AUTOMATED SORTING OF PEGS USING COMPUTER VISION

by

Taylor W Hubbard

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

School of Engineering Technology

West Lafayette, Indiana

December 2018

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Suranjan Panigrahi, Co-Chair

School of Electrical Engineering Technology

Mr. James Jacob, Co-Chair

School of Electrical Engineering Technology

Mr. Bradley Harriger, Committee Member

School of Mechanical Engineering Technology

Approved by:

Dr. Duane D. Dunlap

Chair of SOET Graduate Program

ACKNOWLEDGMENTS

I would like to express my gratitude towards Dr. Suranjan Panigrahi, my advisor, for his support and guidance throughout my thesis. I would also like to thank my committee members Mr. James Jacob and Mr. Bradley Harriger for their support and technical insight and guidance.

I would like to thank both Terry Echard and Lafayette Instruments for their monetary and technical support throughout the project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

REQUIRED SECTION

BOM	Bill of Materials
CAD	Computer Aided Design
ROI	Region of Interest
CCD	Charge Coupled Device
CMOS	Complementary Metal Oxide Semi-Conductor
GPIO	General Purpose Input/Output
HMI	Human Machine Interface
DC	Direct Current
PWM	Pulse Width Modulation

GLOSSARY

Computer-aided design or CAD – Computer technology used in design of 2D or 3D diagrams that can be viewed from any angle, often facilitating the manufacturing process ("What is Computer-Aided Design (CAD)? ").

Computer vision – Science or technology using methods of acquiring, processing, analyzing, and understanding images from the real world, producing a numerical or symbolic output ("Translations for computer vision").

Extrusion – For metal, a piece is forced through a die of smaller cross-sectional area producing a metal piece with a new cross-sectional profile (“Metal Extrusion”).

SolidWorks – 3D CAD software created by Dassault Systemes delivering simulation and design validation. Currently using Solidworks 2017 Student Edition ("Packages").

ABSTRACT

Author: Hubbard, Taylor, W. MS
Institution: Purdue University
Degree Received: December 2018
Title: Automated Sorting of Pegs using Computer Vision
Committee Chair: Suranjan Panigrahi

The thesis covered the creation and testing of a low cost and modular sorting system of pegs used in products by Lafayette Instruments. The system is designed to check peg dimensions through use of computer vision while sorting out nonconforming parts and counting ones that are conforming. Conforming parts are separated into bins of predetermined quantities so that they do not need manual counting. The developed system will save engineers and technicians at Lafayette instruments many man hours from manually sorting and counting the roughly 160,000 pegs a year. The system will be able to sort and count at a speed comparable to a human operator while achieving an overall average accuracy of 95% or higher.

CHAPTER 1. INTRODUCTION

Lafayette Instruments is currently counting and sorting small diameter metal pegs by hand and visually inspecting length to ensure they have met specifications. The process of automating this task of sorting and counting is described in the following thesis.

1.1 Scope

Engineers have developed skills that allow them to excel in their field of design, CAD, manufacturing, or other mentally challenging tasks. Engineers progress can be hindered by the requirement to do tedious and repetitive tasks. The automation of laborious and monotonous human work is becoming more common in today's workplace so that human time can be spent elsewhere. Duties that are highly repetitive and time consuming for a worker can now be carried out by a machine designed for that specific task. Not only does automating processes decrease the unnecessary labor hours, it also increases the reliability and repeatability of the product being manufactured, assembled or sorted.

At Lafayette Instruments, pegs are produced by a cutting and extrusion manufacturing process. The pegs are counted and bagged to be shipped with pegboards. Smaller pegs, or stubs, create issues with counting the quantity and ensuring that the specified number of pegs with correct lengths are shipped to the customer. The counting process is currently a weight based measurement. According to Lafayette Instruments, a total of roughly 160 thousand pegs are purchased each year. Pegs are currently visually inspected. Visual inspection can lead to overlooked stubs left in with the conforming pegs. Moreover, the stubs may be counted as a peg in the counting process leaving the customer short of the pegs necessary to fully assemble a pegboard.

Computer vision is a method that can be used in automating the human visual inspection process. In many cases such as the sorting of grommets, the computer vision process requires a large machine that is not easily moved and requires expensive parts and software. The software alone in this instance costs over 130 thousand dollars (Regazzoni, 2017). The creation of an inexpensive and portable option is needed for smaller environments that still require the increased accuracy and the reduced man hours of accomplishing this tedious and monotonous task.

1.2 Significance

Many computer vision-based sorting systems have been developed for industrial environments. These systems are able to sort parts at greater speed and with higher accuracy than that of a human visual inspector. No system has been developed that is both portable and inexpensive for small metal parts in smaller industries. The system developed in this study will help a local company increase reliability in the sorting and counting of metal pegs while reducing the time engineers and technicians must spend on the task.

1.3 Research Question

The research will be conducted to determine how to develop an inexpensive and portable integrated automatic peg sorting system using computer vision techniques.

1.4 Assumptions

The following assumptions were inherent in this study:

1. Pegs of only three different dimensions are sorted and counted in this study.

2. Pegs of only one type are placed in the hopper at a time and are not mixed with any of the other two types of pegs.

3. Pegs have all correct dimensions with the exception of peg length before introduction into the system.

1.5 Limitations

The following limitations were inherent in this study:

1. The study is limited to the sorting of the three peg dimensions given by Lafayette Instruments without appropriate modifications to the hopper and software parameters.

2. Statistical analysis is limited by the number of pegs given by Lafayette Instruments.

3. The study accuracy is limited by the accuracy and tolerance of some 3D printed and laser cut custom fabrications.

4. Although modular, only one of the systems will be built for the sorting of pegs. Given time and resources, several could be made to handle each type of peg individually.

1.6 Delimitations

The following delimitations were inherent in this study:

1. The study will not be tested on pins or pegs outside of the three specified by Lafayette Instruments.

2. The study does not use neural networks or machine learning to accomplish the task. These techniques along with mild adjustments may be used to increase the number of objects able to be sorted by the system.

1.7 Organization

This thesis contains five major chapters and multiple appendices. Chapter 2 provides an overview of current sorting system used in industry and in research. It begins with current system designs focusing on the electrical and mechanical hardware used. It then discusses computer vision algorithms and techniques and concludes with a summary of the statistical analysis used in the studies reviewed.

Chapter 3 presents an overview to the methodology and framework within the study. The chapter provides a preliminary and revised design of the sorting system in detail along with computer-aided design or CAD representations.

Chapter 4 discusses the results of the study. These results include post build revisions, testing procedures that are applied for each type of peg and a detailed analysis of the performance of each peg type though the sorting system. The results are analyzed and evaluated against the performance criteria for the sorting system.

Chapter 5 provides a summary and conclusions of the overall study. The chapter also discusses the recommendations to further the design and accuracy of the system.

CHAPTER 2. REVIEW OF LITERATURE

The goal of developing a system to fully automate a human process is one that must incorporate both electrical and mechanical devices. Automation also requires software to control the system. The literature review presented is focused on current designs used for automation of the sorting processes. Sorting may require dimensioning of the object, color detection or searching for other visual characteristics. The following section breaks the research down into the electrical and mechanical hardware, as well as, the software currently in use in developed autonomous sorting and vision systems.

2.1 Current System Designs

The design of developed machine vision systems consists of an array of mechanical hardware, electrical hardware and software. Working together, the hardware and software are able to sort objects based on color, area, curvature or circularity as well as object deformation. Machine vision systems are abundant in all industrial applications.

2.1.1 Mechanical and Electrical Hardware

This section consists of mechanical and electrical hardware directly used in the sorting process. Hardware used in common sorting machines includes motors, actuators, conveyors, hoppers and other electromechanical devices.

There are a multitude of sorting machines and systems that can result in a product being properly sorted or categorized. Each system consists of an array of different hardware that must work in coalition to achieve the sorting process. In many sorting operations a crucial precursor to the vision system is the individual feed of objects into the system. Carrots for example, can be characterized by their color and weight. Through these two features, carrots can be assigned a grade and be sorted into graded bins prior to being delivered to the customer. In a study presented by Hong Wei Du, the task of individually feeding the carrots into a vision system used a hopper in conjunction with a grid staggered belt. The spacing of the grid staggered belt ensured that only one carrot was carried from the hopper tray at a time (Du, 2015). Once classified into grades by the computer vision system, a number of linear actuators were used to remove the carrots from the conveyor and push them down chutes into their designated bins (Du, 2015). Systems have also been developed to sort walnuts based on grade and deformations. The walnut sorting system also used a hopper in order to individually feed into the sorting system. Once characterized, servos attached to pushing plates were used to push the walnuts from the conveyor and into designated bins based on grade (Tran, Nguyen, Pham, 2016). From research of sorting machines, a hopper appears to be the common segregation mechanism for the objects of interest. In a study on classification of ceramic substrates, pneumatics were used for the individual sorting of the substrates and suction was used to move the substrate from the stack and to a conveyor. Once classified, pneumatics were used to place the substrates into the designated class boxes. (Li-na & Li-jun, 2012)

Conveyors come in several different variations. In some cases, the conveyor consists of belts and pulleys while others use rollers which rotate the object along the sorting line. In the use of a vision system, rollers can be especially important as they allow pictures to be taken of all sides of the object with a single camera as the object is rolled past. This method is presented in a study on the sorting of apples conducted by Sofu et al. (2016). In order to check the entire surface of apples for scabs, stains and rotting, multiple pictures had to be taken of the apple as it passed. The rollers were used to rotate the object of interest as it traveled down the sorting line. Algorithms were then used to organize the images and analyze for the deformations (Sofu, Kayacan & Cetisli, 2016).

In a study performed by ElMasry et al. (2012), the roller conveyor method was again used when visually inspecting the exterior characteristics of potatoes for color, shape and defects. A second option for conveying the segmented object is a chain driven conveyor. In Hong Wei Du's carrot sorting machine, the main conveyor was driven by a motor, chain and reducer gear. A secondary motor drive chain was used to drive the staggered belt for the hopper (Du, 2015). Conveying can also be accomplished by the use of a motor and a belt. In the sorting of Tamarind, a pod shaped fruit, a white conveyor was used to create contrast between the brown pod of the fruit and its background. This was acceptable as the objects are often curved and would remain stationary without the need for ridges or a staggered grid belt (Jarimopas & Jaisin, 2008).

After the object is classified by a detector or computer vision system, the objects must be physically removed from the conveyor. As discussed earlier, this can be accomplished by the use of pneumatics and electromechanical hardware. Li-na and Li-Jun (2008) developed a system that used pneumatics in the sorting of ceramic substrates. A nozzle was used to create suction to the ceramic substrate so that it could be lifted and transported. In other cases such as in the sorting of walnuts, carrots or sweet tamarind, servos and actuators are used to push the object from the conveyor and into classified containers. In sorting sweet tamarind, the tamarind was pushed into containers using a pneumatic actuator with a plate attached. The plate ensured that all of the tamarind was slid across the conveyor (Jarimopas & Jaisin, 2008).

2.1.2 Vision Inspection System

The food and automotive industry are examples of where the visual inspection of a product is a common verification technique or sorting technique incorporated into the quality control process. The visual inspection of the product can be automated by use of a computer vision system. Several important factors of the acquisition of a defined image include the background, lighting, camera type, camera resolution and camera actuation.

The background of an image plays an important role in gaining defined edges for an object. It is important for the background to create a high contrast to the object in front of it. Background contrast was expressed in a vision system by Probuwono et al. (2006), which checked the tops of bottles and determined if the cap was properly installed. The contrast of the image was maximized by the white light reflected from the bottle and the black background. From the image the edges were determined to evaluate if the cap was missing or installed improperly (Probuwono, Sulaiman, Hamdan & Hasniaty, 2006). In the identification and sorting of sweet tamarind, the tamarind was analyzed for curvature. The curvature of the tamarind was determined by analyzing the image. Image analysis was performed by an algorithm that looked at pixels, starting from outside the object and searching inward until the first black pixel was found. This was done in a radial pattern to defined the outside of the tamarind (Jarimopas & Jaisin, 2008).

Lighting is another crucial factor of the environment and the object. Lighting has the ability to increase the contrast between the object and background. Improper and non-uniform lighting can create shadows which decreases the abilities of algorithms such as edge detection. There are a number of ways lighting systems can be designed in a machine vision system. In a study for the sorting of potatoes, 16 40-watt daylight compact fluorescent tubes were mounted to the sides and tops of the system lighting chamber. The purpose of the high number of bulbs was to "provide homogenous lighting on the inspected tubers as they passed below the camera." (ElMasry, Subero, Molto & Blasco, 2012). Emphasis was placed on the lighting system in this study as directional light over the potatoes had the ability to create spots and specular reflections on the surface. Reflections on the surface can mask underlying deformities. Cross polarized filters were also used over the fluorescent tubes to reduce reflections (ElMasry, Subero, Molto & Blasco, 2012).

Other translucent material can be used to disperse the lighting in a process known as diffusion. In the automated measurement and dimensioning of basil seeds, two fluorescent lamps were placed behind a translucent white seed holder. The diffusion of the light around the seeds reduced shadows and increased the contrast between the seed and the background. This lighting option is known as back lighting (Razavi, Bostan & Rezaie, 2010).

Camera type is an important camera selection criterion as each type has their own advantages and disadvantages. Two types of cameras are typically used, charge-coupled devices (CCD) and complementary metal oxide semiconductor (CMOS). According to *Practical Image and Video Processing Using MATLAB* (Marques, 2011), CCD cameras have become the camera of choice for vision systems as they do not suffer from geometric distortions and have a linear response to reflected light. The geometric distortion found in other cameras is caused when the object is moving too fast for a frame to be fully and accurately captured. However, with high intensity lighting, CCD cameras can suffer from the effect known as blooming. Blooming occurs if the light intensity over-saturates one of the photosites. Photosites are the cells inside the CCD camera which convert light into voltage. The saturated photosite begins to overflow into the neighboring photosites (Marques, 2011). In a study classifying mechanical bearings, two separate CCD cameras were used for image acquisition. The images were then processed to find edges and overall contour (Pop, Grigorescu & Davidescu, 2017). A study used high intensity lighting and a CMOS camera to acquire an image of a non-moving ceramic substrate. The resolution of the camera was 3000x2208 equating to a calibration of 0.04mm/pixel. The pixel resolution was still too low and interpolation algorithms had to be run to satisfy the resolution accuracy of 0.008mm/pixel (Li-na & Li-jun, 2012). The important factor in measuring the substrate dimensions was the camera's resolution.

