaboration established between professors from the United States and China. The goal of this group is to make a lightweight collaborative robot capable of various functions. This robot uses an open source architecture which

enables the Robot Operating System (ROS) to be supported by an API, which means it is capable of being used for both industrial and educational purposes.

Additionally, the Aubo-i5 is equipped with an open source architecture, meaning that the robot has the ability to utilize a CAN bus network, which allows for multiple microcontrollers to communicate with each other; it also has the necessary hardware to adopt bus protocols with open I/O interface extensions and can be easily integrated into current production systems. Moreover, the robot's intelligence and modular design are featured components as well. The Aubo-i5 has the capacity to easily integrate vision systems. Its software system is also interesting in the fact that it is capable of being updated and managed on a cloud based platform.

Scope and Initial Goals

To integrate an AUBO-i5 collaborative robot with both a gripper and camera vision system into a conveyor tracking application. The manipulator's goal is to pick and then precisely place parts from a conveyor belt. Because the AUBO-i5 is a collaborative system it will also integrate a trip laser safety system for reduced speeds and safety zones.

The robotic workcell for this project is to be designed and then implemented by the students. The functionality of the AUBO-i5 must be capable of recognizing a variety of parts (products), which will be presented randomly, and have the ability to perform the pick and place application.

To summarize, the initial goals were as follows: a vision recognition system, working with a mechanical gripper, to pick and place an object, hopefully at random from a conveyor belt, and to incorporate a tripwire safety system

INITIAL RESEARCH AND ANALYSES

Initial research and analyses done for the Aubo-i5 were vital in the start of this process. There was an importance on first understanding what the capabilities of the Aubo-i5 were, as well as, some basic training regarding how the user interface operated and how the safety function, power and force limiting design, worked. Furthermore, it was important to have a firm understanding of what the deliverables were for this project. Moreover, research regarding the various components of the system had to be done to receive a full understanding of what parts would be needed to make this a success.

Vision Recognition

In regards to parts, the starting point, with vision integration being a primary goal, was obviously a camera. While a Cognex camera was originally proposed, such a system would have cost a majority of the budget. Further, we felt that merely adapting an existing system would defeat the purpose of Senior Design, at least for such an integral part of the project. The main real requirements for the camera chosen were computer connectivity (via USB) and fast transfer of data to keep up with the robot; a wide-angle would also be useful to maximize working area. A Logitech HD Pro Webcam C920 was ultimately chosen for these reasons.

The camera would then be used in conjunction with vision recognition software to provide the robot coordinates of objects. When first researching recognition vision software, it was obvious that money could be spent on a pre-packaged software. However, opencv was a very inexpensive option. Opencv is an open source computer vision and machine learning software library. It consist of over 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. Using this opencv would allow more flexibility and creativity when working with the vision aspect of this project.

Several options for image recognition were considered including creating an object profile from several calibration photos which could be used to recognise specific parts. This option, however, does not allow for an accurate enough determination of position and orientation. Instead of this we decided to use black and white QR code-like images which were able to output more precise coordinates.

Safety System

Furthermore, a safety system was to be introduced. To accomplish this, we utilized a laser light (think cat toy) shone onto a photoresistor. The light will be reflected using mirrors to enclose the designated working area. From there, various safety protocols can be instituted. As it stands, the laser will indicate whether a person has entered the safety zone; upon exit, a switch will need to be flipped to indicated that the area is clear.

There are many pre-built systems that could be integrated, but they are costly and we felt this task could be built in house. Initial research was into full 3D imaging equipment (http://scanse.io/) and other manufactured safety systems. Unfortunately, most premade systems were, at minimum, around $500 and thus unattractive. Further, in the spirit of Senior Design, we decided we would be more than capable of making and implementing such a system ourselves. Further research suggest infrared, ultrasound, and lasers as potential proximity detectors. Lasers, as opposed to ultrasound or infrared, was something decided fairly early. This decision was driven by a desire for more accurate readings (ultrasound being too noisy) and the ease of controlling the size (range from robot and angle of rotation about the robot) of the safety zone. Further, when first discussing the issue, we all agreed intuitively that a laser would be a valid choice, so team cohesion and planning were much easier.

Mechanical Gripper

For a grip system, little research was done as the Cherokee farm location where we were working already had a pneumatic gripper we originally planned to modify and use.

Speaking more broadly, though, we did consider what kind of gripper would be most viable for the purpose of this project. The pneumatic gripper already available would use compressed air to open and close the “jaws or fingers”. The other primary option found, an electric gripper, would be driven by a servo. For the purpose of starting the project, it was more feasible to go with the pneumatic gripper as the materials for construction and assembly were already

present. 3D printing would be used to make the fingers of the gripper and an air pump and pneumatic gripper were already on hand. Going with this option would save time and money.

However, we wanted to make the system more flexible and predictable. Ideally, the gripper would have the capability to pick up a wide variety of objects, meaning that the gripper would need to have various opening sizes; this is difficult with a two state pneumatic actuator. As such, we later decided to buy an electric gripper, add force sensors, and then program as needed to meet functional requirements. Using the electric gripper would mean being capable of controlling the position of the gripper fingers, being able to detect grip, controlling the grip force and speed, and eliminating the need for air lines, thus creating more space.

System Enclosure

Considering the setup of the system is one of the most vital components of this project. Having an idea of what the system will be comprised of will help when making future decisions, such as required purchases. When initially discussing potential setups, it was agreed that the system utilize two separate cameras. One would be attached to the robot arm, which would create a more accurate pick up and insertion function. It would also aid in recognizing the approaching objects whenever the manipulator's arm obscured the vision from the external camera. This external camera would be used to signal when objects were approaching the arm on the conveyor belt and provide a rough estimate of the object's various locations.

Following this, a way to attach the secondary camera and the gripper to the robot will need to be designed. As of now, no set plans have been made. Aubo has as an attachment for the Aubo-i5 from which we can build. Alternatively, the attachment has been modeled; while not toleranced and the fit needs adjustment, this model would allow us to fabricate a completely new attachment setup if desired.

Lastly, a way of placing the camera above the workstation is needed. While not difficult in it of itself, the system should also be designed to be as easy to calibrate as possible.

FUNCTIONS AND REQUIREMENTS

For this project, a myriad of parts have to be either created or purchased to make this project successful. To simplify this process it can be separated into two main categories. *Mechatronic Parts*, which will consist of the system setup, as well as, mechanical components such as the gripper, attachment pieces, etc. and *Software and Programming*, which will consist of the vision software, communication software/programming to make the communication between the ROS (Robot Operating System) and I/O interface, etc.

Mechatronic Parts

Aside from the mechanics of the robot itself, which we generally did not deal with, the main mechanical task is gripping the object, installing and running the safety system, and system setup both of which are discussed above.

The initial design was to use a pneumatic system that AUBO already had at their office. However, there were problems with controlling the system, culminating in a burnt Arduino Due and Op Amp. Further, it was not as robust as the grip system we are currently using. Being pneumatic, it only had two states, open and closed. As such, the grip arms and code would need to designed around this. This could be accomplished by creating a grip that had a distinct position for each object and, upon recognition of that object, the robot could move to the corresponding orientation. To accomplish this, we designed a simple clamp where the separation of clamps changed along the length of the clamp; refer to **Figure 1**.

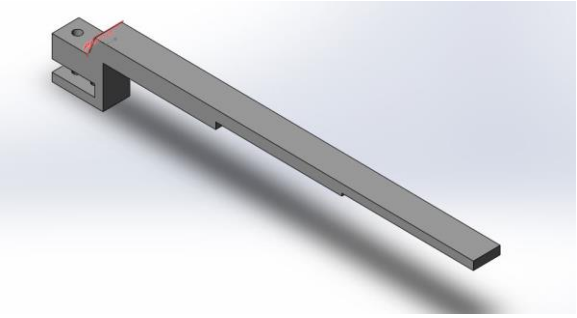


Figure 1: Initial Gripper Concept (1 half); to be attached to pneumatic actuator with different gap sizes for each object.

For the electrical gripper chosen, the actual mechanics were fairly simple. The kit purchased used a servo to run two mechanisms, interlocked by a gear train. Said mechanism caused the two grip arms to move in and out in a nearly parallel motion. For the system, a position of ~85 degrees on the servo was fully open while 0 degrees was fully closed. Refer to **Figure 2** for a visual aid.

Figure 2: Mechanical Grip System, Servo Controlled.

A tangential system to the gripper were the force sensors, pictured below in **Figure 3**.

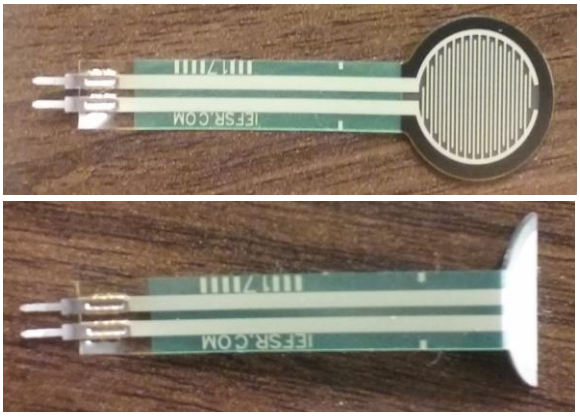


Figure 3: Force Sensors; work even when cut.

These sensors were found to work even when cut, allowing them to be more easily attached to the gripper's arms. Each sensor acts as a simple resistor which decreases in resistance when a force is applied normal to its surface. Attaching this between the jaws of the gripper and the object provides a readable voltage drop which can be used to stop the

grip motion and signal the robot that it has picked up an object.

A few potential problems did arise during testing, however. First, the voltage drop caused by the objects we are picking up (plastic children's toys) is miniscule. Generally, a voltage drop did not register until the grip force had exceeded that which was needed to actually grip the object. This was able to be corrected by making the grip retract some once the sensor has reached its threshold. From a manufacturing standpoint, however, this means that this system would likely be a ill-fit for any elastic items (such as plastic bags). At present, when using hard plastic parts no calibration is needed. Moving forward, data should be gathered to calibrate the system for a variety of materials.

Another issue is that the sensor pad exceeds the thickness of the gripper teeth, and when it presses on an object the bottom (which overhangs the grip) merely bends back and reads no force. Fortunately, the sensor works when cut, so the width of the pad can be made to match the thicknesses of the grip arms and thus this problem is resolved.

The setup for the grip system by itself running on an arduino can be seen below in **Figure 4.**

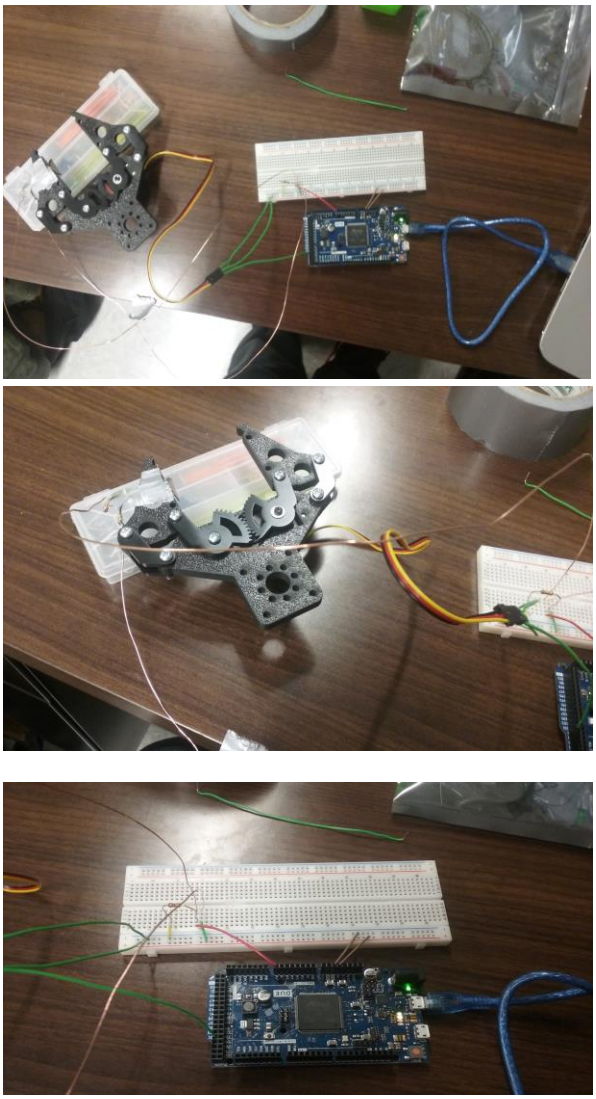


Figure 4: Gripper Setup.

All of this proved seemingly irrelevant, however, as AUBO independently purchased an industrial grip system from ROBOTIQ. Images for the ROBOTIQ gripper can be seen in **Figure 5**. The parts, discussed later, that would attach the camera and related sensors to the robot were designed around this grip.

Figure 5: ROBOTIQ Robotic Gripper, Purchased independently by AUBO

The last mechatronic sensor to be incorporated is the trip laser. Here, a laser light (cat toy) will be shone a photo-resistor. A photo-resistor acts as an electrical resistors whose value changes dependent on incident light. As such, the voltage across the resistor, which can be read by an arduino or computer, will be different when the laser is shining and not shining. If someone walks past the beam, the computer will register it and command the robot to begin the appropriate sequence. As of now, the general idea is that when a person is present, the robot will slow down. It will not return to its original functionality until the person exits the field and flips a switch, although a myriad of protocols can be adapted. For example, one idea originally considered is using a system of lasers to count the number of people present, fully automating the process.

Circuit diagrams for these systems can be found in **Appendix G.**

Furthermore, the system setup, which included the system enclosure as well as the attachments, was designed, finalized, and then implemented into the project work space. The system enclosure was designed using tetrix aluminum square beams. The design shown in **Figure 6** was built with space in mind for the manipulator and conveyor belt. The dimensions of the work cell were 1776 mm by 2044.38mm with a vertical height of 1268 mm. The system cage also incorporated an overhanging c-tunnel beam, whose purpose was to hold one of the cameras overhead the work area.

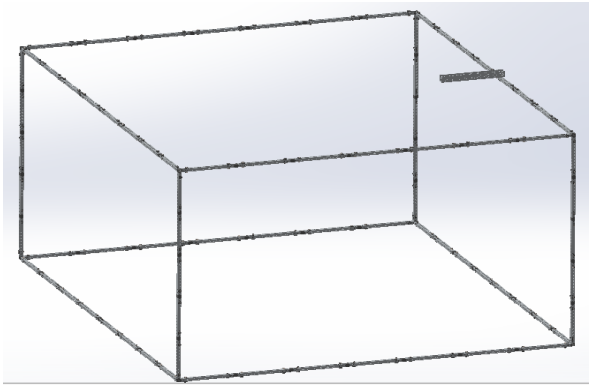


Figure 6: Above is the CAD model of the system enclosure using the tetrix pieces. CAD models of each piece could be found on the website.

The design of the individual beams also would allow for the the lidar system to be attached to the system enclosure. **Figure 7** shows one of the square beams up close.

Figure 7: Square beam utilized in the construction of system enclosure.

A CAD drawing for the overall structure can be found in **Appendix C**.

Attachments were designed to house the cameras as well as the distance and gyroscope sensors. The initial designs for these are pictured in **Figures 8** and **9** below.

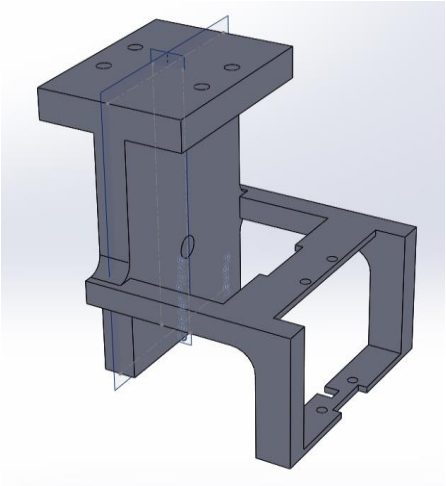


Figure 8: Initial Design for Camera Mount to Cage

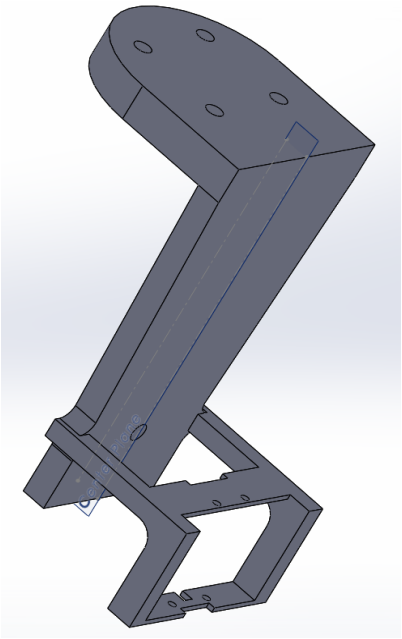


Figure 9: Initial Design for Camera Mount to Robot

The basic design of the Camera Mount for the cage involved suspending the whole apparatus from the C-tunnel beam by the two flanges seen at the top of the structure. The holes were designed to fit #6 screws (which Tetrix uses) and the spacing matched the tetrix C-tunnel beam dimensions. The overhang was designed to hold the sensors in place relative to the camera. The camera would face straight down, being bolted into (with a ¼ inch screw) the flat back part of the Mount. The gyroscope would be placed directly atop the camera; this is the upper cross beam. The time-of-flight distance sensor would be placed directly in front of the lens; this is the lower cross beam. Both beams had a piece at the back cut out so that wires could more easily be soldered to the sensor and so that the sensors would still lie flat with wire soldered in. The time-of-flight sensor, operating using a beam of light, also required a small gap at the front. Both were to be located in the center of their respective beam. Both were to be attached using M2.5 nuts and bolts. To prevent the metal bolt from potentially interfering with the sensor, small vinyl squares were cross-hatched and placed between the bolt head and the sensor board.

The basic design of the Camera Mount for the arm itself is similar, except instead of the two flanges at the top to attach to the Tetrix pieces, a circular piece was designed so that the mount could be placed between the ROBOTIQ robotic gripper. The holes were made for M6 screws and dimensions were made to line up with those found on the AUBO gripper and ROBOTIQ arm. Unfortunately, due to restrictions of communication between the computer, arm, and grip, the ROBOTIQ gripper was

ultimately not used, which will be discussed later.

Drawings for these parts can be found in **Appendix C**. To reduce clutter, three drawings are provided. Ttwo provide dimensions for the end attachments, the pieces that would actually hold the assembly to the cage or arm. The other provides dimensions for the overhang that would hold the sensors. The only dimension not provided is the fillet between the overhang the and end attachments. This fillet was 5mm. This detail is not listed because adjustments made to the pieces during construction rendered it unnecessary, as detailed below.

The initial plan was to 3D print these parts; as such, tolerances were not considered and the limitations of the 3D printer were used. The Innovative and Collaborative Studio (ICS) in the basement of Perkins Hall has small 3D printers that are open for students to use, within a certain length of the material. Past that length, the student would need to provide their own material. The printers are also smaller and more easily accessed, simplifying the logistics of printing. The material used was ABS.

Unfortunately, this proved to be a mistake as a majority of the prints failed. Refer to **Figure 10** for an example of a failed print for a different piece. In short, ABS is a very temperature dependent material. In it of itself, ABS is usable material (understandable as it is still used), but was less than ideal for the smaller, open air printers of the ICS. Fluctuations in the temperature of the room and material as well as the back and forth motion of the printer itself caused the model to bend as it was being printed. This issue was exacerbated by the raw amount of material required to support the overhang and the amount of time required for the print. In this vain, due to the the smaller, cage mount failing, no print was even attempted for the Robot Mount.

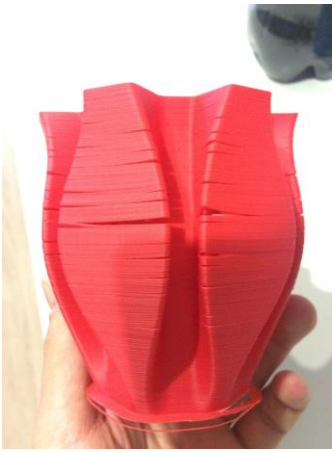


Figure 10: Example of Problems and Failures in 3D Printing with ABS

To combat this issue, several solutions were considered and tested. First, was orientation. The mount was rotated not so that the bottom lay flat, but so that it was angled at 45 degrees. Refer to **Figures 11** and **12** for examples of the two print orientations.

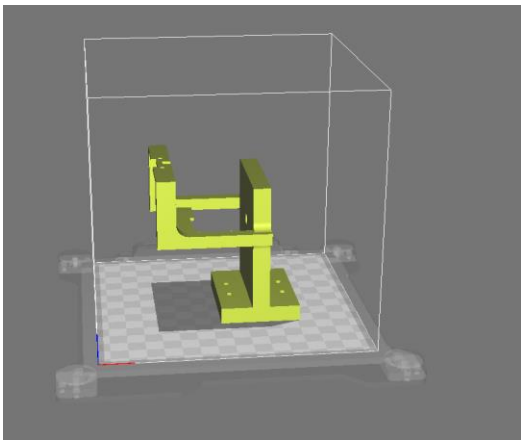


Figure 11: Initial Print Orientation

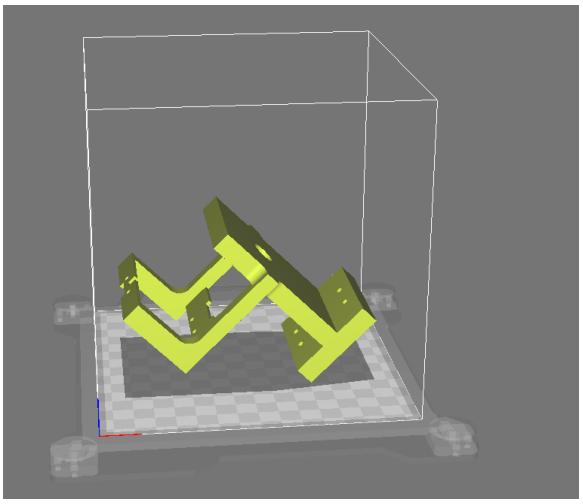


Figure 12: Modified Print Orientation to Reduce Time and Support Material

However, neither of these worked. From there, two primary options presented themselves. First was to use a nicer, enclosed printer. However, already being in the ICS, the second option was much more prevalent. Said option was to split the piece into two, making the sensor overhang via 3D printing and to make the external mounts out of wood.

The sensor overhang piece was small enough, required less support material, and printed fast enough that the shortcomings of ABS were overcome. The process still failed at least once, however, and some parts of the final prints still had to be gorilla glued together. The external mounts were made by hand by the students in the ICS woodshop. The geometry was simple enough that even relative amateurs could fabricate the pieces. The overhang and mounts were ultimately epoxied together to make the final pieces shown in **Figures 13** and **14.**

Figure 13: Cage Camera Mount

Figure 14: Robot Camera Mount

However, there were a few final issues with these pieces. Start with the fact that the final Robot Camera Mount, **Figure 14**, does not match what was designed, **Figure 9**. That is because **Figure 9** was made under the assumption the group would be using the ROBOTIQ gripper. However, communicating with the ROBOTIQ gripper proved impossible and the group defaulted back to using the parallel gripper. However, this meant the whole design had to be changed. Unfortunately, this difficulty arose fairly late in the semester, with little time to fully model or design a new mount.

The cage mount was also unusable. It was fabricated under the assumption that it would hang from the extended C-channel, which would be perpendicular to the conveyor belt/table. This would mean the camera could hang straight down and rely on it's wide angle to see the whole of the table. However, as seen and discussed later, the C-channel would extend parallel to the table, putting the camera at the end. This would mean it would need to be angled. Misunderstanding on the fabricator's (Benjamin Terry) part and miscommunication led to the error going unnoticed. A quick modification using the parts on hand was attempted, but it was unstable, an issue worsened by the cage's instability.

The last mechatronic piece that was investigated was a printed circuit board (PCB) to wire all the sensors and controls together in an orderly, centralized fashion. The process was attempted by hand by the students. The process involved inscribing a circuit design onto a piece of copper laminate; this could be achieved by drawing directly on the laminate with sharpie or ironing it on. From there, the board was submerged in Ferric Chloride, ideally heated. Ideally, the piece would be cut to size first, so as not to waste material. However, drawing the circuit by hand and inexperience resulted in a more dispersed design, and an entire plate was used. For a small piece in heated Ferric Chloride, the etching process takes about 10 minutes. For a large piece in room temperature Ferric Chloride (what the students used), the process takes about 90 minutes. Once etched, remove the "guard ink" any way necessary. For sharpie, it was found that toothpaste is a fairly effective removant. Refer to **Figure 15** and **16** for a print that was not soaked long enough and successful etching, respectively.