Camera actuation is the act where a device or piece of hardware determines when the camera will acquire an image. In ElMasry et al.'s study sorting potatoes, a Matrox Meteor frame grabber was actuated by an encoder connector to the conveyor. The encoder was designed to send out three pulses/mm. Knowing that an image was needed every 80mm, the frame grabber was told to take a picture every 240 pulses (ElMasry, Subero, Molto & Blasco, 2012). An encoder was also used in the study sorting apples based on their external appearance (Sofu, Kayacan & Cetisli, 2016).

2.1.3 Software Processing Algorithms

Software processing is the part of the process where the image is altered, enhanced and analyzed to extract specific data. A variety of algorithms are used in image processing. This section will go into detail on several currently in use systems and the different algorithms that are used to extract the desired information.

2.1.3.1 Preprocessing

The first step after image acquisition is preprocessing. Preprocessing enhances the overall image to better suit the algorithms that will later be performed. Preprocessing can solve many issues such as noise which may have a negative impact on later algorithms. According to *Practical Image and Video Processing Using MATLAB* (2011), preprocessing algorithms include erosion, dilation, segmentation and many others. Erosion is the shrinking of object in an image while dilation is the growing of object in the image. Sementation is the process of increase contrast between the background and the object (Marques, 2011). In a study inspecting metal parts moving on a conveyor, a preprocessing segmentation technique was used (Haider, Anton & Abdullah, 2011). The Otsu binarization technique evaluates the image and creates a threshold value between the object pixel values and the background pixel values. The technique then changes all pixel values above the threshold to 255 and all below the threshold to 0. On a binary scale this is similar to a 1 or 0. Other mathematical preprocessors were used to enhance the object including erosion, dilation and other 3x3 matrix operators (Haider, Anton & Abdullah, 2011). In the study sorting sweet tamarind, the image acquired was enhanced by changing the RGB intensities of the image. The image was broken into its RGB components and plotted on a histogram to show each colors overall intensity. The color spectrum of the image was then standardized by using a scaling factor derived from the histograms on each color. All red pixels were multiplied by 0.299, green pixels by 0.587 and blue pixels by 0.114. By adding up the RGB components of each pixel, a gray-scaled image was created. Further segmentation of the object was performed using Bayesian discriminant analysis (Jarimopas & Jaisin, 2008).

2.1.3.2 Edge Detection

The next process of most vision systems is the exacting of the boundary of the object being sorted. Boundary extraction is often performed using edge detection algorithms.

In a study analyzing and classifying small mechanical parts such as nuts, bolts, washers and screws, several edge detection algorithms were performed. Edge detection is a method that is used to extract the boundaries of an object. It is accomplished by finding the position at which the pixel values change abruptly. Edge detection was presented in a study by You and Zhang as being accomplished by two separate algorithms. The first is by the rank grads operator which uses the derivative to find the sudden change in pixel values. In calculus, this is presented as finding the partial derivative along both the x and y axes. In computer vision, these two partial derivatives are expressed by the equations shown in equation 2.1 (You & Zhang, 2008).

$$\Delta_y f = (\mathrm{i}, \mathrm{j}) - \mathrm{f}(\mathrm{i}, \mathrm{j} - 1)$$

$$\Delta_y f = (\mathrm{i}, \mathrm{j}) - \mathrm{f}(\mathrm{i}, \mathrm{j} - 1) \tag{2.1}$$

Methods must be combined in the analysis of a 2D image as the program begins searching in a grid fashion known as correlation or convolution. Three separate grad operators were described which go through each pixel of the image and look for differences in the neighboring pixels. These operators described were Roberts, Prewitt and Sobel (You & Zhang, 2008). The second method of edge detection is known as the Laplacian operator. The Laplacian operator starts by taking a grid of pixels, in this case a three by three square. The operator then multiplies the center pixel value by eight and subtracts the summation of the pixel values around it. The center pixel is then saved into a new image where edge locations are represented by higher pixel values. This operation is performed to all non-border pixels of the original image. Once the edges are defined, the program goes onto another algorithm focusing on contour tracking. Contour tracking is described as "to acquire the outer contour features of the image" or find the boundaries of the object (You & Zhang, 2008). The algorithm searches line by line in the image until finding the first black pixel. It then looks around that pixel for another black pixel and so on (You & Zhang, 2008).

The study by Haider et al., which inspected metal parts on a moving conveyor, the Otsu thresholding method and the 4-connected component labeling method were used as boundary definition search techniques. The 4-connected component labeling method found a pixel opposing to the background (in this case one with value 255) and then looked at its neighbors to see if they are the same value. It then moved to the next 255 value pixel and continued this process until the entire object of interest had a defined border (Haider, Anton & Abdullah, 2011).

The Canny method is another form of edge detection. In a study presented in the *Robotics and Computer-Integrated Manufacturing* journal, broken inserts of edge profiles in milling heads where identified. Canny's method was used along with a Hough transform to obtain the milling inserts borders. The Canny method finds edges by looking at the local maximum of the gradient. In this application, it computed the gradient after clearing image noise through use of a Gaussian filter. The Canny algorithm is preferred to Sobel, Prewitt or Roberts as its more complex workings were necessary in this study (Fernandez-Robles, Azzopardi, Alegre & Petkov, 2017). The Canny method was applied in conjunction with hysteresis thresholding. Hysteresis thresholding is creating low and high threshold values and then removing possible edges that do not reach the minimum threshold value. Hysteresis thresholding then keeps some of the weaker edges (those between the high and low threshold) that are connected to the strong edges above the higher threshold value (Fernandez-Robles, Azzopardi, Alegre & Petkov, 2017).

2.1.3.3 Classification

Boundaries of the objects are now defined and must be assigned numerical characteristics. The objects are then checked to see what criteria those numerical characteristics fall under. In the study performed by You and Zhang (2008), on the sorting of mechanical parts such as nuts, bolts and washers, a program calculated the area and circularity of each object. The area was calculated by counting the number of pixels inside the boundaries of the object. The circularity is calculated using the equations shown in Equation 2.2 and Equation 2.3 (You & Zhang, 2008).

$$\mathrm{C} = \frac{\mu_{\mathcal{R}}}{\delta_{\mathcal{R}}} \tag{2.2}$$

where $\mu_{\mathcal{R}}$ represents the average distance from the center of gravity to edge point. $\delta_{\mathcal{R}}$ is the mean square error of distance.

$$\mu_{\mathcal{R}} = \frac{1}{K}\sum_{k=0}^{K-1} ||\,(x_k, y_k) \; - \; (x, y)||$$

$$\delta_{\mathcal{R}} = \frac{1}{K}\sum_{k=0}^{K-1}\left[\left|\left|(x_k, y_k) - (x, y)\right|\right| - \delta_{\mathcal{R}}\right]^2 \tag{2.3}$$

Finally, after the algorithms classified each object with an area and a circularity value, the values were compared to a predefined table of ranges. For example, if the circularity fell 0.26-0.28 and the area 57000-59000 then the object was classified as a bolt and sorted accordingly. In the study performed, all objects (two washers, two screws, three nuts and a bolt) were able to be identified and classified correctly (You & Zhang, 2008).

Although this study was successful in a single test, the article did not discuss additional testing to prove the reliability of the algorithms used.

The classification techniques used by Haider et al. (2011) in sorting small metal parts on a conveyor were very similar to that of sorting the nuts and bolts. First, classification rules were created under which each component had a specific range for circularity and area. After calculation of both area and circularity of each object, the object properties were compared to the predefined rules. Objects were then able to be sorted based on the range they fit in. This process was described in the study as forward chaining. The overall accuracy of this system and use of algorithms was very positive at 99.25% for singular parts. The system drops in accuracy to 91.5% for a group of parts (Haider, Anton & Abdullah, 2011). Although the system was able to sort parts with acceptable reliability in singular settings, the system possessed no actuation method to sort the parts.

2.2 Alternative Methods

An alternative method presented by Hou (2001) disregarded using edge detection and classification to identify an object. Instead, Hou's study used a method called template matching. In a vision system developed for the inspection of IC leads, template matching consisted of comparing an image taken with an image previously stored in the database. The images were compared to each other by use of equation 2.4 to calculate a correlation coefficient which ranged from 0 to 1.

$$\mathrm{r}(\mathrm{s},\mathrm{t}) = \frac{\sum_x \sum_y [f(x,y) - \bar{f}(x,y)][w(x-s,y-t) - \bar{w}]}{\left(\sum_x \sum_y [f(x,y) - \bar{f}(x,y)]^2 [w(x-s,y-t) - \bar{w}]^2\right)^{\frac{1}{2}}} \qquad (2.4)$$

The correlation algorithm found the ends of each lead along the IC and through a Euclidean distance equation, calculating the pitch for the IC lead. Pitch of an IC is the distance between the center of one pin to another. This distance value tells if the IC leads are skewed or bent (Hou, 2001).

The object of interest does not need to be fully analyzed in specific applications. In the case of popcorn kernels presented by Pearson (2009) in the *Journal of Computers and Electronics in Agriculture,* the vision system is looking for blue-eye damage. Blue-eye damage is expressed as a dark spot on the outside of the kernel. When processing the image, a total of only 24 pixels aligned horizontally were analyzed. Plotted as a histogram with the pixel values, the blue-eye damage was represented by a dip in pixel values. Conditions were created such as "At least three pixels between pixels 10 and 14 of the 24 pixel set must have an intensity level less than a preset threshold level" to sort out those with blue-eye damage (Pearson, 2009). This condition captures the dip that is presented in the histogram. Analysis where only parts of the image are analyzed greatly speeds up processing times and does not require the many algorithms necessary to find the boundary and edges of the object.

Another alternative to the standard image processing techniques was presented in a system by Mario Regazzoni (2017). The system presented by Regazzoni focuses on high speed sorting of small parts using multiple cameras. The program used for image processing is Retina with a Squeezebrains AI library. In supervised machine learning or AI systems, the program is presented with a multitude of images of the object being analyzed. A human operator then goes through each picture and decides if that part is good or bad. The computer learns from the operator’s decision and formulates rules and criteria to classify the pictures taken of the actual parts. The more pictures classified by the operator, the more accurate the system can become (Regazzoni, 2017).

2.2.1 Statistical Analysis

Several statistical methods are often used to present the accuracy of a sorting or vision system. Haider, Anton & Abdullah (2011) used two different equations to represent sorting accuracy. Individual accuracy of any type of part was given by the summation of correct classification of images divided by the number of objects classified. Group accuracy was represented by the number of images of which all parts were identified correctly divided by the total number of classified images. Both individual and group accuracy methods produce a number between 0 and 1. In the method introduced by Hou (2001) using template matching for finding the co-planarity and skew of IC leads, a statistical analysis was performed. The results of the new algorithm were compared to that of the traditional algorithm by comparing the mean and standard deviation and then calculating a standard value based on the mean plus the standard deviation multiplied by three. The standard value was compared versus the standard algorithm and the specified minimum for co-planarity of IC leads. Next, a confidence interval was calculated of 99.87% and the Type I error was found. Type I error refers to the detection of false positives. The value of the Type I error was not given.

2.3 Summary

This chapter provided a review of the literature relevant to existing automated sorting machines used in industrial applications. The vision system physical setups were shown with lighting and camera options, conveyors and backgrounds. The algorithms used for edge detection, boundary definition, object classification and alternative methods were discussed. The systems in practice were in many cases expensive, heavy and required zero movement once constructed. The systems lacked modularity and in many cases an actuation method to sort the objects of interest. The next chapter provides the framework and methodology to create a small package, inexpensive sorting machine.

CHAPTER 3. RESEARCH METHODOLOGY

This chapter provides the framework and methodology to be used in the research study. The purpose of this study is to design a system which is able to classify and sort pegs based on physical length and place them into bins in predetermined quantities. The system consists of an electromechanical system, camera interface, embedded controller and PLC. The preliminary electrical block diagram is presented in Figure 3.1 and the preliminary systems mechanical block diagram is presented in Figure 3.2. All 3D modeling and design was completed using Solidworks 2017. Post proposal, a sum of money was acquired by Purdue from Lafayette Instruments in return for completion of the sorting system. Due to increased monetary resources, the design was improved to be better suited for an industrial environment. The revised systems electrical block diagram can be found in Figure 3.3 and the revised systems mechanical block diagram in Figure 3.4.