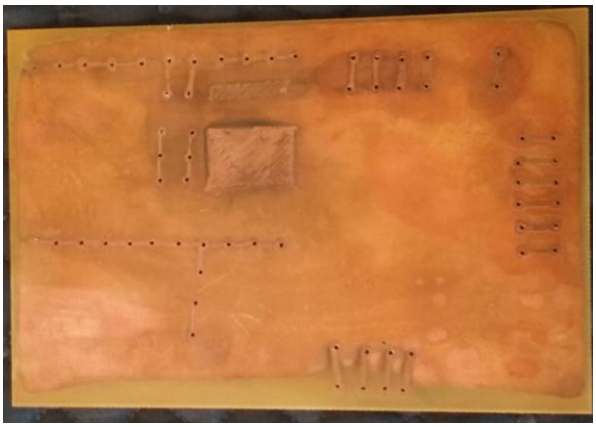


Figure 15: Failed PCB Etch; Not Soaked Long Enough

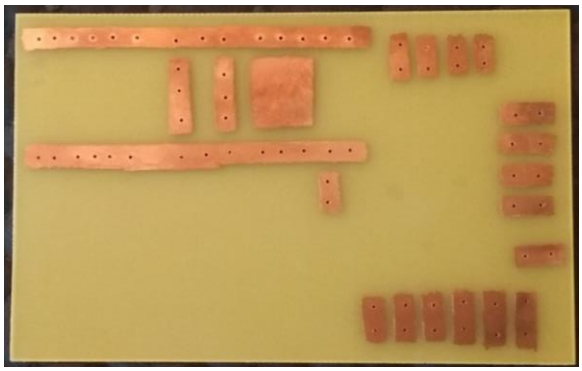


Figure 16: Successful PCB Etch

Once etched, holes were drilled for wire to be fed through and soldered in using micro "drill bits", shown in **Figure 17.** Technically, because of their design, these are not drill bits, but they work they same. Of note is that drilling the holes is difficult using a hand drill. It is suggested that a drill press be used instead. Also, the bits are liable to break, as seen.

Figure 17: Micro Drill Bits for PCB

Software and Programming

AUBO robotic arm is controlled by a vision recognition software that provides the (x,y,z) coordinates of a particular object. The program output serves as the input of an .aubo script file which will allow to manipulate the arm at the desired location.

The vision recognition software is written in C++ and uses the OPENCV library. For object recognition, arauco markers have been chosen and assigned to different types of objects. Every arauco market will contain the information of the dimensions and particular features of the object. The marker/s will be placed on the object at a strategic locations where the camera will be able to detect it.

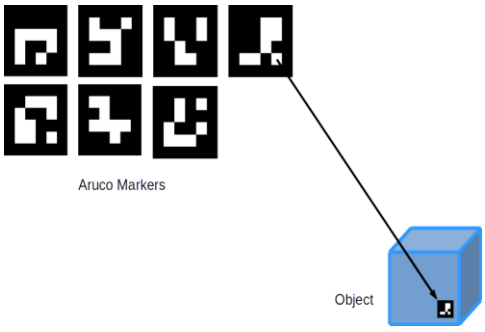
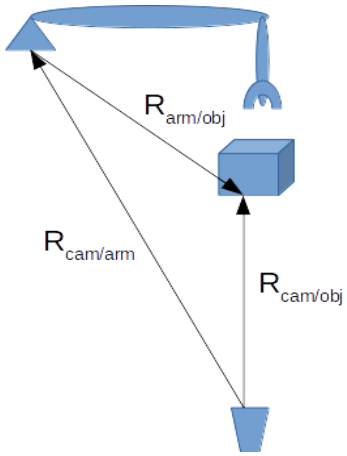


Figure 18: Pictured above is an example of the Aruco Markers utilized for this project.

The vision recognition program will be able to detect in real time the location of one or several object and provide the relative position (x,y,z) of the object/s. This is done by doing a simple vector subtraction of the location of the camera with respect to the object and the location of the arm base with respect to the camera.



$$R_{arm/obj} = R_{cam/obj} - R_{cam/arm}$$

Figure 19: The above image is of the Vector Representation for Object Location.

The (x,y,z) relative position of the object will be passed to a running aubo script, which will contain code lines to move the robotic arm at the provided location and to operate the gripper for a particular object.

The gripper and security system are controlled by an arduino DUO microcontroller. The gripper will open and close according to the readings obtained by the force sensor attached to the gripper and

the security system code line output. The arduino DUO itself must be programmed in order to operate the gripper. Its code line script (.ino file) will be operational with the object recognition software and the aubo script allowing full control of the AUBO arm in real time.

With all of the individual systems in place and working, the next step would be integrating them with the robot. The first inclination was to use the I/O control ports and the modbus system in the control box using a USB connection. To do this we attempted to use an open source library called libUSB which contains functions which should be able to send serial outputs from a personal computer as well as receive and interpret signals sent to it. This was deemed too time intensive as it would require us to implement the individual protocols specific to the AUBO control box. We also considered purchasing a software package known as Modbus Driver. This package includes libraries which would allow us to communicate with the control box using modbus protocols. This was also determined to be very time intensive as it would require us to learn to use this new set of libraries as well as implement them effectively. In addition, the cost of purchasing this software package would be rather expensive. In addition, potentially tied to recent software updates, the I/O connections from the current control box failed to function properly.

Fortunately, we were given access to the program AUBO Script which allows us to create control scripts for the robot from our personal computers. This program also comes built in with the ability to communicate with the control box using the TCP/IP protocols. With this program we were able to upload a control script to the robot using a direct ethernet connection to the control box. The ability to communicate with the robot now gives us the ability to integrate the vision system into the control of the robot. The materials we received about the use of AUBO Script contained information pertaining to this system integration. This material provided several potential setups for the camera, image interpretation, and use of the output positions, however, not every setup was able to be used in this situation. A general schematic of the setup we plan on using can be seen in **Figure 20**. We chose this form of integration because it allows us to interpret the image from the personal computer as well as feed the position information directly to the AUBO Script program.

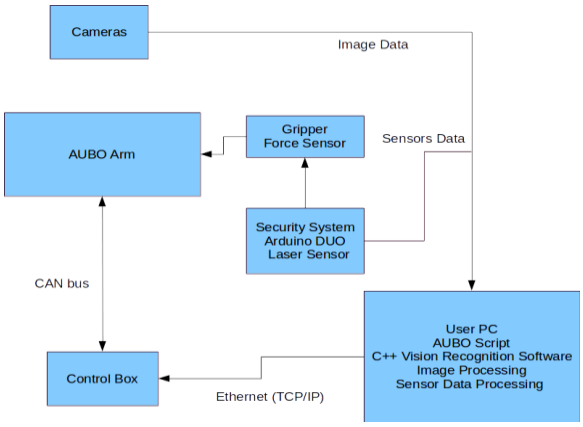


Figure 20: Software and Hardware Architecture.

CONCLUSIONS AND RECOMMENDATIONS

Mechanical Parts

In conclusion, there are various components to this project that were completed. The schematic below, **Figure 21**, illustrates the finalized setup for this project. The finalized setup includes the robotic arm, conveyor belt, safety system, and the vision attachments, and the work cell that was created using aluminum beams purchased from Tetrix Robotics. However, the stability of the cage did not meet the set expectations. Because of this additional tetrix pieces were utilized temporarily to make the structure more rigid. In addition to the original tetrix pieces, wood was used to help support and stabilize the vision attachment. **Figures 22 and 23** show the work cell without and with the necessary support, respectively. It is a recommendation to incorporate a more reliable and stable component, such as wooden trusses that cover the entire side of the structure, to the work cell. By doing so, one could provide the necessary support and stabilize the structure, making it more feasible to attached, incorporate the laser/lidar safety system

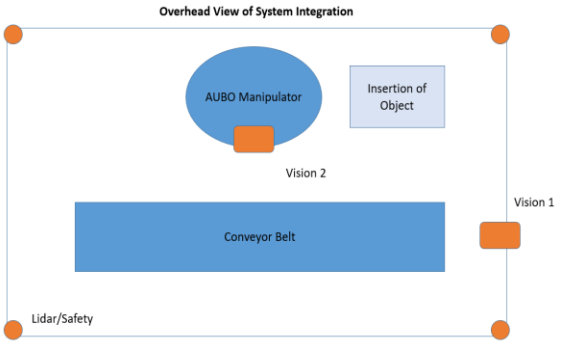


Figure 21: Pictured above is the finalized design for the system setup.

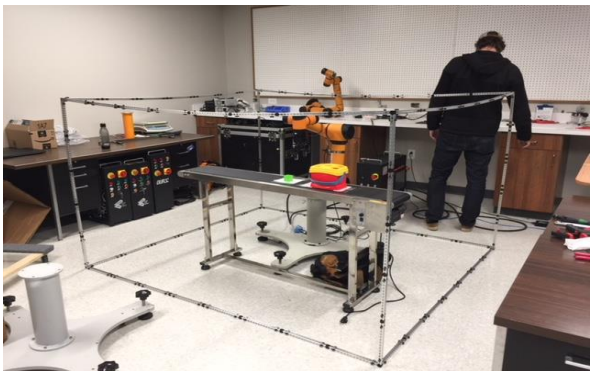


Figure 22: Pictured above is an image of the work cell without additional support.

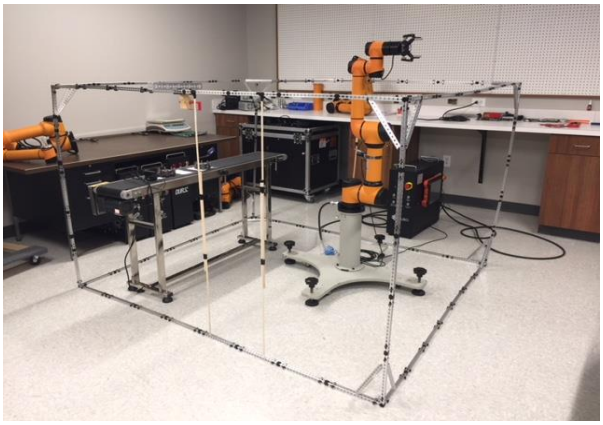


Figure 23: Pictured above is an image of the work cell with additional tetrix beams and wooden sticks for assistance.

Next, consider the more mechatronic elements of the systems: grippers, lasers, mounts, sensors, etc. The biggest here was mechanical integration. Individually, the parts worked perfectly. The mechanical grip could be controlled and the photoresistor/tripwire concept worked well. Putting the two systems together, apart from the rest of the robot, worked well. The various sensors worked well. Jose, the primary investigator for the openCV vision recognition, had a script that could take input from these sensors running in arduino.

The trouble came from fabricating the pieces to tie all of these things together. First, the mounts for the cameras and relevant sensors was to be 3D sensors. While good in theory (3D printing/rapid prototyping is still a very useful tool), issues arose. Namely, ABS was not the appropriate material given the circumstances chosen. Said circumstances--using smaller, simpler 3D printers found in the ICS--were chosen because of ease. Walking in and printing something in the ICS was less time consuming than requesting use of larger printers. And using larger printers seemed excessive for relative small parts. In hindsight, however, the extra time and effort to use a large-scale, nicer printer would have been worth it and the initial plan to 3D parts would have likely succeeded.

As it stands, this does raise an interesting question: why continue to work with the ICS after the initial failed print instead of using a nicer printer? The simple answer is proximity. The student, Benjamin Terry, was already working in the ICS, which also has a woodshop. So, when presented with the failed print, his thoughts went to potential solutions immediately available.

Regardless, the parts were made, though they themselves had issues. First is the misunderstandings. The two parts fabricated did not meet was required. For the cage mount, this arose from misunderstandings over where the camera needed to be. For the robot mount, this arose from difficulty in working with the ROBOTIQ gripper. For the cage mount, the failure of the piece can be wholly attributed to internal, team/member issues. For the robot mount, that is merely unfortunate timing; though perhaps the problem could have been discovered and handled sooner had the initial mounts been fabricated sooner.

Last, incorporating all the pieces via PCBs and some form of mold did not come to fruition for two reasons. First is timing. Work on the mounts, extended due to failure, used more time than expected. Second, and more crucially in hindsight, the system was not fully considered to begin. That the laser and mirrors for the safety system would be incorporated into the Tetrix enclosure was assumed from the start. Exactly how was not considered, however. Similarly, the PCB was considered, but how it, the arduino, and other such components would attach and where, were not considered. This made attempting the problem difficult and the time spent on the camera mounts (which also ultimately failed) meant there was less time to solve it.

Software and Programming

In conclusion, the software written for this project was able to complete several of the tasks that were initially requested. The robotic vision system developed was able to accurately discern the differences between specific aruco codes and carry out certain actions based upon which ones were visible. The code was also able to effectively communicate with the Aubo-i5 robot arm using a TCP/IP connection and pass it the coordinates it needed to move to. In order to determine which joint positions to send to the arm, the cartesian coordinates of the blocks had to be converted to the radial positions of each of the joints. This was done using the equations found in **Figure 26**. The viable joint values in which the gripper was pointing in the correct direction along with the

positions which cannot be reached by the arm are visible in **Figures 27-30**. A serial connection was also set up with the Arduino which controlled the gripper. This was used to tell the Arduino when to open and close the gripper.