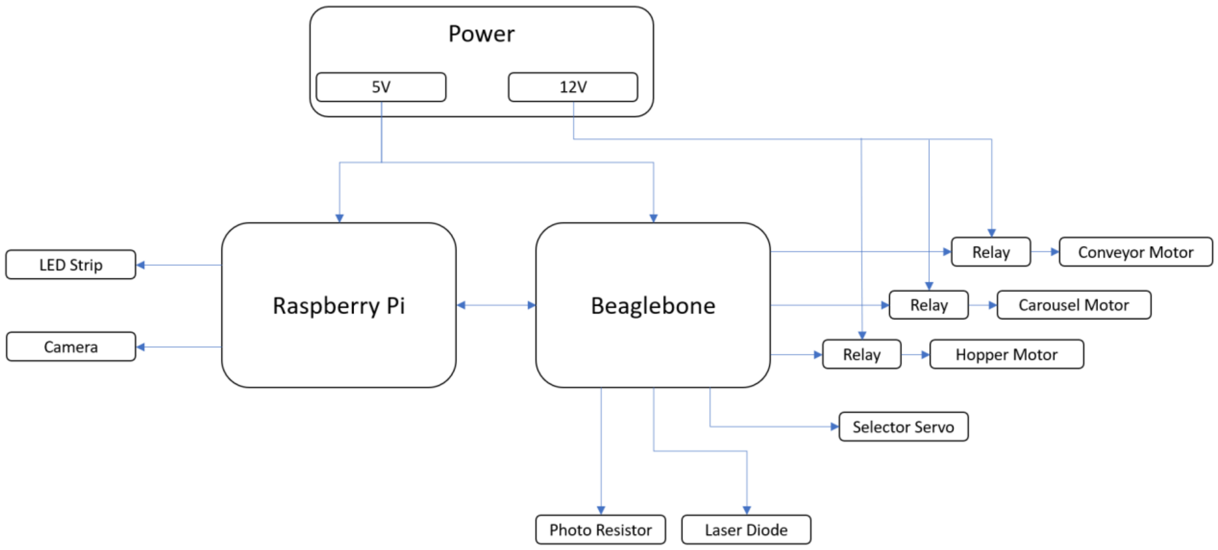


Figure 3-1: Preliminary Sorting System Electrical Block Diagram

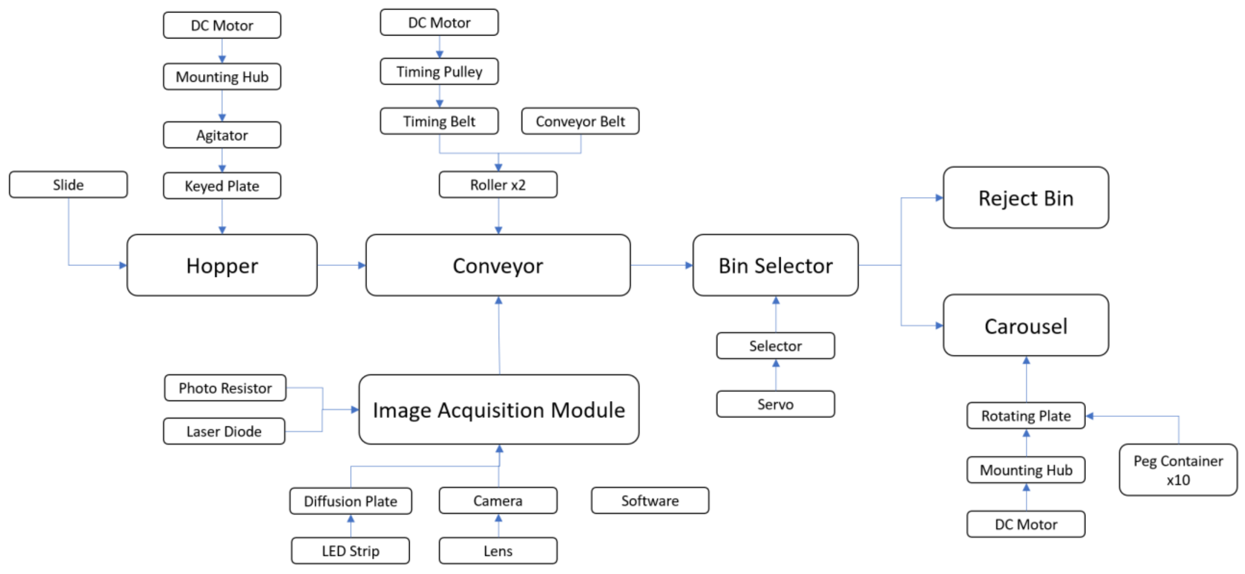


Figure 3-2: Preliminary Sorting System Mechanical Block Diagram

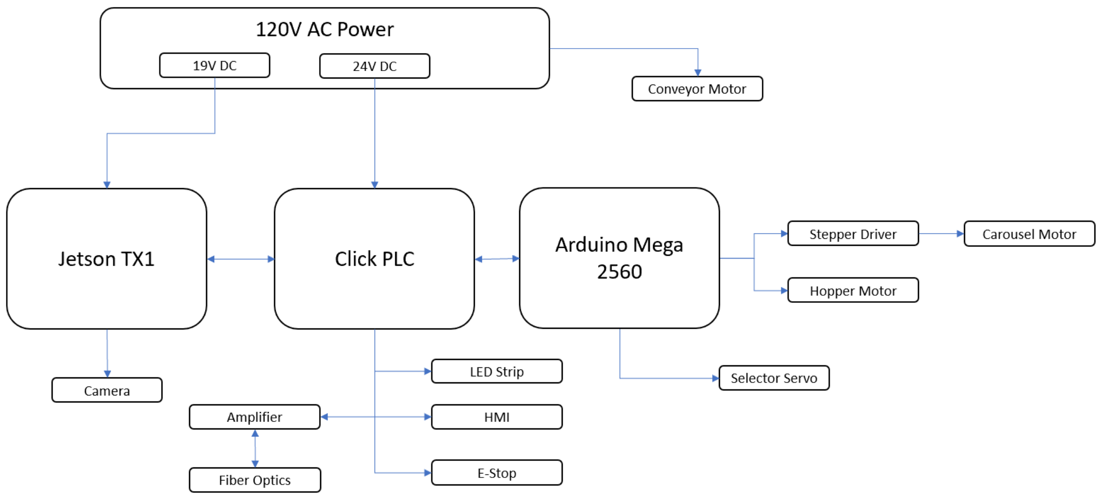


Figure 3-3: Final Sorting System Electrical Block Diagram

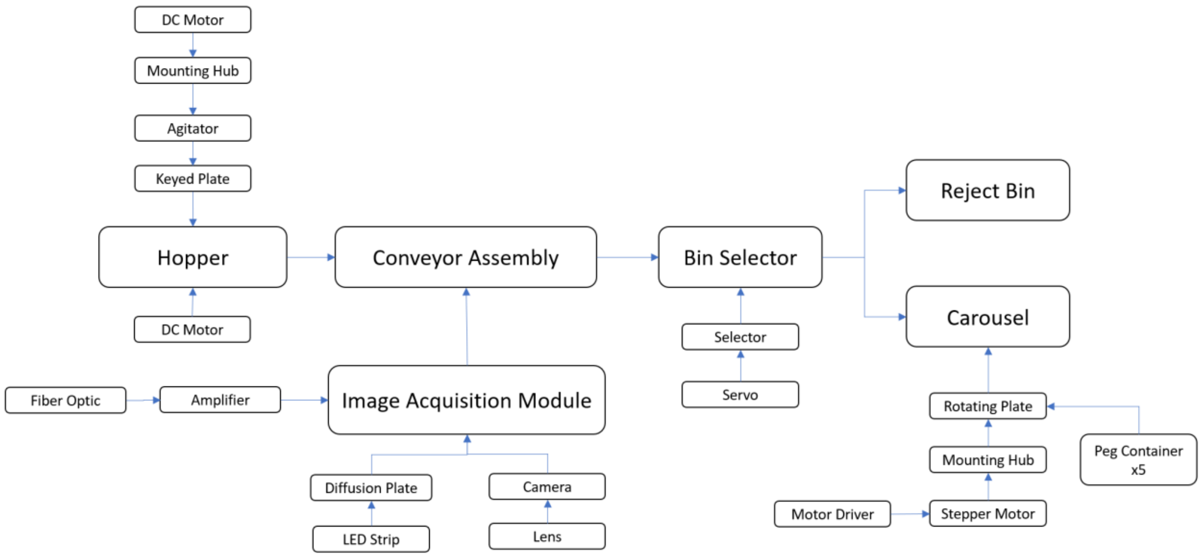


Figure 3-4: Final Sorting System Mechanical Block Diagram

3.1 <u>Initial System Design</u>

The system described in the following section is the initial plan and design for the sorting system. The construction was primarily made of acrylic and 3D printed parts for use in a low budget and non-industrial scenario. Figure 3.5 is an image of the overall system design. The initial design consisted of several different sections. Broken down these were the rotating hopper, conveyor, camera system, bin selector, bin carousel and system controls through an embedded system.

3.1.1 Hopper

The rotating hopper was based off a design by Christopher Robinson (Robinson, 2013) and was used to orientate and separate the pegs. The design was altered to be 3D printed with a larger hopper and a separator plate that would sort pegs instead of bolts and screws. In the initial design, an acrylic plate possessed cutouts slightly larger than the diameter and length of the pegs with a thickness slightly larger than the pegs. The pegs would be captured in the acrylic plate's cutouts and then as the plate rotates by use of DC motor, they were transported individually to the top of the hopper. They were released through a small cutout in the hopper, falling through a slide onto the conveyor below. The hopper design was kept in the final system design.

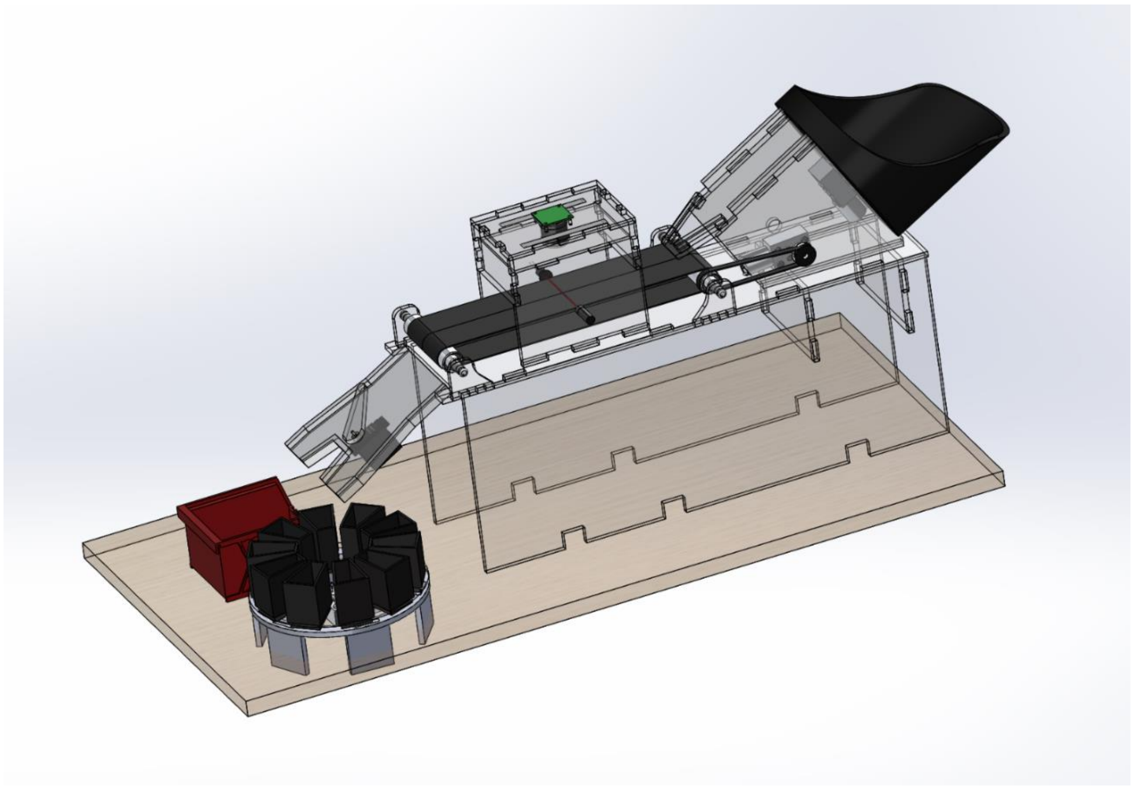


Figure 3-5: Preliminary Assembly Model of Sorting System

3.1.2 Conveyor

The conveyor was belt driven by a mounted DC gear motor. A staggered timing belt and a pulley with teeth were used in the design and the belt attached between the conveyor pulleys and the DC motor. Friction between the belt and pulley were used to ensure the conveyor was driven by the DC motor. The conveyor itself was made up of two rollers supported by quarter-inch acrylic pulley mounts. The conveyor belt was 3.28 inches across and approximately 12.5 inches long. The conveyor was used to transport the pegs under a camera system. Once the system had decided whether the peg possessed conforming dimensional characteristics, the peg was moved to the end of the conveyor belt, falling off into the bin selector.

3.1.3 Camera System

Located above the conveyor system was the vision system to determine if the pegs passing on the assembly line would be placed in the conforming or non-conforming bin. As was discussed in the review of literature section, proper lighting is a very important element of the machine vision system. In the system designed for the sorting of pegs, six white LEDs were placed behind the camera. In order to reduce shadows as well as reflections off the highly reflective surface of the pegs, additional mechanisms were used to diffuse the light inside the chamber. The light was diffused through an eighth-inch frosted acrylic sheet before emitting onto the conveyor surface.

The camera being used in this design was an Arducam OV5647, 5 Megapixel camera. Attached to the camera was a CS Mount lens with integral IR filter. The camera resolution allowed for a still frame picture of 2592x1944 pixels to be acquired. The camera was actuated by a combination of a 650nm laser diode and a photo-resistor. The laser diode emits across the top of the conveyor belt and onto the photo-resistor on the opposite side. When a peg was moved to a position interrupting the emitting laser, a signal would have been sent actuating the camera within the system to acquire an image. This actuation system would have decreased overall image processing by reducing the need for real time image acquisition. The overall function of the camera system was to measure the dimensions of each peg passing on the conveyor belt. The pegs would have been sorted later in the process. The machine vision system was also responsible for the counting of the total number of conforming and non-conforming pegs passed to ensure a predetermined number were allowed into each bin.

3.1.4 Bin Selector

The bin selector was based on a design presented by Prodieco (2016), which was used to determine which container the pegs falling from the conveyor will land in. The peg decision was based on given dimensional criteria and classification as conforming or non-conforming. The bin selector sat at an angle to the conveyor and used a quarter-inch acrylic selector to redirect the falling pegs. The selector was driven by an MG995 Servo controlled by the central micro-controller.

3.1.5 Bin Carousel

The pegs would have been divided into predetermined quantities to later be placed in bags manually. The bin carousel rotated the bins into the proper position to catch the conforming pegs. The bin carousel consisted of ten separate 3D printed containers. The 3D printed containers were smaller at their base allowing them to remain vertical while also having effortless removal for pouring the pegs into their bags. A DC motor and a rotating plate which was keyed to hold the containers would have been used to spin the containers and properly position them under the end of the bin selector. The micro-controller would have then decided when bins were full and when to turn to the next empty bin.

3.2 Preliminary Design Software Control

The system was developed on two separate communicating platforms. The Raspberry Pi was used for image acquisition as well as image analysis and calculation of peg length. Communication with an embedded system such as a Beaglebone allowed for control of motors and actuators based on image data.

The overall sorting system would have been controlled by software to dictate peg location, time and overall decision making and actuation of the electromechanical components. For the device controlling the overall system, a Beaglebone was used. The Beaglebone's code would have been written to control the pegs from the time it is separated to the bin it ended in.

The first step in the software was initialization of all the electromechanical devices. Initialization consisted of turning on the motors for the separator and conveyor as well as setting the servo position and turning on the laser system. The Beaglebone was in charge of doing multiple different processes at the same time. The first process was to watch the separator plate to know a peg had gotten stuck. This would have been completed by watching the voltage of the DC motor. If the voltage dropped sharply, the motor would have been stuck. The motor would have then back stepped and then continued forward again. The Beaglebone was also responsible for watching the laser detection module. If the laser was interrupted, it would temporarily disable the module, send instructions to the Raspberry Pi, actuating the camera system and then re-enable the laser after image acquisition. Based on the data received from the camera system, the Beaglebone would have decided the servo location to sort the conforming and non-conforming pegs. All conforming pegs that were sorted would have been counted until the predetermined container capacity had been met. Once met, the motor which controlled container location would have been activated to move the next container to the proper location. Once all ten containers had been filled, the system would shut down until reset by human interaction.