Some issues encountered in the development of this project were that the delay in the communication to the Arduino based gripper made it so that picking up moving objects would not be possible. It is recommended to switch to a TCP/IP connection to prevent this delay. Another issue we faced was that the webcam we used was not accurate enough to determine the exact positions of the aruco codes. The calibration data collected from the camera can be seen in **Figure 24** and it represents the position read by the camera with respect to the camera's actual distance from the code being read. It is recommended that in future builds multiple cameras are used to determine more accurate positions of these codes. One final issue encountered in the development of this software was with the Aubo Script which was used to control the robot arm itself. This control software, while useful, did not come without bugs. Some functions such as linear movement with respect to the tool head and direct velocity control were non-functional. A few of these missing functions are integral to the task of picking up moving objects. If another group is to work on this project in the future, they would benefit from waiting for an update to the Aubo Script software.

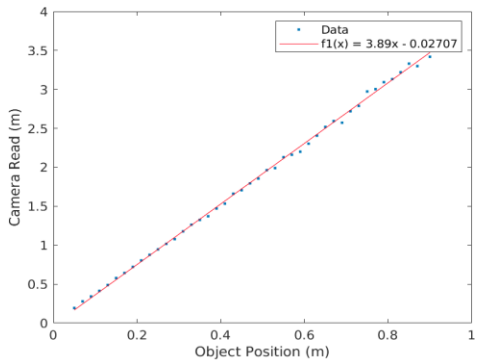


Figure 24: Linear fit to obtain calibration factor

$$calfactor = 0.2571 \quad objPos/camRead$$

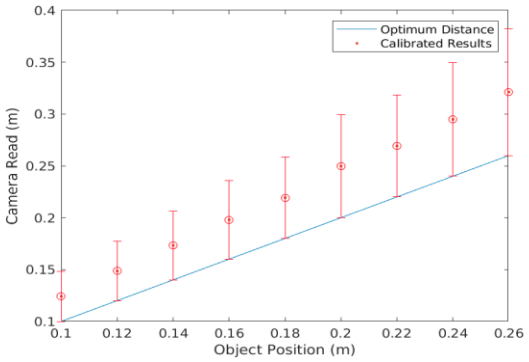


Figure 25: Calibrated camera distance

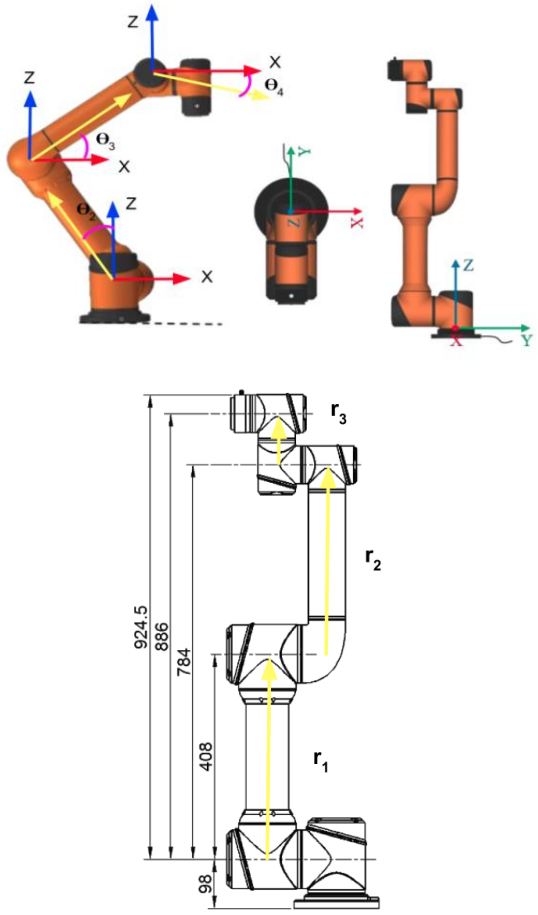


Figure 26: Joint coordinate system used during derivation of joint-angle equations

$$R = (x, y, z)$$
$$r_1 = 0.408\ m$$
$$r_2 = 0.376\ m$$
$$r_3 = 0.102\ m$$

$$\theta_{2,abs} = cos^{-1}\left[\frac{x - r_3 - r_2 cos(\theta_2)}{r_1}\right] \quad (1)$$

$$\theta_{3,abs} = sin^{-1}\left[\frac{r_1 sin(\theta_1) - z}{r_2}\right] \quad (2)$$

$$\theta_1 = tan^{-1}(y/x) \quad (3)$$

$$\theta_2 = -(90^o - \theta_{2,abs}) \quad (4)$$

$$\theta_3 = \theta_{2,abs} + \theta_{3,abs} \quad (5)$$

$$\theta_4 = -(90^o - \theta_2 + \theta_3) \quad (6)$$

$$\theta_5 = angle_{coeff} \quad (7)$$

$$\theta_6 = angle_{coeff} \quad (8)$$

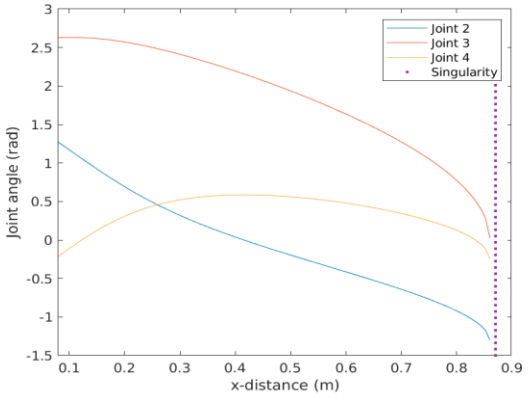


Figure 27: Joint angular distribution at constant z-distance = 0.2m from Joint 1.

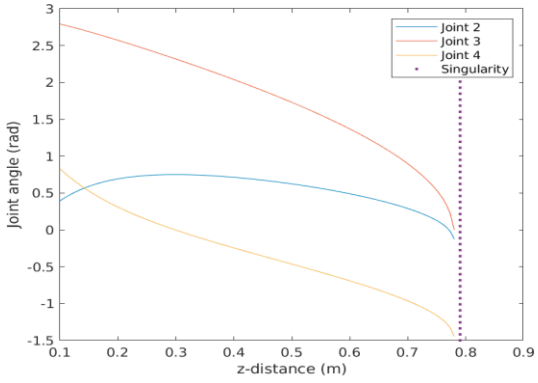


Figure 28: Joint angular distribution at constant x-distance = 0.2m from Joint 1.

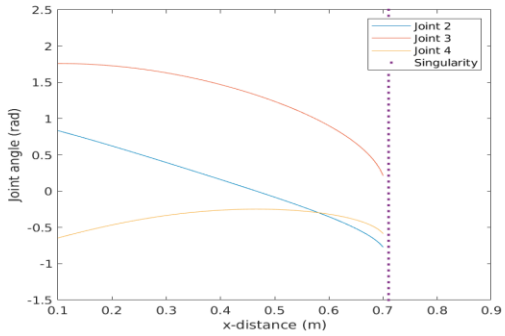


Figure 29: Joint angular distribution at constant z-distance = 0.5m from Joint 1.

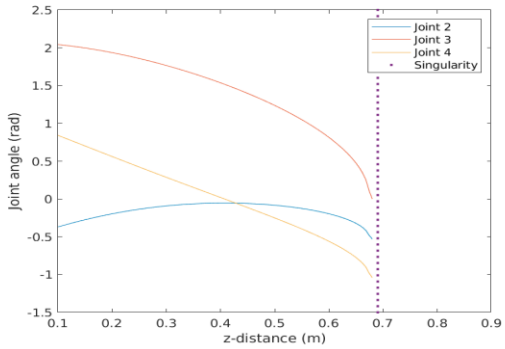


Figure 30: Joint angular distribution at constant x-distance = 0.5m from Joint 1.

In conclusion, looking back at the initial goals, a vision system with a mechanical gripper was implemented. However, the ultimate system was simplified from the original plan due to mechanical and electrical shortcomings. A safety system was not implemented, though fabricated and functional as a separate system.

REFLECTION AND LESSONS LEARNED

Lessons learned for the duration of this project, semesters Fall 2017 and Spring 2018, can be summed into multiple categories. The first being the material learned or, in better terms, the learning curve that was conquered as well as the project management component with a focus on time management. At the beginning of the fall semester, the team worked with setting out a timeline to complete tasks. This was important in the process because it laid out a fairly detailed plan of action. However, there was an issue in being realistic with the timeline. Some particular factors were not taken into consideration when forming the tentative schedule, thus, the team was unable to meet all of the goals set out for this semester. Focusing more on realistic planning at the beginning of the spring semester was hoped to make the team more successful and help with decision making. For example, instead of building the conveyor belt, as originally planned, it proved more justifiable to purchase a conveyor belt utilizing the budget. This way time that would have been geared toward research, building, debugging, and optimizing was put toward other, more prevalent matters such as debugging, integrating systems, etc. This was the correct move, given the difficulties with the other systems. Moreover, when discussing the amount of research and learning that had to be completed in the beginning period of this project, it was evident receiving a better understanding of the robot itself and other relevant components was a top priority. This should have also been taken into consideration when forming the schedule. The team underestimated the amount of research/time it would take to come to conclusive decisions and/or answers.

For the spring semester, timing was more realistic, however, the factor of ordering, shipping, and receiving of parts for this

project was not properly taken into account. One of our biggest challenges was receiving parts on time. Because of this certain tasks were delayed and set back the original timeline. Furthermore, there was an issue with the the communication of the robotic arm and the computer, which held and sent the code. Because of this there were delays, which hindered our ability to reach the next milestone, which included incorporating the dynamic component of this project. Last, failure and difficulties arose in fabrication of some of the mechanical designs, pushing the schedule perpetually back. In conclusion, there were some aspects of this project that were unforeseen and as result, ultimately, delayed the progress of the completion of the project.

Benjamin Terry

I was responsible for a couple of things. In the fall, I was tasked with handling the gripper, tripwire, and related sensors. I also helped make the Bill of Materials and keep track of purchases. Overall, I'd say I did decent work there; definitely a good start, if overall not that complex. In the spring, I was tasked with building mounts for the cameras and various sensors and ultimately wiring everything together. This, as discussed, went less that ideally. But what did I take away from it? For the fall semester, it was my first year foray into mechatronics and tech (along with a few other classes); about time, since that's the type of work I think I want to do. It went well, reaffirmed my desires, and showed me some of the things I still need to work on.

But spring semester is where things really went wrong and I really gained some insights. First, my time management was atrocious. Part of that was assuming 3D printing would be faster (which it's supposed to be, as rapid prototyping is a key advantage), but overall it was a shortcoming on my part. Of course, being exposed to these shortcomings, both personal and technical, means I am more aware of them going forward. Speaking of technical shortcomings, this project also strengthened a lot of external skills. We've done some of these things in classes, like 3D printing, but this was more, and exposed me to some things I had never seen in class. For example, I dove deeper into some of the shortcomings of 3D printing, not discussed in class. I explored fabrication (woodworking and PCB etching), something never mentioned or practiced on in class. Last I learned more about systems design. This largely goes back to not designing all the pieces as a whole at the start of the spring semester. I think treating the enclosure as one thing and the rest as attachments was a poor planning on our part.

NEXT STEPS (Fall 2018)

Looking forward, the next team that works on this project will have to focus on incorporating the dynamic component to the system. They will also have to work on the safety system and making sure to integrate it into the system as well. Furthermore, the team will have to achieve the functionality part of this project so that it is complete and they are able to optimize and deliver a finished product.

Appendix A: Work Breakdown

Schedule with Milestones (Fall 2017)

Pictured below are screenshots from the team's gantt chart. It includes prospective completion dates, as well as, a complete breakdown of steps that need to be taken to make this project a success. The main milestones for the fall semester included: making a decision on what object/toy would be used for this experiment, completing research to figure out what parts would be needed and/or appropriate for this project, ordering all necessary components, properly implementing/integrating them into the system.