3.2.1 Image Acquisition Module

The image system was be driven by a Raspberry Pi 3 Model B. The image would have first been acquired through the Arducam camera. The actuation of the camera would have been from the laser system above the conveyor. Once acquired, the image was stored and saved. The image went through preprocessing, processing and classification before determining if the peg conformed to the specified dimension range.

3.3 Revised System Design

The system described in the follow section was for the revised design of the sorting system. The construction and frame material were primarily extruded aluminum with an industrial grade camera and conveyor system. Figure 3.6 is an image of the overall system design. The system consisted of several different sections. Broken down these were the rotating hopper, conveyor, camera system, bin selector, bin carousel and system controls through communicating embedded system and PLC.

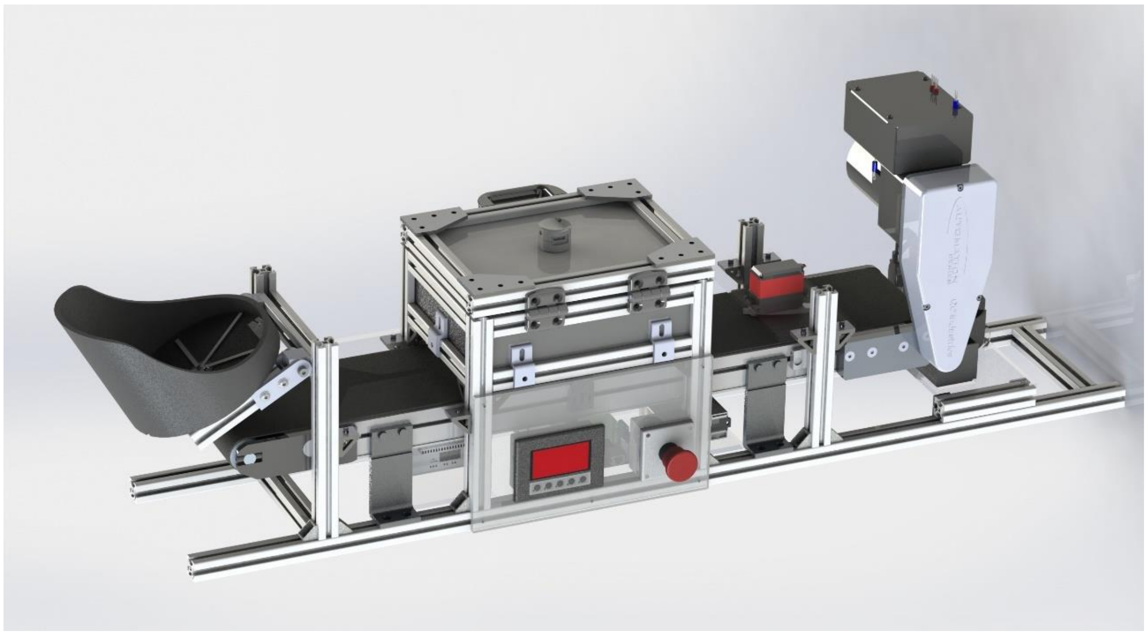


Figure 3-6: Revised Assembly Model of Sorting System

3.3.1 Rotating Hopper

The rotating hopper was based from a design by Christopher Robinson (2013) which was used to orientate and separate the pegs. The design was modified using a larger hopper, agitator and peg specific separator plate. The hopper was made up of a rotating plate, agitator, hopper housing and a slide. The plate was designed with specifically dimensioned cutouts that allowed pegs of a specific size and orientation to be carried around the hopper. Once inside the cutouts, the pegs were carried to the top where a cutout was placed in the hopper housing. The plate was turned by a mounted 24V DC gear motor. The pegs would have then fell from the hopper and dropped onto the conveyor belt. The hopper included an agitator which orientated pegs that were not parallel to the cutouts in the rotating plate. The agitator was made from quarter-inch acrylic and followed the rotation of the rotating plate. The agitator extruded above the separator plate to orientate and push the pegs at the bottom of the hopper. The plate was made from sixteenth-inch acrylic and required a swap with other plates depending on which of the three pegs was being sorted. This was necessary as larger pegs would not have fit in the cutouts in the rotating plate. The plate for the small pegs was revised post build to account for the diameter of the small pegs. The front section of the hopper was bowed outwards. This allowed a very large number of pins to be placed in the hopper at a single time to decrease overall human interruption in the process.

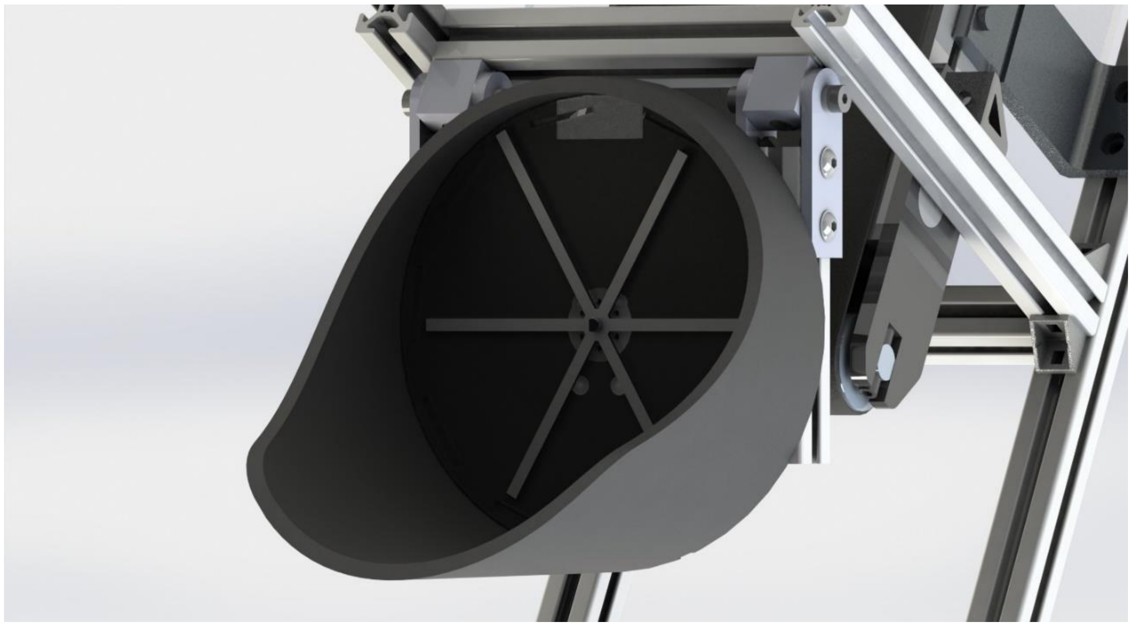


Figure 3-7: Hopper Assembly for Isolation of Pegs

3.3.2 Conveyor

The conveyor, shown in Figure 3-8 was designed and produced by QC Industries LLC and purchased through Neff Automation (Neff Automation). The operation was to carry the pegs from the hopper, through the lighting enclosure so that an image may be acquired of each individual peg. The conveyor was six inches in width and 36 inches in length which was the shortest length conveyor that is produced by QC Industries. The conveyor belt that was used had a black dimple top cut resistor belt for high contrast to the metal pegs. The dimple top helped to hold the pegs and keep them from rolling during transportation under the camera system. The conveyor had a flip up tail allowing for easy removing and replacement of the belt. The conveyor was driven by an Oriental Motors USM540 40-watt single phase 120-volt AC motor connected to a 30:1 gear reducer. The configuration allowed for a speed of 13 to 26 feet per minute. This would have allowed a theoretical sorting speed of 120 pegs per minute. This calculation is allowed one inch of space between each peg at a conveyor rate speed of 20 ft/min. The aluminum frame came with raised side mounts allowing the conveyor to be connected above an aluminum t-slot frame.

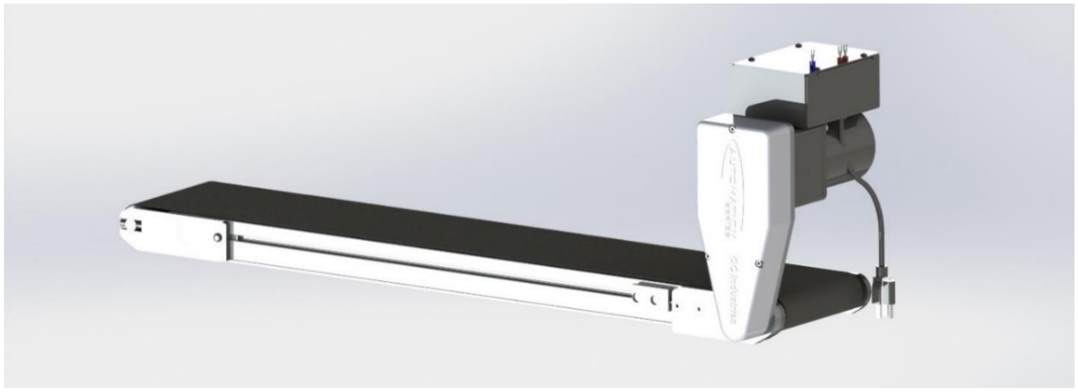


Figure 3-8: Conveyor Assembly for Transportation of Pegs

3.3.3 Camera System

The camera system is shown in Figure 3-9. This section of the system is responsible for camera actuation and image acquisition.

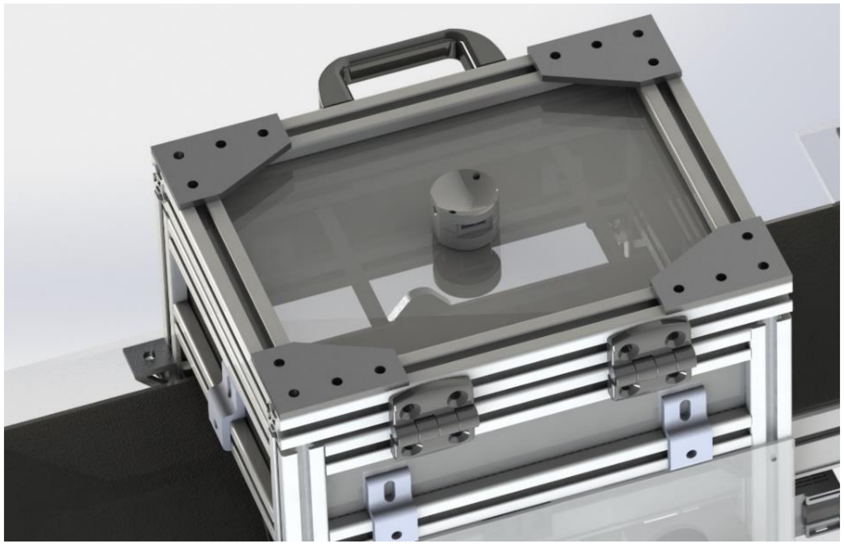


Figure 3-9: Overall Camera System

3.3.3.1 Lighting Enclosure

The lighting enclosure was made of one-inch t-slot aluminum rails by 80/20 Inc. The enclosure was 4.9" x 9" x 12.3" and had a 0.95" gap between the enclosure and the conveyor belt. Enclosure size was determined by the focal length of the camera and lens combination. At the top of the enclosure were a set of Ecolocity RL-SC-RSU24-T-W-10 LEDs to simulate all RGB components of natural light for a total of 36 LEDs. The LEDs were to produce 372 lumen per foot. These LEDs were arranged in a square pattern to ensure equal distribution of light on the conveyor belt. A diffusion layer made from eighth-inch frosted acrylic was also added between the LEDs and the conveyor to minimize direct reflections of the LEDs on the metallic surface of the pegs.

3.3.3.2 Camera, Len and Filter

The puA1280-54uc camera was selected from the Basler Pulse camera series. The puA1280-54uc was a color camera with USB3.0 data connection and a resolution of 1280 x 960 pixels. It came with holes for mounting of the camera onto a plate. The camera possessed a global shutter which was ideal for moving objects. The camera had a frame rate of 54 frames per second. It was able to be connected to a C-Mount Lens. The camera was compared to similar products from both the Dart and Pulse series. The camera was chosen from the pulse series as it allowed a USB 3.0 connection unlike the ACE series. The Dart series was constructed with an open back allowing the PCB to be visible. The Pulse series was fully enclosed and was one-third the price of a similar ACE series camera.

The lens was chosen according to the desired region of interest. The region of interest was determined by taking the size of a peg and adding two inches to the width and height to allow for the background to be included in the image. The ROI was determined to be 3.000in x 2.063in as a minimum. The equation presented in Equation 3.1 was used to determine the magnification factor and Equation 3.2 to determine the lens focal length required (Alper, 2011).

$$\begin{gathered}\text{Magnification} = SensorSize/FieldofView(width)\\ \text{Magnification} = 6mm/(3*25.4)mm\\ \text{Magnification} = 6\text{mm}/76.2\text{mm}\\ \text{Magnification} = 0.0787\end{gathered} \tag{3.1}$$

$$\text{Focal Length} = \text{Magnification} * \text{WorkingDist}/(1 + \text{Magnification})$$
$$\text{Focal Length} = 0.0787 * 133.35\text{mm}/(1 + 0.0787) \qquad (3.2)$$
$$\text{Focal Length} = 9.73\text{mm}$$

The focal length was calculated to be 9.73mm, so the Basler C125-0818-5M lens was chosen which has a focal length of 8mm. The field of view was recalculated through the machine vision store's online entrocentric lens calculator that the new field of view was 3.15"x2.36" (Machine Vision Store, 2018). The 3.15" (80mm) was then divided by total pixels (1280) to get a resolution of 0.0625mm/pixel.

3.3.3.3 Peg Detection

A fiber optic sensors were used for peg detection. Fiber optics were chosen as the method for detection of the object as the beam width was narrow and could be completely blocked by 0.063" diameter pegs unlike other light and proximity sensors. When a peg traveled underneath the camera, it broke the fiber optic beam and triggered the camera to take a picture. The fiber optic cable chosen was the CF-TB2-20 along with the DFP-AN-1A amplifier. The fiber optic system was able to be used at a range of 700mm or lower and had an amplifier input voltage of 24V to communicate directly to the PLC.

3.3.4 Bin Selector

Due to concerns from the initial system design that the pegs may have bounced off the bin selector. The selector had been moved above the conveyor as seen in Figure 3-10. The flipper was attached to a servo motor driven by the Jetson TX1. The flipper was used to move the pegs so that they would fall either in the reject container or the conforming bins and counted. The selected servo was the TGY-1270HV by Turnigey. This servo possessed metal internal gears as well as metal teeth to create a press fit with the flipper. The servos flipper was designed to rest on the conveyor belt to ensure a peg cannot travel underneath the servo.