Task Name	Duration	Start	Finish
Gripper	**13 days**	**Wed 9/20/17**	**Fri 10/6/17**
Pneumatic Actuator	12 days?	Wed 9/20/17	Thu 10/5/17
Finalize Pneumatic Actuator	1 day	Fri 10/6/17	Fri 10/6/17
System Setup	**13 days**	**Wed 9/20/17**	**Fri 10/6/17**
Parameters/Rec			
Sketch System Setup Options	7 days	Wed 9/20/17	Thu 9/28/17
Finalize System Setup	1 day?	Mon 10/2/17	Mon 10/2/17
Setup System	4 days	Tue 10/3/17	Fri 10/6/17
Programming	**17 days**	**Wed 10/11/17**	**Thu 11/2/17**
Object Recognit	16 days	Wed 10/11/17	Wed 11/1/17
Tracking	16 days	Wed 10/11/17	Wed 11/1/17
Sufficient Object Recognition and Tracking	1 day?	Thu 11/2/17	Thu 11/2/17

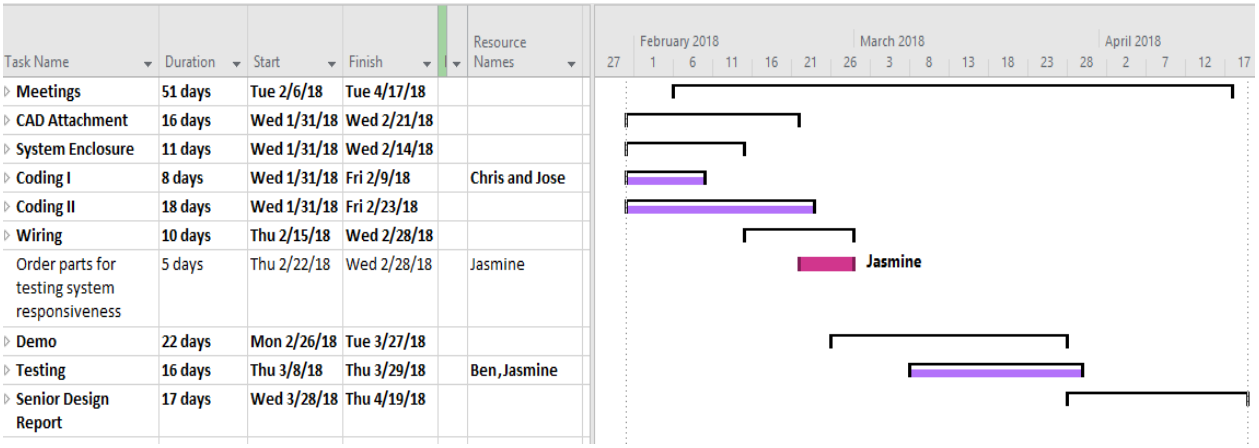
Task Name	Duration	Start	Finish
Fall Semester	**53 days**	**Wed 9/20/17**	**Fri 12/1/17**
Pick/ Order Toy	1 day	Fri 9/22/17	Fri 9/22/17
Camera	7 days	Wed 9/20/17	Thu 9/28/17
Software	0 days	Wed 9/20/17	Wed 9/20/17
Proximity Sensors	**7 days**	**Mon 9/25/17**	**Tue 10/3/17**
Requirements	2 days	Mon 9/25/17	Tue 9/26/17
Research Proximity Sensors	4 days	Wed 9/27/17	Sun 10/1/17
Finalize Proximity Sensors and Purchase	1 day	Mon 10/2/17	Mon 10/2/17

Task Name	Duration	Start	Finish
1st Milestone Reached	0 days	Thu 11/2/17	Thu 11/2/17
Integrate Actual Parts	**18 days**	**Mon 11/6/17**	**Wed 11/29/17**
Programming			
Adjustments to System Setup			
Force/Torque Sensors	**18 days**	**Mon 11/6/17**	**Wed 11/29/17**
Research Force/Torque Sensors			
Finalize Force/Torque Sensors			

Schedule with Milestones (Spring 2018)

Pictured below are screenshots from the team's gantt chart. It includes prospective completion dates, as well as, a complete breakdown of steps that need to be taken to make this project a success. To make this spreadsheet better, we also attached names and notes to tasks. Furthermore,

tasks were more detailed in description. The main milestones for the spring semester included: coding and additional research, ordering all necessary components, properly implementing/integrating them into the system, and building/constructing mechanical components.



Task Name	Duration	Start	Finish	Resource Names
Meetings	51 days	Tue 2/6/18	Tue 4/17/18	
CAD Attachment	16 days	Wed 1/31/18	Wed 2/21/18	
System Enclosure	11 days	Wed 1/31/18	Wed 2/14/18	
Coding I	8 days	Wed 1/31/18	Fri 2/9/18	Chris and Jose
Coding II	18 days	Wed 1/31/18	Fri 2/23/18	
Wiring	10 days	Thu 2/15/18	Wed 2/28/18	
Order parts for testing system responsiveness	5 days	Thu 2/22/18	Wed 2/28/18	Jasmine
Demo	22 days	Mon 2/26/18	Tue 3/27/18	
Testing	16 days	Thu 3/8/18	Thu 3/29/18	Ben,Jasmine
Senior Design Report	17 days	Wed 3/28/18	Thu 4/19/18	

Pictured above is an overview of project timeline.

Task Name	Duration	Start	Finish	F	Names
CAD Attachment	**16 days**	**Wed 1/31/18**	**Wed 2/21/18**		
Camera Assembly I	**6 days**	**Wed 1/31/18**	**Wed 2/7/18**		**Ben**
Design and Model attachment for the Vision	6 days	Wed 1/31/18	Wed 2/7/18		Ben
Print piece for this	6 days	Wed 1/31/18	Wed 2/7/18		Ben
Refine CAD model for attachment	6 days	Wed 1/31/18	Wed 2/7/18		Ben
Camera Assembly II	**5 days**	**Thu 2/8/18**	**Wed 2/14/18**		**Ben**
Design and Model gripper	5 days	Thu 2/8/18	Wed 2/14/18		Ben
Print piece for attachment	5 days	Thu 2/8/18	Wed 2/14/18		Ben
Refine Camera Assembly	5 days	Thu 2/8/18	Wed 2/14/18		Ben

Task Name	Duration	Start	Finish	F	Resource Names
Camera Assembly III	**5 days**	**Thu 2/15/18**	**Wed 2/21/18**		**Ben**
Finish Attachme	5 days	Thu 2/15/18	Wed 2/21/18		Ben
Integrate Camera Assembly to overhang work	5 days	Thu 2/15/18	Wed 2/21/18		Ben
System Enclosure	**11 days**	**Wed 1/31/18**	**Wed 2/14/18**		
System Enclosure I	**6 days**	**Wed 1/31/18**	**Wed 2/7/18**		**Jasmine**
Finalize dimensions for	1 day	Fri 2/2/18	Fri 2/2/18		Jasmine
Create a schematic	4 days	Sat 2/3/18	Wed 2/7/18		Jasmine
Order parts for enclosure	1 day	Tue 2/6/18	Tue 2/6/18		Jasmine
System Enclosure II	**5 days**	**Thu 2/8/18**	**Wed 2/14/18**		**Jasmine**
Build System Enclosure	5 days	Thu 2/8/18	Wed 2/14/18		Jasmine

Appendix B: Bill of Materials

The following is a screenshot of the bill of materials for this project. The bill of materials includes the budget as well as the part count for materials purchased for this project. All items have corresponding links, costs, and descriptions within spreadsheet.

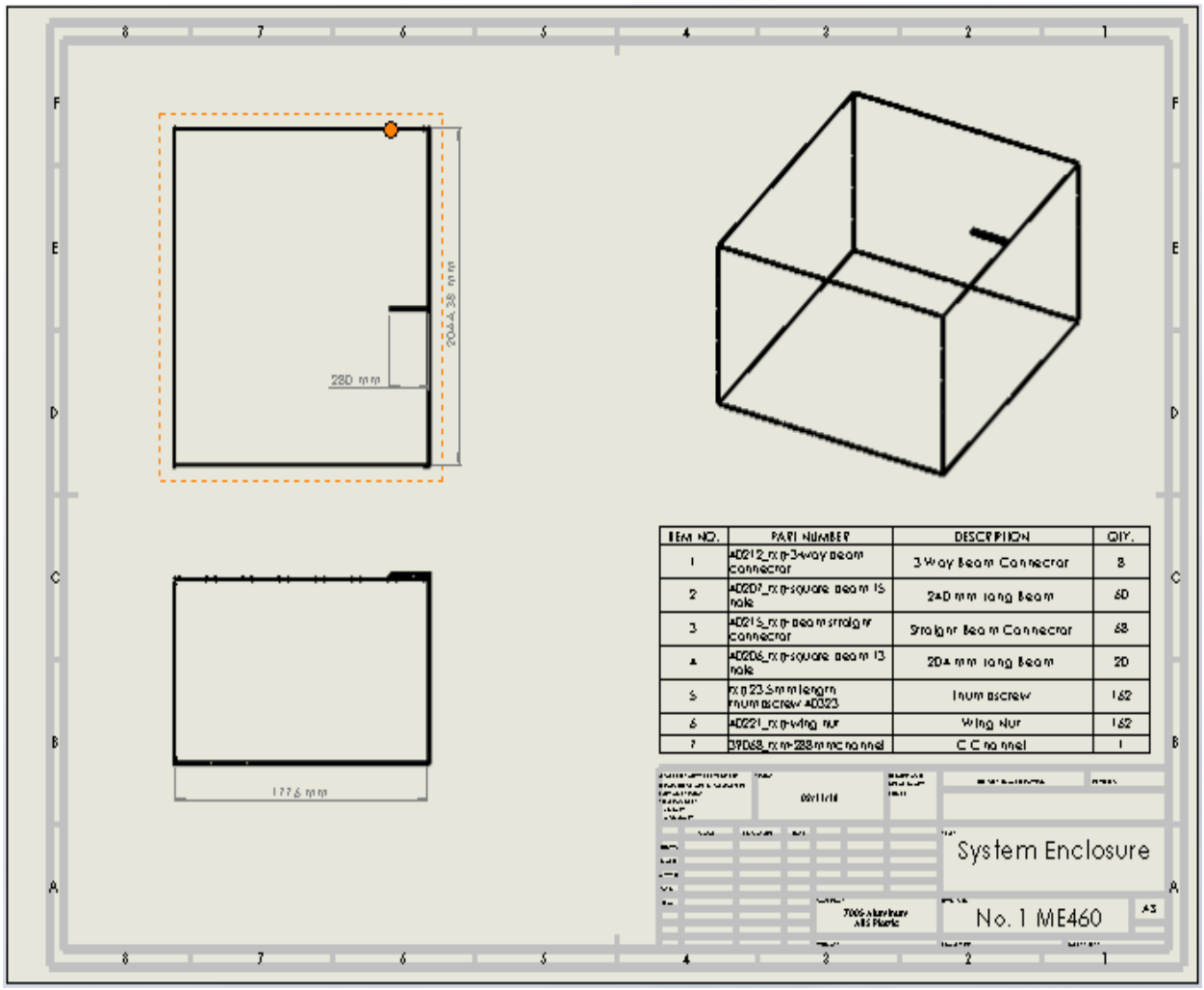
Bill of Materials			
	Project: Aubo Robotics		
	Advisor: Dr. Jindong Tan		
	Course: ME450/460		
	Cost:		
	Initial Budget	Materials:	$121.77
	$5,000.00	Mechanical:	$1,590.85
		Software and Related:	$222.28
	Remaining	Electrical and Sensors:	$375.06
	$2,690.04	**Total**	$2,309.96
	Part Count:		
		Materials:	8
		Mechanical:	11
		Software and Related:	5
		Electrical and Sensors:	13
		Total	37

Link to spreadsheet:
https://docs.google.com/spreadsheets/d/1sGuwYoi6w41bW_Qs5Qa0oNcsMnsE2yJ3QFXYwl5N7rE/edit?usp=sharing

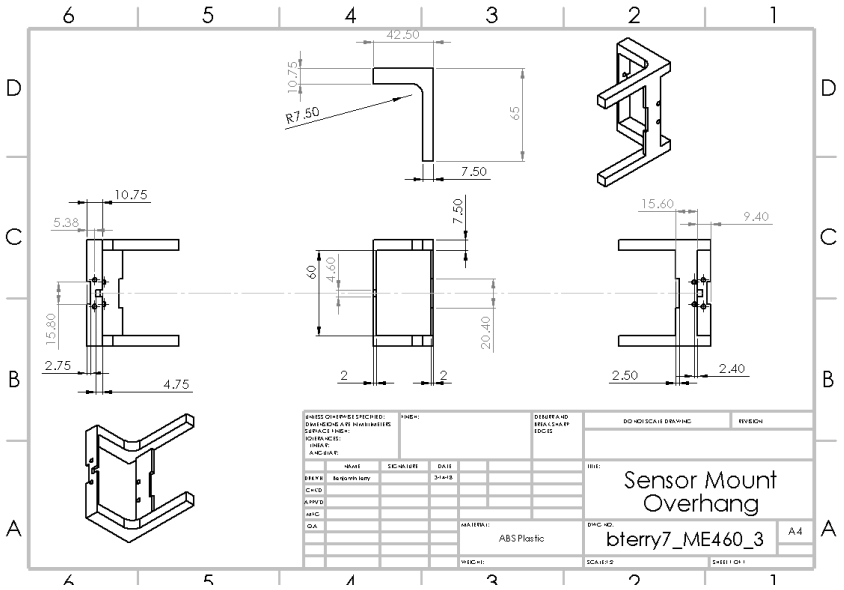
Link will only work for University of Tennessee associated gmail accounts i.e. "@vols.utk.edu".