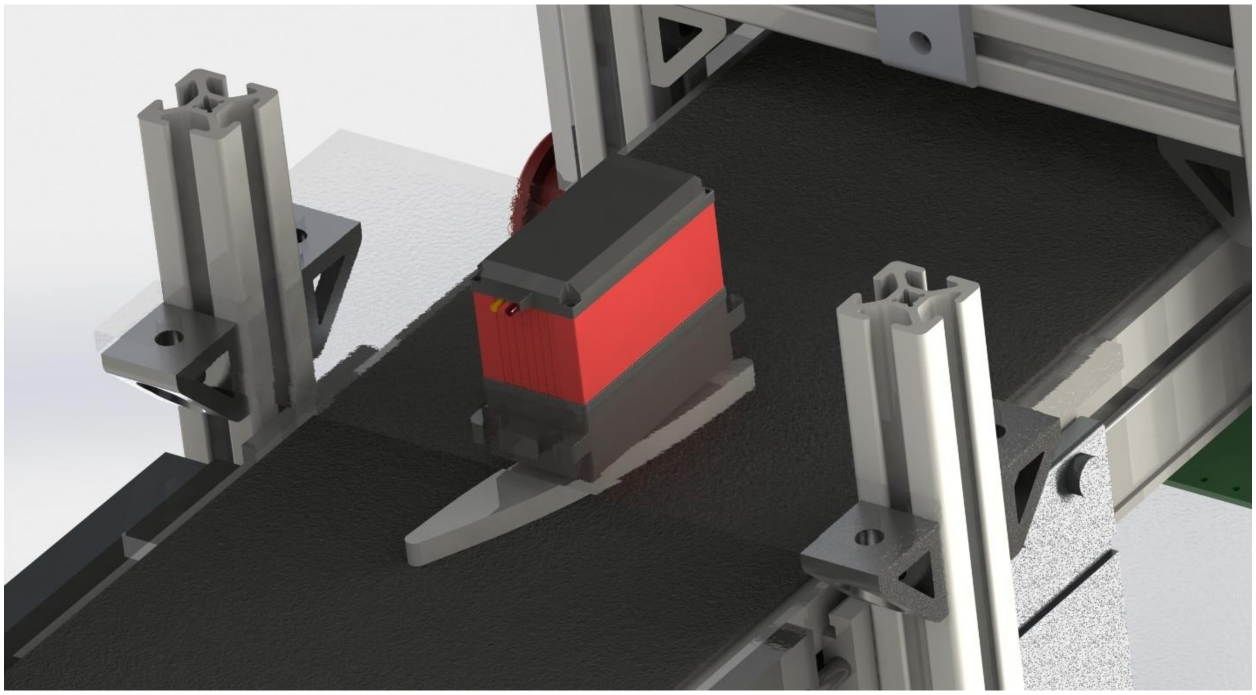


Figure 3-10: Selector for Peg Sorting

3.3.5 Bin Carousel

The pegs were divided into predetermined quantities to later be placed in bags manually. The bin carousel shown in Figure 3-11, was used to rotate the bins into the proper position to catch the conforming pegs. The bin carousel consisted of five separate 3D printed containers. The 3D printed containers were smaller at their base allowing them to remain vertical while also having effortless removal for pouring the pegs into their bags. A DC motor and a rotating plate which was keyed to hold the containers were used to spin the containers and properly position them under the end of the bin selector. The carousel sat on top of a frame with a circular cutout on top. The circular cutout was to hold up the rotating plate while only touching under the outside of the plate. The minimal contact ensured smooth rotation by the DC motor. The micro-controller was used to decide when bins were full and when to turn to the next empty bin.

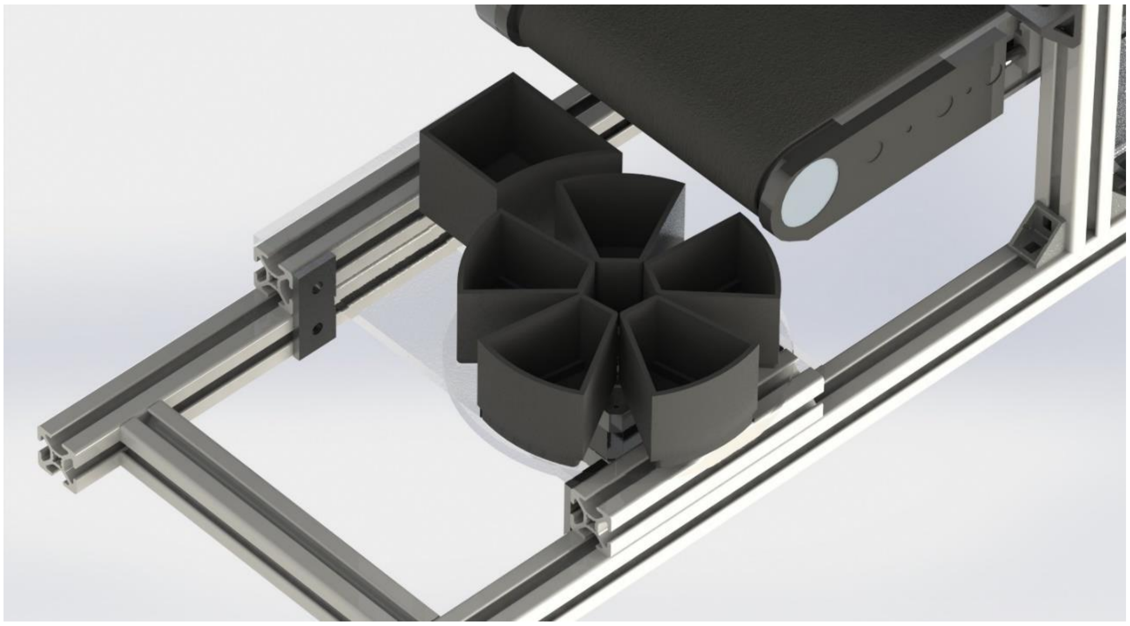


Figure 3-11: Carousel with Bins for Containment of Sorted Pegs

3.3.6 Software Control

The following section outlines the overall software and algorithms to be used for control of the mechanical system. The system was developed on two separate communicating platforms. The Jetson TX1 was used for image acquisition as well as image analysis and calculation of peg length. Communication with a Click PLC was used for overall system control. The PLC controlled the motors and steppers to ensure proper system functionality. A comparison was made between similar platforms for the image analysis processing in Table 3-1.

Table 3-1: Comparison of Development Boards

	Jetson TX1	Jetson TK1	Zyng 7000	STM32F7
GPU	256 Cuda Cores	196 Cuda Cores	N/A	N/A
CPU	ARM Cortex A15	ARM Cortex A57	ARM Cortex A9	ARM Cortex M7
Bluetooth	Yes	No	No	No
Wi-Fi	Yes	No	No	Expandable
Ethernet	Yes	Yes	Yes	No
HDMI	Yes	Yes	Yes	No
SD Slot	Yes	Yes	Yes	No
USB	Yes	Yes	Yes	Expanable
Price	$499.00	$129.00	$895.00	$38.22

The Nvidia Jetson TX1 was chosen as the image processing tool of choice. While the STM32F7 was cheaper, it did not possess many external peripherals. The STM32F7 lacked ethernet not allowing it to connect to a network and the lack or HMDI and USB would have made programming and testing more difficult. The ZynQ-7000- also lacked several important features. The SOC only possessed an ARM Cortex A9 which was substantially slower than the A15 and the A57 processors. It also lacked the functionality of Bluetooth and Wi-Fi. This came at a high price of $895 compared to the below $500 price of all other development boards.

The Nvidia Jetson TX1 and Nvidia Jetson TK1 were very comparable in operation. Both came with dedicated GPUs with CUDA cores and quad core processors. The price was higher for the TX1, however, the Bluetooth and Wi-Fi peripherals were important features to the customer for data transfer and connectivity as well as ease of development.

3.3.7 Image Acquisition Module

The image system was to be driven by a Nvidia Jetson TX1. The flowchart presented in Figure 3-12 shows the logic for the overall imaging system. The image was first be acquired through the Basler Camera. The actuation of the camera was from the fiber optic system across the conveyor. Once acquired, the image was be stored and saved. The image went through preprocessing, processing and classification before determining if the peg conformed to the specified dimension range.

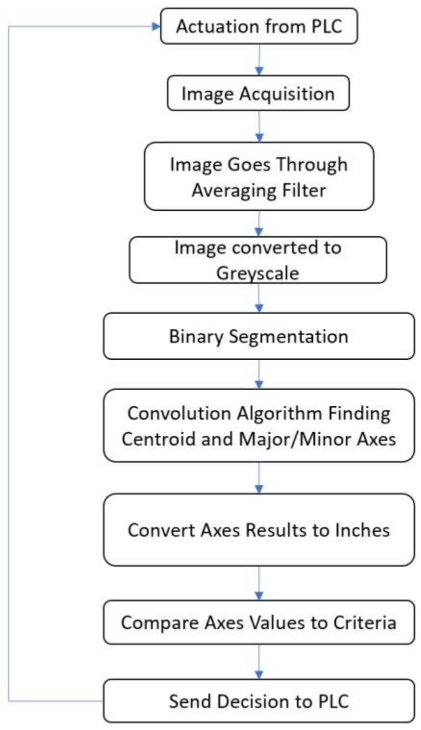

Actuation from PLC
Image Acquisition
Image Goes Through Averaging Filter
Image converted to Greyscale
Binary Segmentation
Convolution Algorithm Finding Centroid and Major/Minor Axes
Convert Axes Results to Inches
Compare Axes Values to Criteria
Send Decision to PLC

Figure 3-12: Overall Imaging System Flowchart

Preprocessing of the image was an important step to ensure the image would be ready for edge detection and feature extraction. The first algorithm that was used is the averaging filter. The averaging filter took the original image and averaged 3x3 matrices of all non-border image pixels, saving them into a new image. The image was converted to gray-scale by taking the value of each RGB color, dividing by three and adding them all together. The new gray-scaled image was then ready for segmentation. Segmentation was used with a threshold to create a binary image. All pixel values above the threshold were to be set to 255 and all below the threshold were set to zero. Segmentation was important to create a clear boundary between the object and its background.

The next step was to process the image and set a boundary around the peg in the image allowing for determination of the overall length of the peg. First, a convolution method was used. An example of a possible algorithm would be the Laplacian operator. This operator operates by defining the boundary of the object by use of another 3x3 matrix operation. The center pixel is multiplied by eight and then decreased by the summation of all the surrounding pixel values. This method allows for all areas of change (edges) in the image to be captured and all other areas set to zero. With all edges defined, the next step was to find the major and minor axis. In the image, the major axis represented the length of the peg and the minor axis represented the width or diameter of the peg. The major and minor axis dimensions were saved to be later used. The peg was sorted based on criteria predetermined by the objects designed dimensions. For example, a 1"x0.063" may be desired. A tolerance of ±0.05” was then applied and a range of acceptable values were created. In this example, pegs that had a length between 0.95 and 1.05 inches were classified as conforming. Pegs with values outside of this range were classified as non-conforming and were separated. The decision was sent to the PLC through onboard GPIOs or general-purpose input/output pins.

3.3.8 Sorting System PLC Control

The overall sorting system had to be controlled by software to dictate peg location, time and overall decision making and actuation of the electromechanical components. A C0-00DR-D Click PLC was used for overall system control. The PLC code was written to control the pegs from the time they were separated to the bins they ended in. A flowchart of the code is shown in Figure 3-13. The Click PLC was connected to a C-more EA1-S3MLW-N three inch monochrome human machine interface (HMI). The HMI allowed for the operator to enter what peg was being sorted and the quantity that was needed to be counted to for each sorted bin.

The Click PLC included a free download of the CLICK Programming software. This software was used to code the PLC for operation.

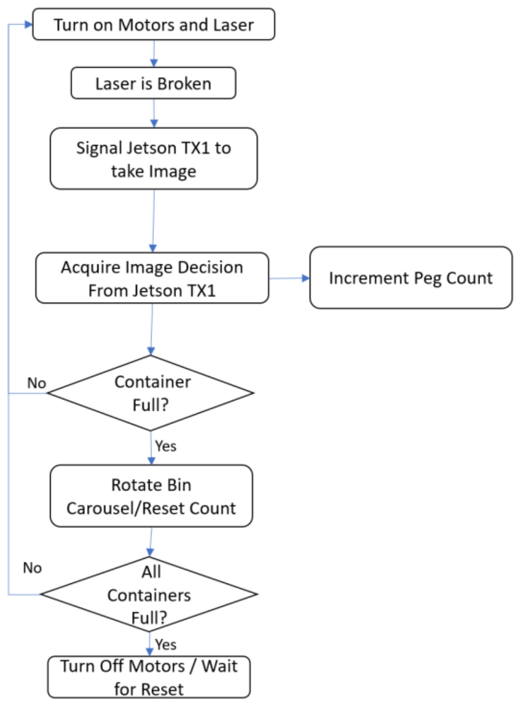

Turn on Motors and Laser
Laser is Broken
Signal Jetson TX1 to take Image
Acquire Image Decision From Jetson TX1
Increment Peg Count
Container Full?
No
Yes
Rotate Bin Carousel/Reset Count
All Containers Full?
No
Yes
Turn Off Motors / Wait for Reset

Figure 3-13: PLC Logic Flowchart

3.3.9 Power Requirements

The overall system contained several different parts that required power. The Jetson TK1 operated from an included 12V DC supply. The rest of the system ran on 24V. The current requirements were 120mA at 24V for the PLC, 220mA at 24V for the HMI, 200mA max at 24V for the fiber optic amplifiers, 450mA max at 24V for the LEDs, and 200mA at 24V for the DC motor. This created a requirement of 1.2A at 24V. The CO-01AC was chosen for its capabilities of 24V at 1.3A continuous and 1.6A before overprotection.

3.3.10 Summary

This chapter provided the framework and methodology of the study. The 3D model and flow charts used represented the overall operation of the system in the sorting of pegs.

CHAPTER 4. RESULTS

The following section guides through the mechanical assembly of the sorting system along with revisions and development of code for the various embedded systems utilized in the study. The sorting system requires the integration of both mechanical and electrical hardware. Revisions to the design were made during assembly and preliminary testing to provide proper functionality.

4.1 Mechanical Build

The mechanical build of the system allowed for the framework and electromechanical devices to work together to accomplish a common goal. Revisions were made to the design throughout the build process. The revisions were to account for tolerances, intersections of multiple parts and the addition of parts necessary to ensure devices and sections of the frame were securely mounted in place.