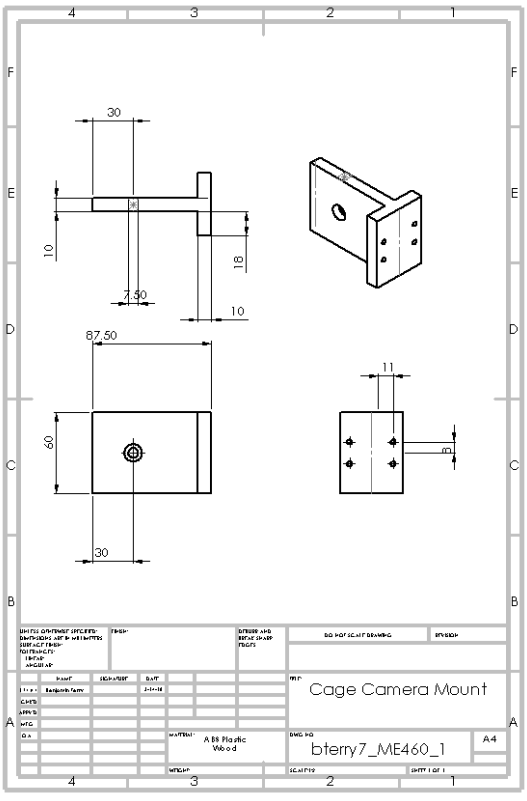
Appendix C: CAD Drawings

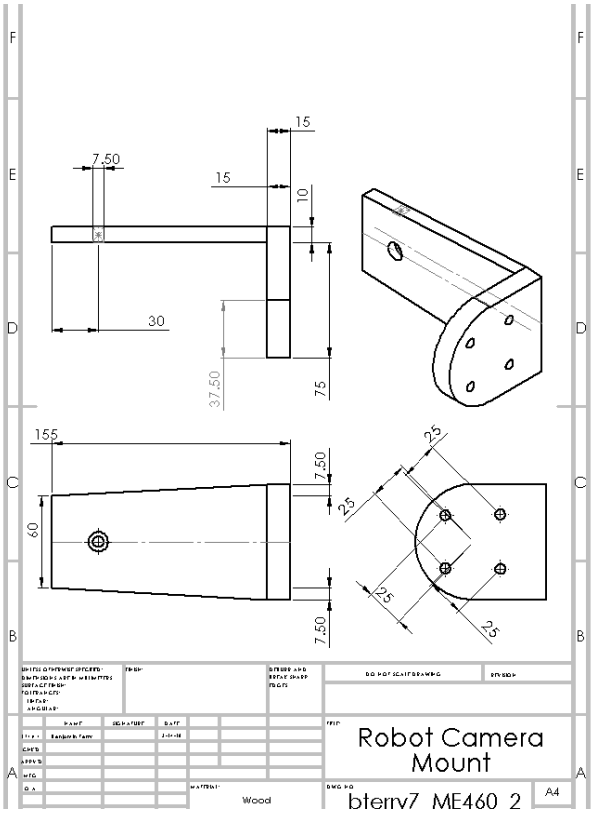
Pictured below is the CAD drawing for the system enclosure.



Pictured below is the CAD drawings for the camera mounts


42.50
10.75
R7.50
65
7.50
10.75
5.38
15.80
2.75
4.75
7.50
60
4.60
20.40
2
2
15.60
9.40
2.50
2.40
Sensor Mount Overhang
ABS Plastic
bterry7_ME460_3
A4


30
10
18
7.50
10
87.50
60
30
11
8
Cage Camera Mount
A Bit Plastic Wood
bterry7_ME460_1
A4


15
7.50
15
10
30
37.50
75
155
7.50
60
7.50
25
25
25
25
Robot Camera Mount
Wood
bterry7_ME460_2
A4

Appendix D: Visual Recognition Code Line (C++)

track_marker.cpp

```
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/aruco.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/ccalib.hpp>
#include <sstream>
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <cmath>
#include <tgmath.h>
using namespace std;
using namespace cv;


const float calibrationSquareDimensions = 0.01905f; //meters
const float arucoSquareDimension = 0.0528f; //meters
const Size chessboardDimensions = Size(6,9);

/*
void createArucoMarkers()
{
	Mat outputMarker;
	Ptr<aruco::Dictionary> markerDictionary =
aruco::getPredefinedDictionary(aruco::PREDEFINED_DICTIONARY_NAME(0)); //aruco::DICT_4X4_50

	for(int i; i < 50; i++)
	{
		aruco::drawMarker(markerDictionary,i,500,outputMarker,1);
		ostringstream convert;
		string imageName = "4X4Marker_";
		convert << imageName << i << ".jpeg";
		imwrite("vv.jpeg", outputMarker);//convert.str()
	}


}
*/


void createKnownBoardPosition(Size boardSize, float squareEdgeLength, vector<Point3f>& corners)
{
	for(int i; i < boardSize.height; i++)
	{
		for(int j = 0; j < boardSize.width; j++)
		{
			corners.push_back(Point3f(j * squareEdgeLength, i * squareEdgeLength, 0.0f));
		}
	}
}


void getChessboardCorners(vector<Mat> images, vector<vector<Point2f> >& allFoundCorners, bool showResults = false)
{
	for(vector<Mat>::iterator iter = images.begin(); iter != images.end(); iter++)
	{
		vector<Point2f> pointBuf;
```

```
			bool found = findChessboardCorners(*iter, Size(9,6), pointBuf, CV_CALIB_CB_ADAPTIVE_THRESH |
CV_CALIB_CB_NORMALIZE_IMAGE);

			if(found)
			{
				allFoundCorners.push_back(pointBuf);
			}

			if(showResults)
			{
				drawChessboardCorners(*iter, Size(9,6), pointBuf, found);
				imshow("Looking for Corners", *iter);
				waitKey(0);
			}
		}

}


void cameraCalibration(vector<Mat> calibrationImages, Size boardSize, float squareEdgeLength, Mat& cameraMatrix, Mat&
distanceCoefficients)
{
		vector<vector<Point2f> > checkerboardImageSpacePoints;
		getChessboardCorners(calibrationImages, checkerboardImageSpacePoints, false);

		vector<vector<Point3f> > worldSpaceCornerPoints(1);

		createKnownBoardPosition(boardSize, squareEdgeLength, worldSpaceCornerPoints[0]);
		worldSpaceCornerPoints.resize(checkerboardImageSpacePoints.size(), worldSpaceCornerPoints[0]);

		vector<Mat> rVectors, tVectors;
		distanceCoefficients = Mat::zeros(8, 1, CV_64F);

		calibrateCamera(worldSpaceCornerPoints, checkerboardImageSpacePoints, boardSize, cameraMatrix,
distanceCoefficients, rVectors, tVectors);

}

bool saveCameraCalibration(string name, Mat cameraMatrix, Mat distanceCoefficients)
{
		ofstream outStream(name);
		//outStream.open;// ("cal_coeff.txt");
		if(outStream)
		{
			uint16_t rows = cameraMatrix.rows;
			uint16_t columns = cameraMatrix.cols;

			outStream << rows << endl;
			outStream << columns << endl;

			for(int r = 0; r < rows; r++)
			{
				for(int c = 0; c < columns; c++)
				{
					double value = cameraMatrix.at<double>(r, c);
					outStream<< value << endl;

				}
			}

			rows = distanceCoefficients.rows;
			columns = distanceCoefficients.cols;

			outStream << rows << endl;
			outStream << columns << endl;

			for(int r = 0; r < rows; r++)
			{
				for(int c = 0; c < columns; c++)
				{
```

```
                                        double value = distanceCoefficients.at<double>(r, c);
                                        outStream<< value << endl;

                                }
                        }

                        outStream.close();
                        return true;
                }

                return false;
}


bool loadCameraCalibration(string name, Mat& cameraMatrix, Mat& distanceCoefficients)
{
                ifstream inStream(name);
                if(inStream)
                {
                        uint16_t rows;
                        uint16_t columns;

                        inStream >> rows;
                        inStream >> columns;

                        cameraMatrix = Mat(Size(columns, rows), CV_64F);

                        for(int r = 0; r < rows; r++)
                        {
                                for(int c = 0; c < columns; c++)
                                {
                                        double read = 0.0f;
                                        inStream >> read;
                                        cameraMatrix.at<double>(r,c) = read;
                                        cout << cameraMatrix.at<double>(r,c) << "\n";
                                }
                        }
                        // Distance Coefficients
                        inStream >> rows;
                        inStream >> columns;

                        distanceCoefficients = Mat::zeros(rows, columns, CV_64F);

                        for(int r = 0; r < rows; r++)
                        {
                                for(int c = 0; c < columns; c++)
                                {
                                        double read = 0.0f;
                                        inStream >> read;
                                        distanceCoefficients.at<double>(r,c) =read;
                                        cout << distanceCoefficients.at<double>(r,c) << "\n";
                                }
                        }
                        inStream.close();
                        return true;

                }

                return false;
}


int startWebcamMonitoring(const Mat& cameraMatrix, const Mat& distanceCoefficients, float arucoSquareDimension)
{
                Mat frame;

                vector<int> markerIds;
                vector<vector<Point2f> > markerCorners, rejectedCandidates;
                aruco::DetectorParameters parameters;
```

```
	Ptr < aruco::Dictionary> markerDictionary = aruco::getPredefinedDictionary(aruco::DICT_4X4_50);
//aruco::PREDEFINED_DICTIONARY_NAME::DICT_4X4_50

	VideoCapture vid(0);//1

	if(!vid.isOpened())
	{
		return -1;
	}

	namedWindow("Webcam",CV_WINDOW_AUTOSIZE);

	vector<Vec3d> rotationVectors, translationVectors;

	while(true)
	{
		if(!vid.read(frame))
			break;

		aruco::detectMarkers(frame, markerDictionary, markerCorners, markerIds);
		aruco::estimatePoseSingleMarkers(markerCorners, arucoSquareDimension, cameraMatrix,
distanceCoefficients, rotationVectors, translationVectors);

		for(int i = 0; i < markerIds.size(); i++)
		{
			aruco::drawAxis(frame, cameraMatrix, distanceCoefficients, rotationVectors[i], translationVectors[i],
0.1f);
			cout<<translationVectors[i]<<endl;
		}

		imshow("Webcam", frame);
		if(waitKey(30) >= 0) break;
	}

	return 1;
}


void cameraCalibrationProcess(Mat& cameraMatrix, Mat& distanceCoefficients)
{
	Mat frame;
	Mat drawToFrame;

	vector<Mat> savedImages;
	vector<vector<Point2f> > markerCorners, rejectedCandidates;

	VideoCapture vid(0);

	if(!vid.isOpened())
	{
		return;
	}

	int framesPerSecond = 20;//20

	namedWindow("webcam", CV_WINDOW_AUTOSIZE);

	while(true)
	{
		if(!vid.read(frame))
			break;

		vector<Vec2f> foundPoints;
		bool found = false;

		found = findChessboardCorners(frame, chessboardDimensions, foundPoints,
CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_NORMALIZE_IMAGE);
		frame.copyTo(drawToFrame);
		drawChessboardCorners(drawToFrame, chessboardDimensions, foundPoints, found);
		if(found)
```

```
                imshow("Webcam", drawToFrame);
            else
                imshow("Webcam", frame);

            char character = waitKey(1000 / framesPerSecond);


            switch(character)
            {
                case 32: //SpaceBar key
                //saving image
                if(found)
                {
                    Mat temp;
                    frame.copyTo(temp);
                    savedImages.push_back(temp);
                }
                break;

                case 9: //Tab key
                //start calibration
                if(savedImages.size() > 3)
                {
                    cameraCalibration(savedImages, chessboardDimensions, calibrationSquareDimensions,
cameraMatrix, distanceCoefficients);
                    saveCameraCalibration("CamCal", cameraMatrix, distanceCoefficients);
                }
                break;

                case 27: //Esc key
                //exit
                return;
                break;
            }

        }
}


int main(int argv, char** argc)
{
        //createArucoMarkers();
        Mat cameraMatrix = Mat::eye(3, 3, CV_64F);
        Mat distanceCoefficients;

        //cameraCalibrationProcess(cameraMatrix, distanceCoefficients);
        loadCameraCalibration("CamCal",cameraMatrix,distanceCoefficients);
        startWebcamMonitoring(cameraMatrix, distanceCoefficients, 0.099f);



        return 0;

}
```

calibration.cpp

```
#include "opencv2/core.hpp"
#include <opencv2/core/utility.hpp>
#include "opencv2/imgproc.hpp"
#include "opencv2/calib3d.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/highgui.hpp"
#include <cctype>
```

```
#include <stdio.h>
#include <string.h>
#include <time.h>
using namespace cv;
using namespace std;


enum { DETECTION = 0, CAPTURING = 1, CALIBRATED = 2 };
enum Pattern { CHESSBOARD, CIRCLES_GRID, ASYMMETRIC_CIRCLES_GRID };

static double computeReprojectionErrors(
        const vector<vector<Point3f> >& objectPoints,
        const vector<vector<Point2f> >& imagePoints,
        const vector<Mat>& rvecs, const vector<Mat>& tvecs,
        const Mat& cameraMatrix, const Mat& distCoeffs,
        vector<float>& perViewErrors )
{
    vector<Point2f> imagePoints2;
    int i, totalPoints = 0;
    double totalErr = 0, err;
    perViewErrors.resize(objectPoints.size());

    for( i = 0; i < (int)objectPoints.size(); i++ )
    {
        projectPoints(Mat(objectPoints[i]), rvecs[i], tvecs[i],
                      cameraMatrix, distCoeffs, imagePoints2);
        err = norm(Mat(imagePoints[i]), Mat(imagePoints2), NORM_L2);
        int n = (int)objectPoints[i].size();
        perViewErrors[i] = (float)std::sqrt(err*err/n);
        totalErr += err*err;
        totalPoints += n;
    }

    return std::sqrt(totalErr/totalPoints);
}

static void calcChessboardCorners(Size boardSize, float squareSize, vector<Point3f>& corners, Pattern patternType =
CHESSBOARD)
{
    corners.resize(0);

    switch(patternType)
    {
      case CHESSBOARD:
      case CIRCLES_GRID:
        for( int i = 0; i < boardSize.height; i++ )
            for( int j = 0; j < boardSize.width; j++ )
                corners.push_back(Point3f(float(j*squareSize),
                                          float(i*squareSize), 0));
        break;

      case ASYMMETRIC_CIRCLES_GRID:
        for( int i = 0; i < boardSize.height; i++ )
            for( int j = 0; j < boardSize.width; j++ )
                corners.push_back(Point3f(float((2*j + i % 2)*squareSize),
                                          float(i*squareSize), 0));
        break;

      default:
        CV_Error(Error::StsBadArg, "Unknown pattern type\n");
    }
}

static bool runCalibration( vector<vector<Point2f> > imagePoints,
                    Size imageSize, Size boardSize, Pattern patternType,
                    float squareSize, float aspectRatio,
                    int flags, Mat& cameraMatrix, Mat& distCoeffs,
                    vector<Mat>& rvecs, vector<Mat>& tvecs,
                    vector<float>& reprojErrs,
                    double& totalAvgErr)
```

```
{
    cameraMatrix = Mat::eye(3, 3, CV_64F);
    if( flags & CALIB_FIX_ASPECT_RATIO )
        cameraMatrix.at<double>(0,0) = aspectRatio;

    distCoeffs = Mat::zeros(8, 1, CV_64F);

    vector<vector<Point3f> > objectPoints(1);
    calcChessboardCorners(boardSize, squareSize, objectPoints[0], patternType);

    objectPoints.resize(imagePoints.size(),objectPoints[0]);

    double rms = calibrateCamera(objectPoints, imagePoints, imageSize, cameraMatrix,
                    distCoeffs, rvecs, tvecs, flags|CALIB_FIX_K4|CALIB_FIX_K5);
                    ///*|CALIB_FIX_K3*/|CALIB_FIX_K4|CALIB_FIX_K5);
    printf("RMS error reported by calibrateCamera: %g\n", rms);

    bool ok = checkRange(cameraMatrix) && checkRange(distCoeffs);

    totalAvgErr = computeReprojectionErrors(objectPoints, imagePoints,
                rvecs, tvecs, cameraMatrix, distCoeffs, reprojErrs);

    return ok;
}


static void saveCameraParams( const string& filename,
                        Size imageSize, Size boardSize,
                        float squareSize, float aspectRatio, int flags,
                        const Mat& cameraMatrix, const Mat& distCoeffs,
                        const vector<Mat>& rvecs, const vector<Mat>& tvecs,
                        const vector<float>& reprojErrs,
                        const vector<vector<Point2f> >& imagePoints,
                        double totalAvgErr )
{
    FileStorage fs( filename, FileStorage::WRITE );

    time_t tt;
    time( &tt );
    struct tm *t2 = localtime( &tt );
    char buf[1024];
    strftime( buf, sizeof(buf)-1, "%c", t2 );

    fs << "calibration_time" << buf;

    if( !rvecs.empty() || !reprojErrs.empty() )
        fs << "nframes" << (int)std::max(rvecs.size(), reprojErrs.size());
    fs << "image_width" << imageSize.width;
    fs << "image_height" << imageSize.height;
    fs << "board_width" << boardSize.width;
    fs << "board_height" << boardSize.height;
    fs << "square_size" << squareSize;

    if( flags & CALIB_FIX_ASPECT_RATIO )
        fs << "aspectRatio" << aspectRatio;

    if( flags != 0 )
    {
        sprintf( buf, "flags: %s%s%s%s",
            flags & CALIB_USE_INTRINSIC_GUESS ? "+use_intrinsic_guess" : "",
            flags & CALIB_FIX_ASPECT_RATIO ? "+fix_aspectRatio" : "",
            flags & CALIB_FIX_PRINCIPAL_POINT ? "+fix_principal_point" : "",
            flags & CALIB_ZERO_TANGENT_DIST ? "+zero_tangent_dist" : "" );
        //cvWriteComment( *fs, buf, 0 );
    }

    fs << "flags" << flags;

    fs << "camera_matrix" << cameraMatrix;
    fs << "distortion_coefficients" << distCoeffs;
```

```
    fs << "avg_reprojection_error" << totalAvgErr;
    if( !reprojErrs.empty() )
        fs << "per_view_reprojection_errors" << Mat(reprojErrs);

    if( !rvecs.empty() && !tvecs.empty() )
    {
        CV_Assert(rvecs[0].type() == tvecs[0].type());
        Mat bigmat((int)rvecs.size(), 6, rvecs[0].type());
        for( int i = 0; i < (int)rvecs.size(); i++ )
        {
            Mat r = bigmat(Range(i, i+1), Range(0,3));
            Mat t = bigmat(Range(i, i+1), Range(3,6));

            CV_Assert(rvecs[i].rows == 3 && rvecs[i].cols == 1);
            CV_Assert(tvecs[i].rows == 3 && tvecs[i].cols == 1);
            //*.t() is MatExpr (not Mat) so we can use assignment operator
            r = rvecs[i].t();
            t = tvecs[i].t();
        }
        //cvWriteComment( *fs, "a set of 6-tuples (rotation vector + translation vector) for each view", 0 );
        fs << "extrinsic_parameters" << bigmat;
    }

    if( !imagePoints.empty() )
    {
        Mat imagePtMat((int)imagePoints.size(), (int)imagePoints[0].size(), CV_32FC2);
        for( int i = 0; i < (int)imagePoints.size(); i++ )
        {
            Mat r = imagePtMat.row(i).reshape(2, imagePtMat.cols);
            Mat imgpti(imagePoints[i]);
            imgpti.copyTo(r);
        }
        fs << "image_points" << imagePtMat;
    }
}

static bool readStringList( const string& filename, vector<string>& l )
{
    l.resize(0);
    FileStorage fs(filename, FileStorage::READ);
    if( !fs.isOpened() )
        return false;
    FileNode n = fs.getFirstTopLevelNode();
    if( n.type() != FileNode::SEQ )
        return false;
    FileNodeIterator it = n.begin(), it_end = n.end();
    for( ; it != it_end; ++it )
        l.push_back((string)*it);
    return true;
}


static bool runAndSave(const string& outputFilename,
                const vector<vector<Point2f> >& imagePoints,
                Size imageSize, Size boardSize, Pattern patternType, float squareSize,
                float aspectRatio, int flags, Mat& cameraMatrix,
                Mat& distCoeffs, bool writeExtrinsics, bool writePoints )
{
    vector<Mat> rvecs, tvecs;
    vector<float> reprojErrs;
    double totalAvgErr = 0;

    bool ok = runCalibration(imagePoints, imageSize, boardSize, patternType, squareSize,
                  aspectRatio, flags, cameraMatrix, distCoeffs,
                  rvecs, tvecs, reprojErrs, totalAvgErr);
    printf("%s. avg reprojection error = %.2f\n",
           ok ? "Calibration succeeded" : "Calibration failed",
           totalAvgErr);
```

```
    if( ok )
        saveCameraParams( outputFilename, imageSize,
                          boardSize, squareSize, aspectRatio,
                          flags, cameraMatrix, distCoeffs,
                          writeExtrinsics ? rvecs : vector<Mat>(),
                          writeExtrinsics ? tvecs : vector<Mat>(),
                          writeExtrinsics ? reprojErrs : vector<float>(),
                          writePoints ? imagePoints : vector<vector<Point2f> >(),
                          totalAvgErr );
    return ok;
}


int main( int argc, char** argv )
{
    Size boardSize, imageSize;
    float squareSize, aspectRatio;
    Mat cameraMatrix, distCoeffs;
    string outputFilename;
    string inputFilename = "";

    int i, nframes;
    bool writeExtrinsics, writePoints;
    bool undistortImage = false;
    int flags = 0;
    VideoCapture capture;
    bool flipVertical;
    bool showUndistorted;
    bool videofile;
    int delay;
    clock_t prevTimestamp = 0;
    int mode = DETECTION;
    int cameraId = 0;
    vector<vector<Point2f> > imagePoints;
    vector<string> imageList;
    Pattern pattern = CHESSBOARD;

    cv::CommandLineParser parser(argc, argv,
        "{help ||}{w||}{h||}{pt|chessboard|}{n|10|}{d|1000|}{s|1|}{o|out_camera_data.yml|}"
        "{op||}{oe||}{zt||}{a|1|}{p||}{v||}{V||}{su||}"
        "{@input_data|0|}");
    if (parser.has("help"))
    {
        help();
        return 0;
    }
    boardSize.width = parser.get<int>( "w" );
    boardSize.height = parser.get<int>( "h" );
    if ( parser.has("pt") )
    {
        string val = parser.get<string>("pt");
        if( val == "circles" )
            pattern = CIRCLES_GRID;
        else if( val == "acircles" )
            pattern = ASYMMETRIC_CIRCLES_GRID;
        else if( val == "chessboard" )
            pattern = CHESSBOARD;
        else
            return fprintf( stderr, "Invalid pattern type: must be chessboard or circles\n" ), -1;
    }
    squareSize = parser.get<float>("s");
    nframes = parser.get<int>("n");
    aspectRatio = parser.get<float>("a");
    delay = parser.get<int>("d");
    writePoints = parser.has("op");
    writeExtrinsics = parser.has("oe");
    if (parser.has("a"))
        flags |= CALIB_FIX_ASPECT_RATIO;
    if ( parser.has("zt") )
        flags |= CALIB_ZERO_TANGENT_DIST;
```

```
if ( parser.has("p") )
    flags |= CALIB_FIX_PRINCIPAL_POINT;
flipVertical = parser.has("v");
videofile = parser.has("V");
if ( parser.has("o") )
    outputFilename = parser.get<string>("o");
showUndistorted = parser.has("su");
if ( isdigit(parser.get<string>("@input_data")[0]) )
    cameraId = parser.get<int>("@input_data");
else
    inputFilename = parser.get<string>("@input_data");
if (!parser.check())
{
    help();
    parser.printErrors();
    return -1;
}
if ( squareSize <= 0 )
    return fprintf( stderr, "Invalid board square width\n" ), -1;
if ( nframes <= 3 )
    return printf("Invalid number of images\n" ), -1;
if ( aspectRatio <= 0 )
    return printf( "Invalid aspect ratio\n" ), -1;
if ( delay <= 0 )
    return printf( "Invalid delay\n" ), -1;
if ( boardSize.width <= 0 )
    return fprintf( stderr, "Invalid board width\n" ), -1;
if ( boardSize.height <= 0 )
    return fprintf( stderr, "Invalid board height\n" ), -1;

if( !inputFilename.empty() )
{
    if( !videofile && readStringList(inputFilename, imageList) )
        mode = CAPTURING;
    else
        capture.open(inputFilename);
}
else
    capture.open(cameraId);

if( !capture.isOpened() && imageList.empty() )
    return fprintf( stderr, "Could not initialize video (%d) capture\n",cameraId ), -2;

if( !imageList.empty() )
    nframes = (int)imageList.size();

if( capture.isOpened() )
    printf( "%s", liveCaptureHelp );

namedWindow( "Image View", 1 );

for(i = 0;;i++)
{
    Mat view, viewGray;
    bool blink = false;

    if( capture.isOpened() )
    {
        Mat view0;
        capture >> view0;
        view0.copyTo(view);
    }
    else if( i < (int)imageList.size() )
        view = imread(imageList[i], 1);

    if(view.empty())
    {
        if( imagePoints.size() > 0 )
            runAndSave(outputFilename, imagePoints, imageSize,
                    boardSize, pattern, squareSize, aspectRatio,
```

```
                flags, cameraMatrix, distCoeffs,
                writeExtrinsics, writePoints);
    break;
}

imageSize = view.size();

if( flipVertical )
    flip( view, view, 0 );

vector<Point2f> pointbuf;
cvtColor(view, viewGray, COLOR_BGR2GRAY);

bool found;
switch( pattern )
{
    case CHESSBOARD:
        found = findChessboardCorners( view, boardSize, pointbuf,
            CALIB_CB_ADAPTIVE_THRESH | CALIB_CB_FAST_CHECK | CALIB_CB_NORMALIZE_IMAGE);
        break;
    case CIRCLES_GRID:
        found = findCirclesGrid( view, boardSize, pointbuf );
        break;
    case ASYMMETRIC_CIRCLES_GRID:
        found = findCirclesGrid( view, boardSize, pointbuf, CALIB_CB_ASYMMETRIC_GRID );
        break;
    default:
        return fprintf( stderr, "Unknown pattern type\n" ), -1;
}

// improve the found corners' coordinate accuracy
if( pattern == CHESSBOARD && found) cornerSubPix( viewGray, pointbuf, Size(11,11),
    Size(-1,-1), TermCriteria( TermCriteria::EPS+TermCriteria::COUNT, 30, 0.1 ));

if( mode == CAPTURING && found &&
   (!capture.isOpened() || clock() - prevTimestamp > delay*1e-3*CLOCKS_PER_SEC) )
{
    imagePoints.push_back(pointbuf);
    prevTimestamp = clock();
    blink = capture.isOpened();
}

if(found)
    drawChessboardCorners( view, boardSize, Mat(pointbuf), found );

string msg = mode == CAPTURING ? "100/100" :
    mode == CALIBRATED ? "Calibrated" : "Press 'g' to start";
int baseLine = 0;
Size textSize = getTextSize(msg, 1, 1, 1, &baseLine);
Point textOrigin(view.cols - 2*textSize.width - 10, view.rows - 2*baseLine - 10);

if( mode == CAPTURING )
{
    if(undistortImage)
        msg = format( "%d/%d Undist", (int)imagePoints.size(), nframes );
    else
        msg = format( "%d/%d", (int)imagePoints.size(), nframes );
}

putText( view, msg, textOrigin, 1, 1,
         mode != CALIBRATED ? Scalar(0,0,255) : Scalar(0,255,0));

if( blink )
    bitwise_not(view, view);

if( mode == CALIBRATED && undistortImage )
{
    Mat temp = view.clone();
    undistort(temp, view, cameraMatrix, distCoeffs);
}
```

```
        imshow("Image View", view);
        char key = (char)waitKey(capture.isOpened() ? 50 : 500);

        if( key == 27 )
            break;

        if( key == 'u' && mode == CALIBRATED )
            undistortImage = !undistortImage;

        if( capture.isOpened() && key == 'g' )
        {
            mode = CAPTURING;
            imagePoints.clear();
        }

        if( mode == CAPTURING && imagePoints.size() >= (unsigned)nframes )
        {
            if( runAndSave(outputFilename, imagePoints, imageSize,
                    boardSize, pattern, squareSize, aspectRatio,
                    flags, cameraMatrix, distCoeffs,
                    writeExtrinsics, writePoints))
                mode = CALIBRATED;
            else
                mode = DETECTION;
            if( !capture.isOpened() )
                break;
        }
    }

    if( !capture.isOpened() && showUndistorted )
    {
        Mat view, rview, map1, map2;
        initUndistortRectifyMap(cameraMatrix, distCoeffs, Mat(),
                        getOptimalNewCameraMatrix(cameraMatrix, distCoeffs, imageSize, 1, imageSize, 0),
                        imageSize, CV_16SC2, map1, map2);

        for( i = 0; i < (int)imageList.size(); i++ )
        {
            view = imread(imageList[i], 1);
            if(view.empty())
                continue;
            //undistort( view, rview, cameraMatrix, distCoeffs, cameraMatrix );
            remap(view, rview, map1, map2, INTER_LINEAR);
            imshow("Image View", rview);
            char c = (char)waitKey();
            if( c == 27 || c == 'q' || c == 'Q' )
                break;
        }
    }

    return 0;
}
```