4.1.1 System Frame

The system frame was constructed from extruded aluminum procured from 80/20 Inc. All aluminum rails were one inch by one inch with a t-slot design on all four sides and a smooth finish. Figure 4-1 represents a fully assembled model of the aluminum frame. The exploded view with the bill of material (BOM) for the subassembly can be found in Appendix A. The drawing of the frame with dimensions is found in Appendix B. Two, 48-inch rails ran the length of the system and allowed all other parts and subassemblies to be mounted to them. The width between the bars was ensured by installing seven-inch rails secured between the 48-inch rails using M6 bolts, corner brackets and slide in t-nuts. All other parts of the aluminum frame were considered part of the other subassemblies.

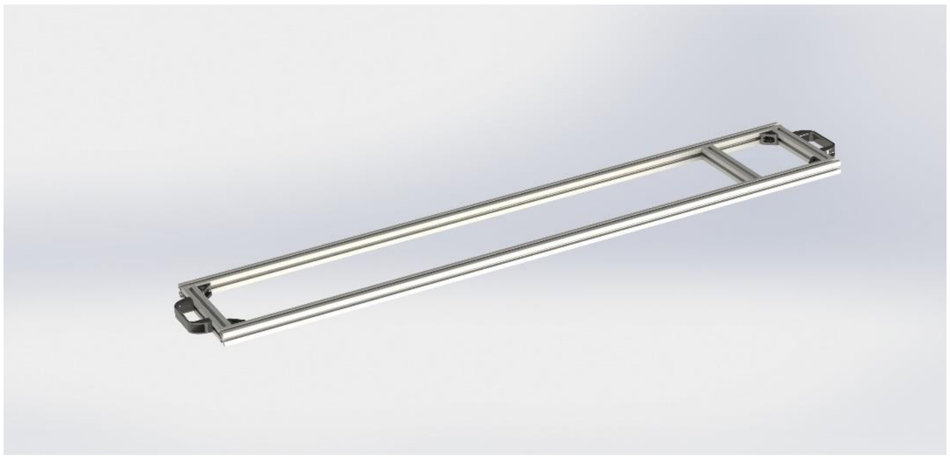


Figure 4-1: System Frame Subassembly

4.1.2 Hopper Subassembly

The hopper subassembly was constructed using a combination of extruded aluminum, various sizes of laser cut acrylic and 3D printed black PLA. The hopper casing was printed using a Lulzbot Taz 6 3D printer supplied by Purdue University. The acrylic and laser, also supplied by Purdue University, were used to construct the three types of separator plates, the agitator, and the pin positioner plates. The subassembly drawings and subassembly drawings with dimensions are found in Appendix A and B respectively. The 24-volt DC motor was first connected to the separator casing using four M3 machine screws. The separator casing was connected to the five-inch aluminum rails using M6 screws. A flat surface under the separator plate was created by countersinking the screws into the back of the separator casing to hide each screws head. The five-inch aluminum rails were joined together by right angle pivot joints and another five-inch rail. The pivot joints were installed on a seven-inch rail which connected to two 9.75-inch rails perpendicular to the frame. During initial testing, the small pegs were able to fall between the gaps of the outside of the separator plate and get stuck beneath the plate. A reduction of the occurrence of stuck pegs was implemented by designing and installing a separator back plate between the separator plate and the separator casing.

The pin positioners were used to center the pegs before entering the lighting enclosure. Initial testing showed that the small pegs got stuck on the side of these and rolled several times before straightening and continuing along the length of the conveyor. The peg positioners were revised to have a much larger radius to no longer give the pegs a linear surface to roll against. An assembled view of the hopper subassembly is shown in Figure 4.2. Drawings for all custom parts can be found in Appendix C.

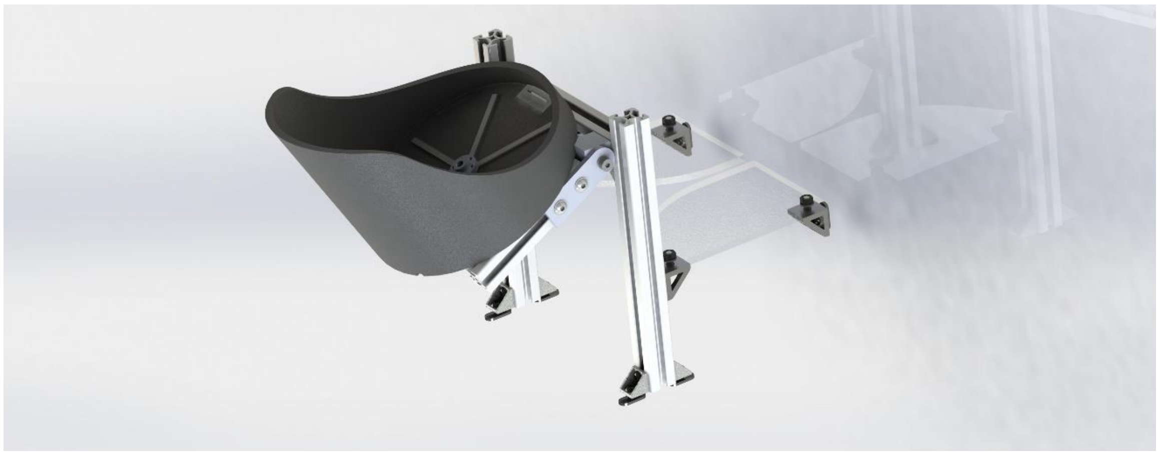


Figure 4-2: Hopper Subassembly

4.1.3 Conveyor

An exploded view of the conveyor subassembly can be found in Appendix A and a subassembly drawing with dimensions in Appendix B. The conveyor was delivered in several different parts, unassembled from the distributor, Neff Automation. The conveyor required the motor and gearbox to be attached along with the drive belt. The conveyor mounts were attached to the sides of the conveyor with the M6 bolts supplied. The length of the bolts supplied required an external washer between the mount and the head of the bolt to ensure the bolt did not bottom out within the conveyor rails. The conveyor was then mounted to the frame of the system using M6 screws and the supplied t-nuts.

Fiber optic sensors were located along the conveyor to locate each peg during its travel. A set was installed inside the lighting enclosure to actuate the camera and image processing algorithms. The fiber optic mounts were created by laser cutting from quarter inch acrylic. The fiber optic mounts were attached to the conveyor using a set of roll-in t-nuts and M6 bolts. A second set of fiber optic cables were used at the end of the conveyor to ensure the system knew when each peg had fallen into its designated container. The second set of fiber optic sensors allowed the PLC to know when the containers were full and when the last peg had fallen. The system then rotated to the next bin. The secondary set of fiber optic mounts were 3D printed and were meant to latch onto the conveyor. The design was different from the other fiber optic mounts since there is no rail to mount to at the end of the conveyor. The conveyor subassembly is shown in Figure 4.3.

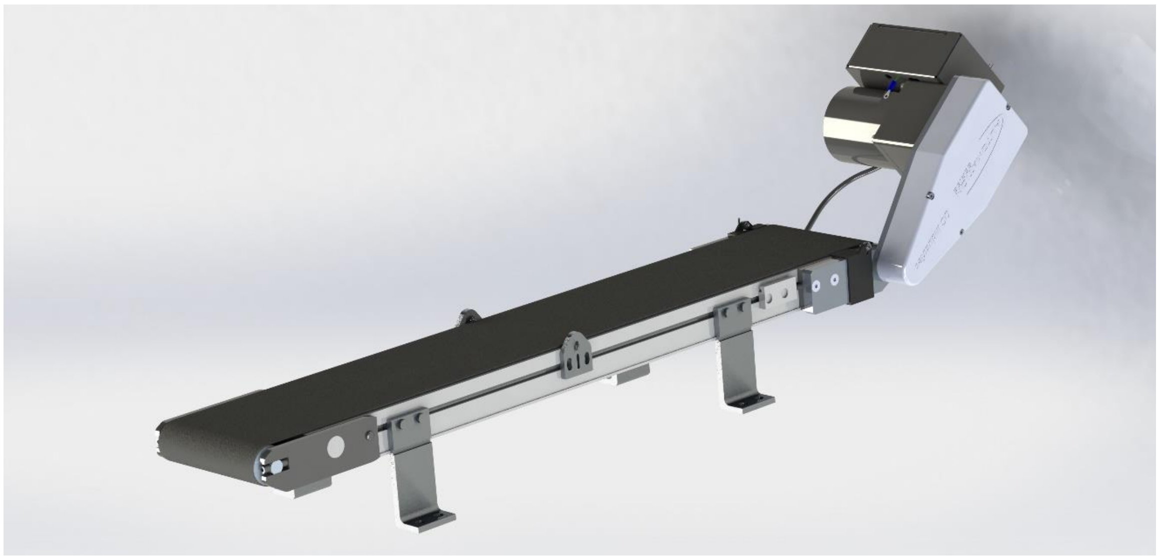


Figure 4-3: Conveyor Subassembly

4.1.4 Lighting Enclosure and Camera System

The lighting enclosure and camera system were critical sections in determining whether a peg was passing through the system and had conforming dimensions. The following section describes the assembly of the lighting enclosure body as well as the camera system which includes the enclosure top.

4.1.4.1 <u>Lighting Enclosure Body</u>

The lighting enclosure frame was constructed with aluminum t-slot rails. An exploded view of the enclosure and camera can be found in Appendix A. The sides of the enclosure were made from black PVC plastic to block out external light. The initial design proposed used several 2491 panel mounts (80/20 Inc.) to ensure that all side panels mounted properly. During assembly it was realized that the panel mounts were not necessary as the top t-slot rails held the side panels firmly in place.

4.1.4.2 Camera System and Enclosure Top

The enclosure top was also ordered with a PVC plastic panel. The PVC was to be laser cut to create the holes for the camera and screws. Due to a chlorine gas that would have been released when cutting PVC with a laser, the PVC was replaced by quarter-inch acrylic. The enclosure top slides between the rails and was held in place when all four rails of the top of the enclosure were connected. The camera mount was designed to hold the camera still and was 3D printed in black PLA. Holes were drilled in the acrylic to mount the camera to the enclosure top. With the frame of the lighting enclosure constructed and the camera mounted, the Ecolocity LEDs were cut with three LED sections per side. This totaled to 36 LEDs at 150mW each. Each strip was mounted with the adhesive supplied on the LED strips. The orientation and length of the LEDs are shown in Figure 4-4.

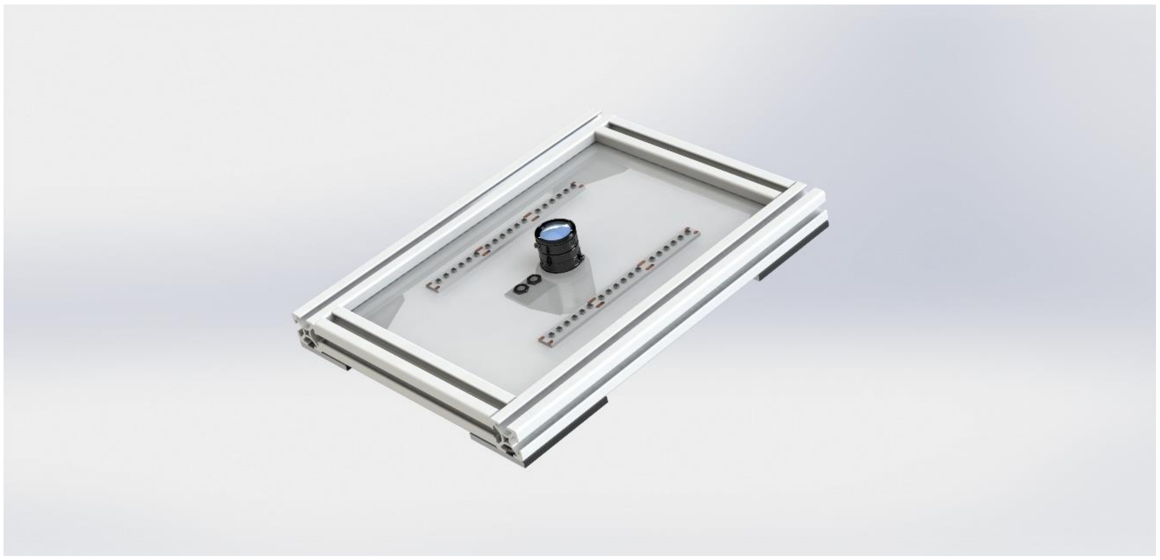


Figure 4-4: Camera Enclosure Top

The camera and lens shown in Figure 4-4 were ordered initially with an adapter ring between the detector and the lens along with another adapter that would allow the attachment of a neutral density filter. Initially, images were taken by using the Basler's software development kit (SDK). These images showed that the adapter ring degraded the image quality by effecting distance between the lens and the detector. Initial testing showed no desire for a neutral density filter. With the lens installed in front o the camera, the lens focal point was adjusted to the maximum distance using the two focusing rings on the outside of the lens.

A diffusion layer was added next for the diffusion of light and the elimination of shadows. The layer was cut from eighth-inch acrylic and was given three uniform layers of a frosted spray paint coating on each side. The frosted diffusion layer was then mounted to the enclosure top. A handle and hinges were installed, connecting the enclosure top and camera to the lighting enclosure. A fully assembled view of the camera enclosure is shown in Figure 4-5.

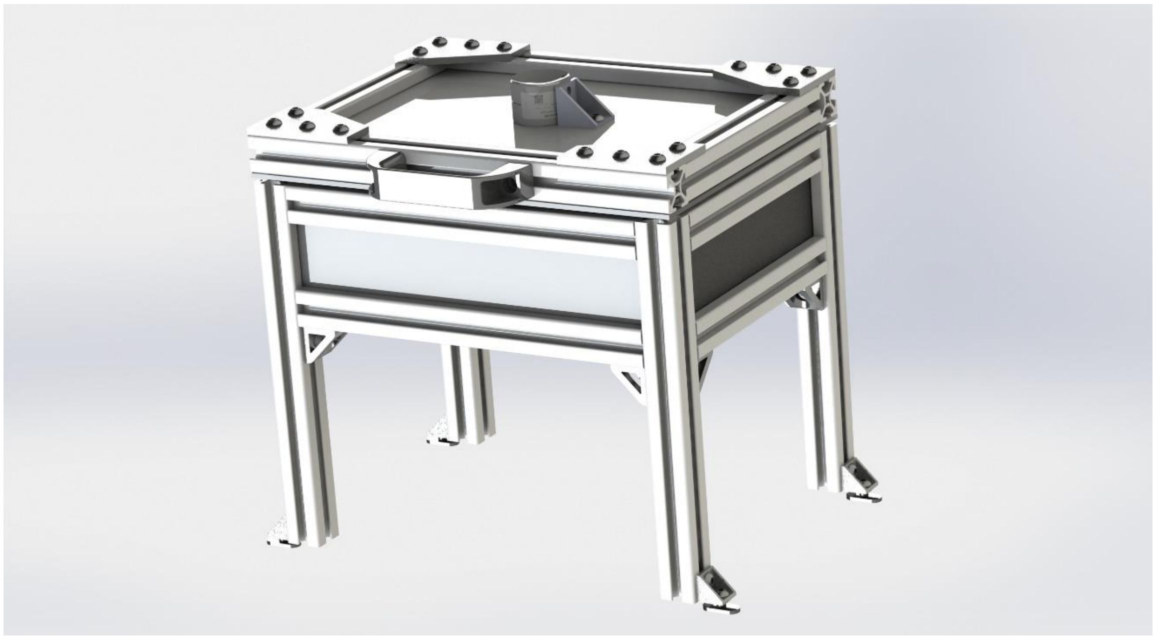


Figure 4-5: Camera Enclosure Subassembly

4.1.5 Bin Selector

An exploded view of the bin selector can be found in Appendix A and a view with dimensions in Appendix B. The bin selector was connected between two standing t-slot rails and the camera enclosure subassembly. A mounting plate for the selector servo was cut from quarter-inch acrylic and the servo was connected to the bottom of the mounting plate using 4-32 machine screws and nuts. The flipper was cut from quarter-inch acrylic and press fit to the servo head.

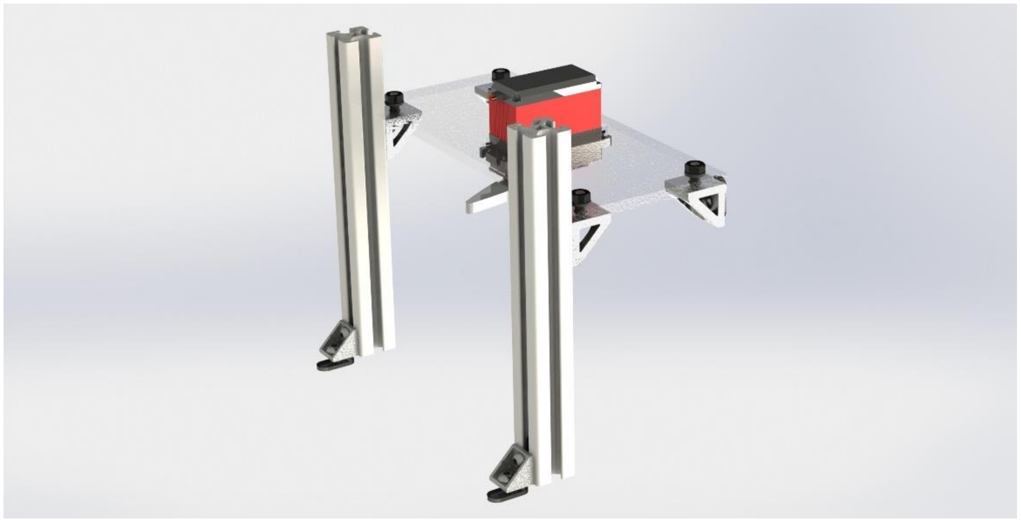


Figure 4-6: Bin Selector Subassembly

4.1.6 Bin Carousel

The bin carousel was important in collecting the sorted pegs into the predetermined quantities for the operator. An overall view of the bin carousel can be found in Figure 4-7. The bin selector was built on t-slot rails and a quarter-inch acrylic platform. A Quimat nema-17 stepper motor was connected via a six-millimeter mounting hub to the carousel plate. The plate was then populated with five bins for the conforming pegs. A reject bin was also populated into the carousel supporting plate to collect the non-conforming pegs. All bins were 3D printed with black PLA.

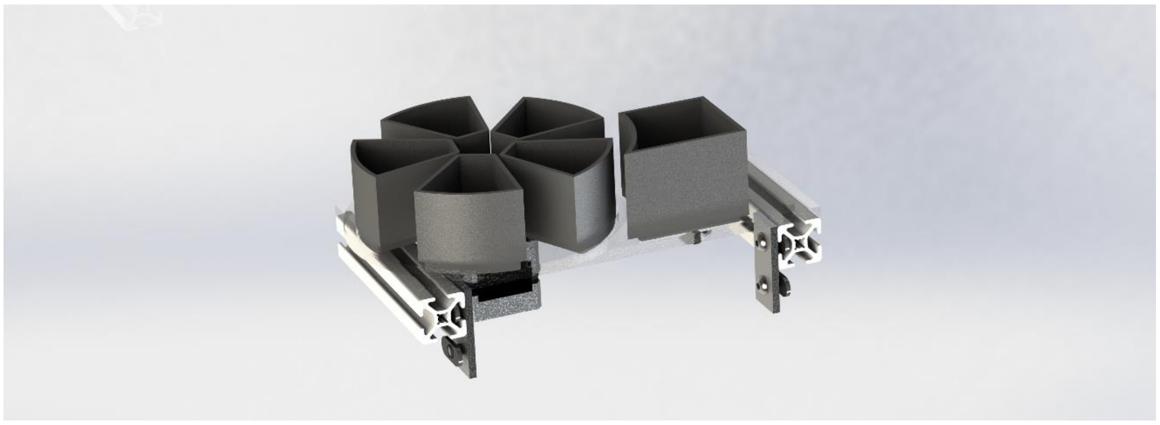


Figure 4-7: Bin Carousel Subassembly

4.1.7 Microcontrollers and PLC

The Jetson TX1 is located beneath the conveyor and lighting enclosure, mounted on a quarter-inch acrylic sheet. The added Arduino Mega 2560, discussed in future sections, was also located on the acrylic sheet along with the stepper motor driver and a relay bank PCB which allows communication between the 24V PLC and the microcontrollers. The Arduino, Jetson TX1, and relay PCB were mounted using quarter-inch standoffs to avoid solder joint damage from the joints being pressed into the acrylic. An array of miscellaneous screws and nuts were used which were supplied by Purdue University.

A Quimat TB6600 stepper motor driver was attached using a custom designed and 3D printed mount allowing the motor drivers heatsink to sit away from the acrylic. The microcontroller subassembly can be found in Appendix A along with a drawing with dimensions in Appendix B. Figure 4-8 displays the overall electronics mount.

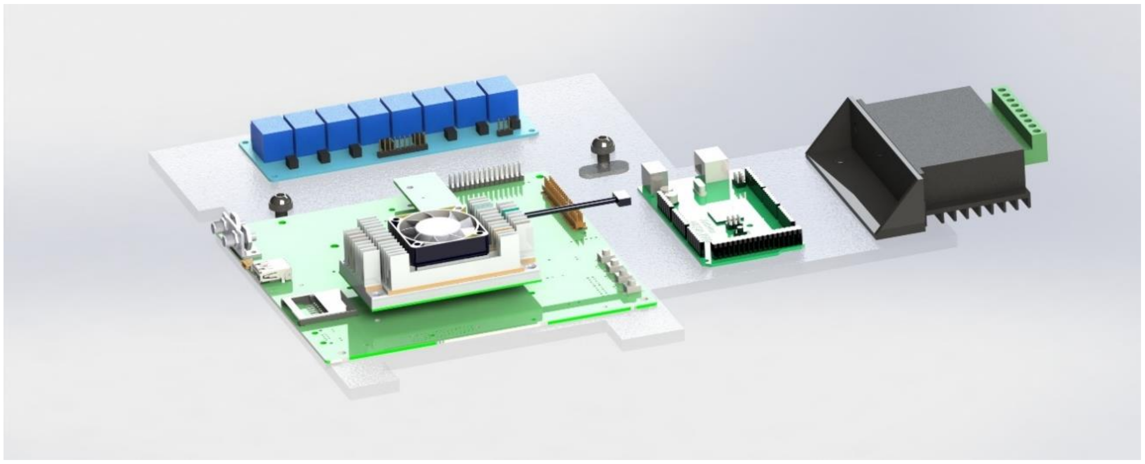


Figure 4-8: Electronics Mount Subassembly

The PLC was previously to be mounted to the plate underneath the conveyor as well. Due to height constraints, the Click PLC, power supply and fiber optic amplifiers were all mounted to a DIN rail on the back of the lighting enclosure. Wiring the multitude of components of the system to the PLC required common ground and 24V supply rails. To solve this issue DIN rail terminal strips were mounted and installed allowing five external 24V connections and five external ground connections.

During testing of the hopper subassembly, it became clear that an adjustable speed of the rotating separator plate would be necessary to create proper system timing. A DC-DC converter was used to reduce the 24V from the PLC and was mounted in a custom designed and 3D printed DIN rail mountable enclosure. The PLC subassembly can be found in Appendix A along with a drawing with dimensions in Appendix B. Figure 4.8 shows the overall PLC and DIN rail subassembly.

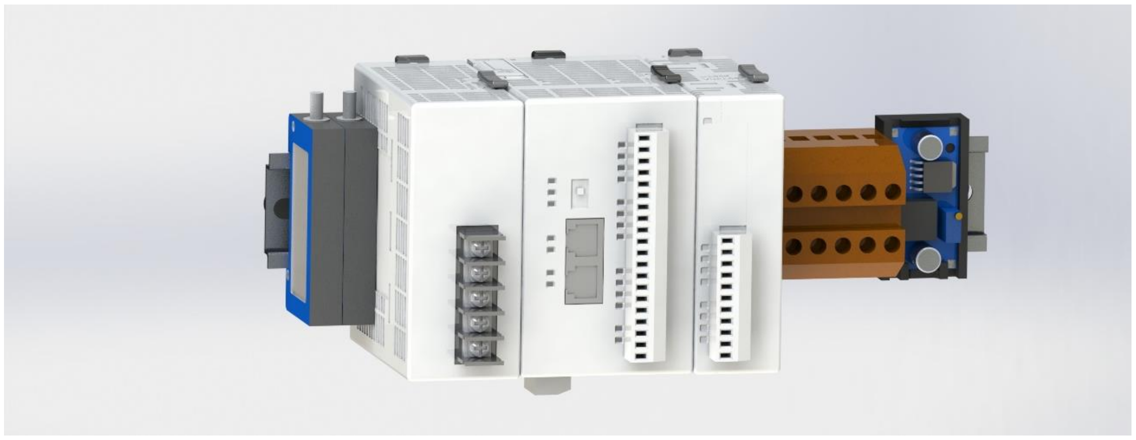


Figure 4-9: PLC and Din Rail Subassembly

A C-more EA1-S3MLW-N human machine interface (HMI) was mounted along with an emergency stop button (non-functional) to the front of the system allowing for user input and control. The HMI consisted of five pushbuttons which allowed navigation through setup screens and programming of the quantity of conforming pegs to be sorted into each bin. The HMI and emergency stop button were mounted to quarter-inch acrylic with cutouts for both parts. The subassembly was then mounted to the front of the lighting enclosure. The HMI subassembly can be found in Appendix A along with a drawing with dimensions in Appendix B. Figure 4-10 shows the overall HMI Interface subassembly.

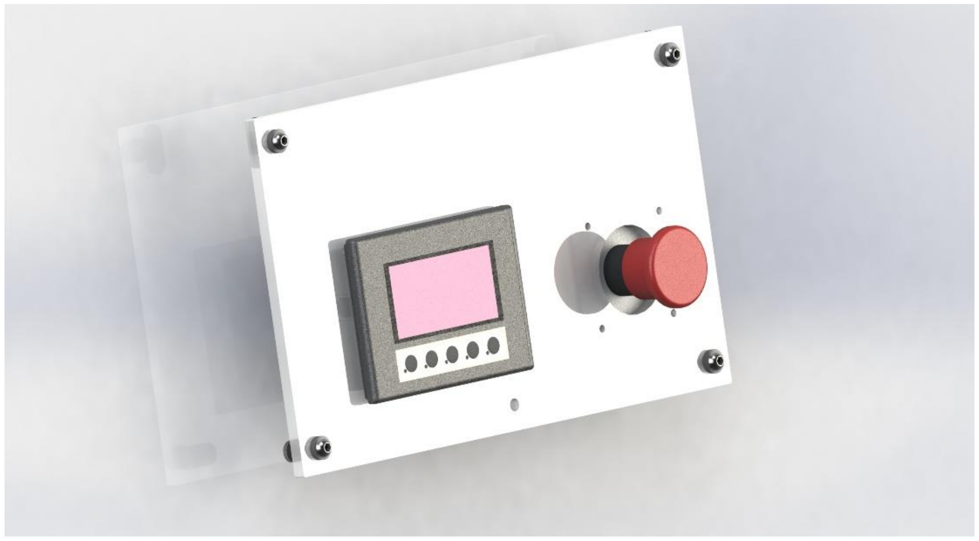


Figure 4-10: HMI Interface Subassembly

4.1.7.1 Overall Assembly

The overall system was assembled as described in the previous subsections. The overall assembly is found in Appendix A along with a drawing with dimensions of the overall system in Appendix B.

4.1.7.2 Electronics Wiring

Custom wiring harnesses were used throughout the electronics assembly. These were created by using both male and female crimp connectors and inserting them into single row blank header sockets. All wires used were 20-guage stranded to allow for the wire to bend easily between connections.

The fiber optics were cut to the desired length using the cutting tool supplied with the fiber optic transmitter and receivers. The optic cable was fed through the cutting tool and the blade pressed down on the cable to make a clean cut. The wires were then inserted into the fiber optic amplifiers. No consideration of reciever and transmitter was required when installing the optical cables.

Communication between the PLC and HMI was transmitted through an RJ12 networking cable. The RJ12 cable also supplied power to the HMI. During programming of the PLC, the HMI was disconnected from the PLC's port number one and reconnected after programming.

4.2 Non-Conforming Peg Modification

Non-conforming pegs were not given to Purdue University by Lafayette Instruments. Simulation of the shortened pegs was created by shortening pegs of each group by the use of a rotary tool with a metal cutting wheel. Ten pegs were shortened from each type of peg group. Two of each of the small and medium pegs were marked to be shortened at 0.9, 0.8, 0.7, 0.6, and 0.5 inches. The keyed pegs were marked to be cut, two of each, at 1.1, 1.0, 0.9, 0.8, 0.7 and 0.6 inches. The table below shows the pegs desired cut length versus the actual length cut with error coming from hand cutting the pegs.