Appendix E: AUBO Script Communication Proof of Concept

TestFile1.aubo

```
--Initializes the movement of the robot and moves it to the zero position
init_global_move_profile()
move_joint({0.0,0.0,0.0,0.0,0.0,0.0},true)

--moves the robot to the initial position and back to zero 3 times
i = 0
while(i<3)do
	move_joint({-0.000003, -0.127267, -1.321122, 0.376934, -1.570796, -0.000008},true)
	move_joint({0.0,0.0,0.0,0.0,0.0,0.0},true)
	i = i + 1
End
```

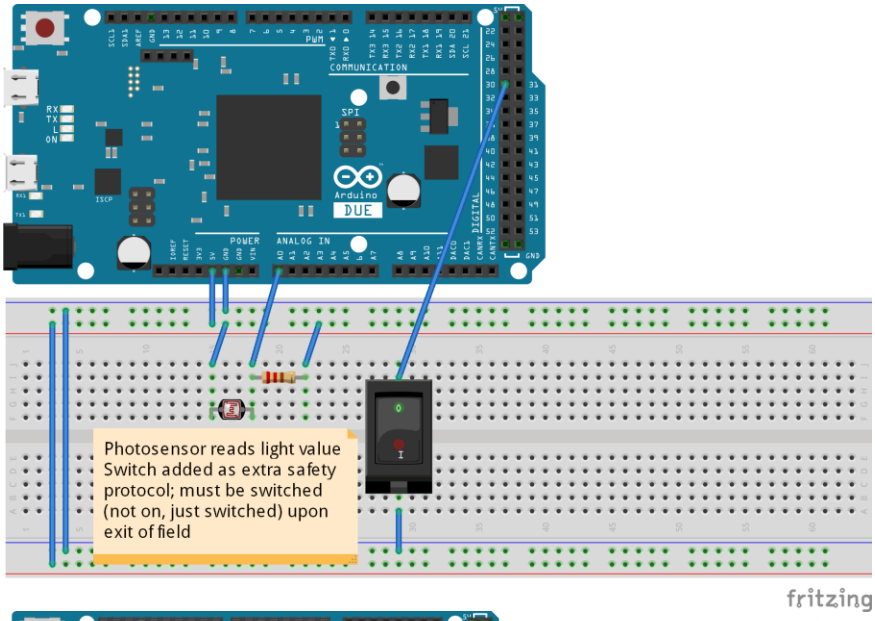
Appendix F: Arduino Code for integrated Grip and Trip Wire

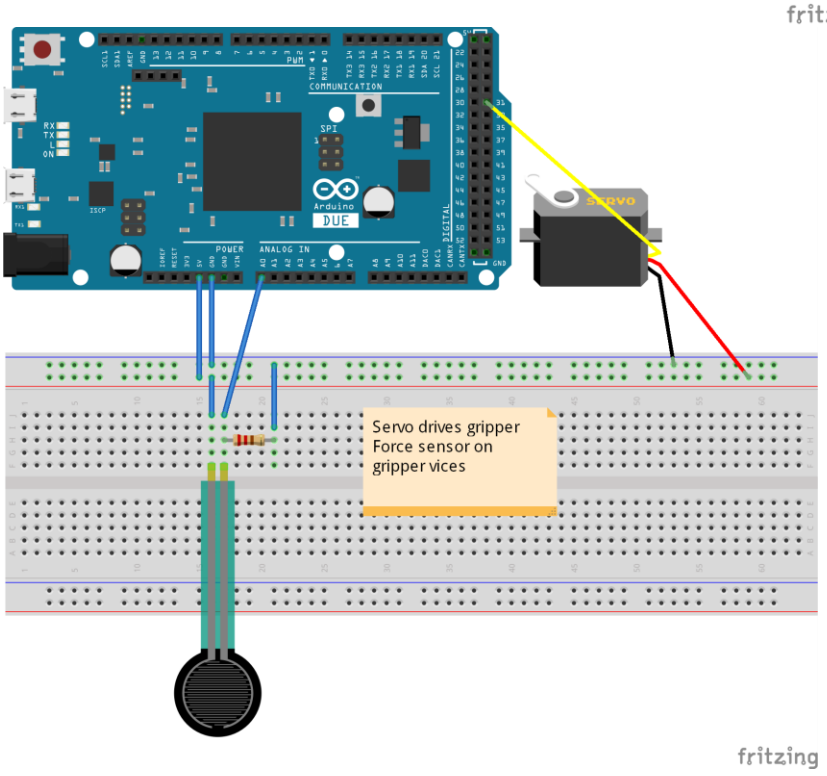
bterry7_ME450_AUBOStrong_MainAll.ino

```
boolean person = false;
boolean switched = false;
boolean switched_read;
uint16_t voltage; //measure voltage across resistor
uint8_t sensorPin = A5;
uint8_t switchPin = 33;

uint8_t ledPin1 = 53;
uint8_t ledPin2 = 31;

#include <Servo.h>
Servo gripper;

uint8_t pos = 80; //80 is full open, 0 is full closed
uint16_t voltage_f;
uint8_t sensorPin_f = A7; //change these to match current configuration
uint8_t servoPin = 29;
/*Turn off/on Led to test certain parts of code
uint8_t ledPin1 = 0;
uint8_t ledPin2 = 0;
*/

//Inputs/Outputs for starting grip and saying it is gripped
boolean gripped = 0;
boolean grip_start = 1;


void setup() {
  Serial.begin(9600);
  pinMode(switchPin,INPUT);
  digitalWrite(switchPin,HIGH); //pullup resistors, means it needs to be connected to ground
  pinMode(ledPin1,OUTPUT);
  pinMode(ledPin2,OUTPUT);
  digitalWrite(ledPin1,LOW);
  digitalWrite(ledPin2,LOW);

   gripper.attach(servoPin);

}

void loop() {
  voltage = analogRead(sensorPin);
  switched_read = digitalRead(switchPin);

  Serial.println(switched_read);

  if (voltage < 1000) {
    person = true;
    switched = switched_read; //precludes the possibility of flipping the switch and then reintroducing light, which we also can (and
should) physically constrain; may be annoying if you try to flip it and walk, but I think that only really applies to testing/holding the
laser by hand
 } else {
    if(person && switched_read == switched) {
      person = true;
    } else {
      person = false;
      switched = switched_read;
    }
  }
  if (!person) {
    digitalWrite(ledPin1,HIGH);
    digitalWrite(ledPin2,LOW);
```

```
  } else {
    digitalWrite(ledPin1,LOW);
    digitalWrite(ledPin2,HIGH);
  }

  if(!person) {
  gripper.write(pos); //start at full open or hold value
  if(grip_start){
    voltage_f = analogRead(sensorPin_f);
    if(voltage_f < 5 && pos != 0 ) { //Threshold value, can be changed as needed
      pos -= 1;
      gripper.write(pos);

    } else {
      if (gripped = 0) {
        pos -= 2; // very low threshold, so once past we can back off a bit and not make the servo keep trying to close, kind of pointless
right now because then voltage will drop and then just resqueeze
        gripper.write(pos);
        gripped = 1;
      }
      gripper.write(pos);
    }
    delay(500); //for testing purposes, to be removed for actual use
    Serial.println(voltage_f);
  }
  }

}
```

Appendix G: Circuit Diagrams for Grip System and Trip Laser Safety System





Made using Fritzing