Table 4-1: Theoretical Non-Conforming Peg Size and Cut Peg Size

Peg #	Small		Medium		Keyed	
	Theoretical (Inches)	Actual (Inches)	Theoretical (Inches)	Actual (Inches)	Theoretical (Inches)	Actual (Inches)
1	0.500	0.492	0.500	0.497	0.600	0.598
2	0.500	0.496	0.500	0.537	0.600	0.625
3	0.600	0.563	0.600	0.571	0.700	0.739
4	0.600	0.615	0.600	0.600	0.700	0.723
5	0.700	0.688	0.700	0.655	0.800	0.785
6	0.700	0.714	0.700	0.655	0.800	0.818
7	0.800	0.788	0.800	0.773	0.900	0.906
8	0.800	0.837	0.800	0.757	0.900	0.926
9	0.900	0.867	0.900	0.895	1.000	0.975
10	0.900	0.867	0.900	0.865	1.000	0.943

4.2.1 PLC Software

The PLC was used as the central hub for communication and control of the overall system. The PLC was responsible for the fiber optics, activating the camera and transferred communication between the Jetson TX1 and the Arduino. The connection to the HMI allowed for selection of peg quantities along with visual display of current container fill status to the operator. The PLC code was based on ladder logic which was a series of singular lines controlling the overall system. The PLC was programmed through a "Click Koyo" programming software (Click Programming Software Ver 2.20). Programming involved the use of an RJ-45 ethernet cable connected from the PLC port 1 to a serial to USB converter and into the PC. The ladder logic can be found in Appendix D.

The HMI provided a user interface for the operator. The HMI was coded through "C-more Micro Programming Software" and was sent directly into the HMI by RJ-45 cable. The purpose of the user interface was to allow the operator to select the number of pegs that were going into each bin and know the progress as the bins were filled. The user interface also allowed the operator to stop the system by turning on and off the hopper. The HMI screens and database are found in Appendix D.

4.3 Imaging System

The imaging system was designed around the Jetson TX1 for its capabilities with image processing, analysis and GPU speeds. The Jetson TX1 required an initial flash to install the correct operating system and software, Pylon5 and libraries needed to be installed and functional code needed be written for the image processing and analysis.

4.3.1 Jetson TX1 System Setup

The Jetson TX1 required installation of Ubuntu 16.04 and OpenCV 3.3.1 through use of the installation package Jetpack 3.2.1. The install guide located within the Jetpack 3.2.1 release notes was followed (NVIDIA Corporation, 2018).

To install the new operating system, OpenCV and dependencies, a Linux machine was required. Using a Surface Pro 4 as the host computer with Oracle VM VirtualBox 5.2.11 installed, a virtual machine running Ubuntu 16.04 LTS was created. An account was created with Nvidia via their website and Jetpack was installed onto the virtual machine. Through the terminal the program was then unpacked, given admin rights using the chmod command and ran. OpenCV was selected as an additional option during the flashing of the Jetson TX1 with the Ubuntu operating system.

Once installed, a USB cable was connected between the Surface Pro 4 and the Jetson TX1’s USB micro-B recovery port. The installation required that the Jetson TX1 was connected directly through Ethernet to the same network as the host computer. Once all connections were made, the Jetson TX1 was put in recovery mode by holding the reset and recovery push buttons simultaneously, then releasing the reset push button and three seconds later releasing the recovery push button as described in the release notes. The installation resumed once the host machine found the IP address of the Jetson TX1 and OpenCV along with dependencies were pushed to the Jetson TX1.

4.3.1.1 Jetson TX1 Libraries

Upon completion of the Jetpack program’s installation, the Jetson TX1 was restarted to ensure functionality. The OpenCV package was then unzipped and installed through the terminal window. This allowed the OpenCV libraries to now be used from the C++ program. The JetsonGPIO library by JetsonHacks was also used for easy access to the GPIO pins located on GPIO header J21 of the Jetson TX1 Development board. The JetsonGPIO library was downloaded and installed from the github repository (JetsonHacks, 2015).

The Pylon 5 API was installed from the Basler website (Basler AG, 2018). The Pylon 5 API comes with all the libraries and functions necessary to operate the Pulse camera using C++. The Pylon API proved difficult to run with C++ and Basler Support was contacted for assistance. Basler support was able to provide a sample file which is included in Appendix D (Basler Support, 2018). The Basler sample code actuated the Pulse camera taking an image and displaying it to the user. The sample code was expanded on and altered using the pylon libraries to match the necessary functions of the overall system.

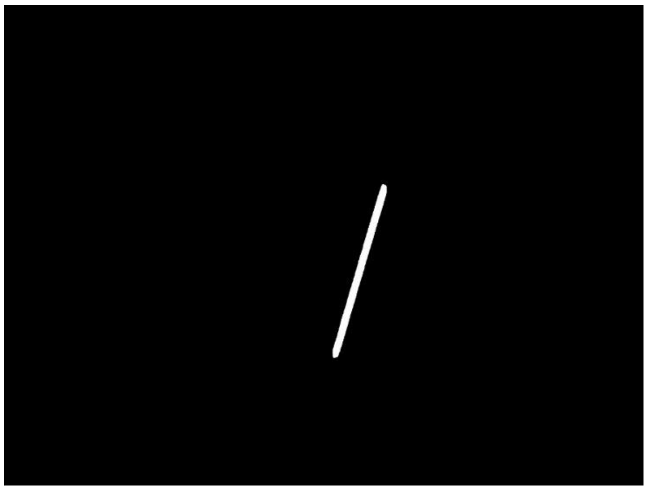
4.3.1.2 Jetson TX1 and Image Processing Code

The image processing code was developed from the flowchart in Figure 3-12. The OpenCV functions in this section were based on examples which were found in the OpenCV 2.4.13.7 documentation (OpenCV Dev Team). Development of the image processing system involved several iterations and checks along the way. In the initial system, the camera was initialized by creating an instant camera object using the Pylon 5 API function library. A pointer was created to grab the data from the frame buffer after actuation. The camera was then actuated by the PLC switching an output pin to a logic high. The Jetson GPIO library (JetsonHacks, 2015) was used to read the digital input and start the program. An image was acquired and saved using the imwrite() function. The saved image then went through a variety of filters and image processing algorithms. The OpenCV functions were used to convert the image to grayscale using the cvtColor() function. The image was passed through medianBlur() to get rid of random noise and then a threshold was applied to the overall image using the OpenCV function threshold(). The contours were found using the findContours() function. The function found the bodies or objects represented as white in the image and identified them as contours.

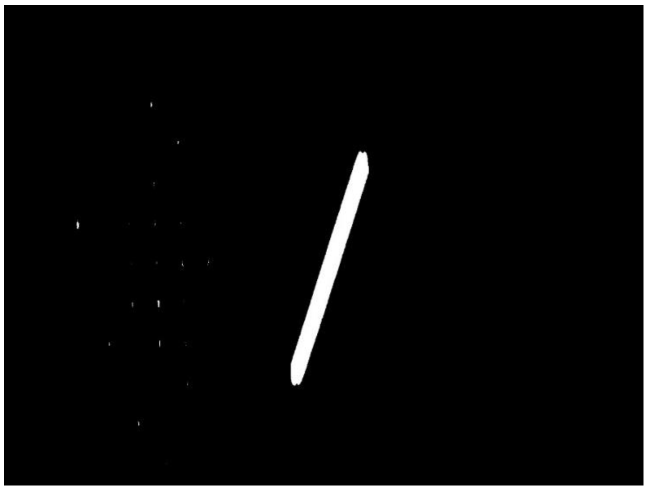
The contour data was passed to a function which finds the outlines of the contours and can approximate length and width in pixels. The initial method consisted of using the boundingRect() function to find height and width in pixels. The boundingRect() function was unable to account for the angle of the peg which made the height decrease and width increase proportionately to the angle of the peg within the ROI. Accounting for angle involved switching to the minAreaRect() function to analyze the contours and then using the RotatedRect class to pull the correct length and width from the contour data.

A conforming or non-conforming decision was based on the conversion of the length in pixels by use of a calibration equation. The calibration equations used were specific to each peg type. The calibration equation was extracted from the calibration data and the number of pixels was converted to inches. Formulation of the calibration equations is described in later sections and extracted from the graphs in Figure 4-12. The length dimension was compared to a stored tolerance for that individual peg type and a decision was made. The decision was transferred to the PLC by sending a GPIO pin to a digital low. The decision was passed to the Arduino for control the selection servo. The pseudo code along with the C++ code for the keyed pegs is located in Appendix D.

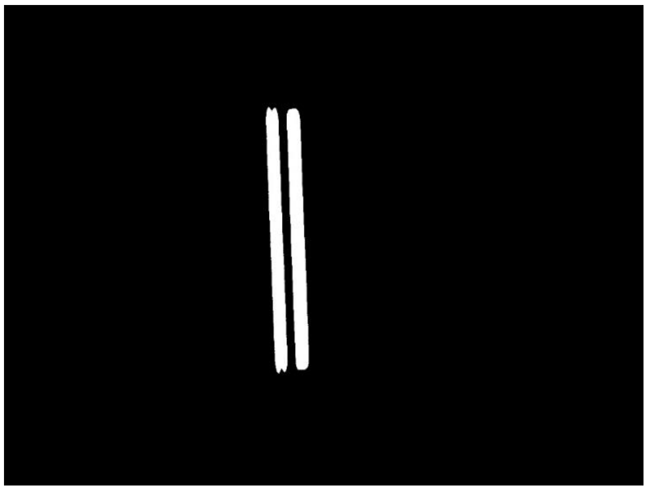
The code was replicated for each peg type changing the camera calibration equation. The keyed peg code was altered for the number of contours allowed in each image before rejecting the pegs. Small and medium pegs were rejected if more than one peg was detected. A shadow along the key side of the keyed peg created two different contours for a single peg. Keyed pegs were rejected if three peg sized contours were found. The largest contour found in the keyed peg image was used in calculating the overall length. Images of each peg type are shown in Figure 4-11 post segmentation.



(a) Segmentation of Small Peg



(b) Segmentation of Medium



(c) Segmentation of Keyed Peg

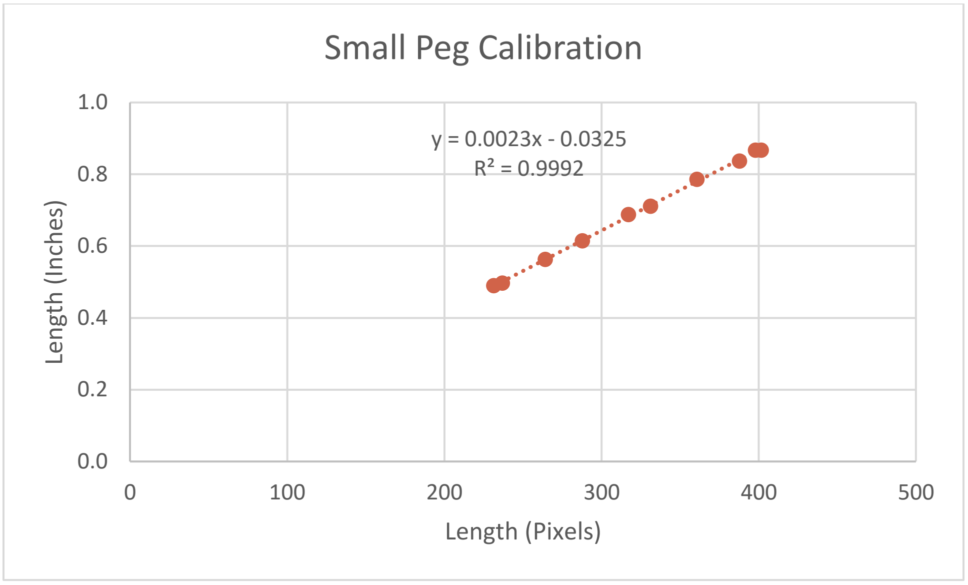
Figure 4-11: Segmentation Image of Each Peg Type

4.3.1.3 <u>Camera Calibration</u>

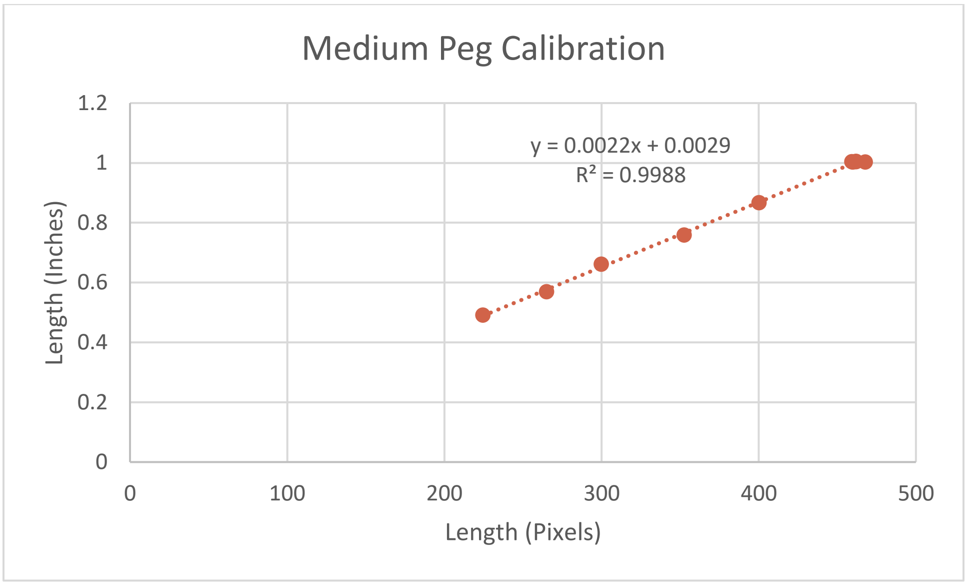
The camera calibration was conducted for each of the three peg types. Individual calibrations were necessary to account for the diameter which affects the distance from the top of the peg to the camera. A picture was taken of each peg placed directly beneath the camera and parallel with the conveyor. During testing with a moving conveyor small blur was observed in images and the calibration was performed again with the pegs on the moving conveyor. The length of the peg in pixels using a moving peg was outputted from the program and written into a table in excel. The pixel count was then compared to the actual length of the pegs using a scatter plot. The trendline created from the scatterplot gives the equation of length=0.0023*pixels-0.0325. The equation was applied to all new peg images to determine the length of incoming pegs in inches. The following procedures were applied to each peg type. The calibration equation for the medium pegs was length=0.0022*pixels–0.0029 and the calibration equation for the keyed pegs was length=0.0023*pixels-0.0447. Table 4-2 displays the pegs size versus the pixel count. Figure 4-12 displays the three peg type graphs with trendline for each calibration. The trendline plots were created using Microsoft Excel ("Office 365. Excel", 2016).

Table 4-2: Calibration Table of Measured Peg Dimension to Pixel Length

Peg #	Small		Medium		Keyed	
	Measured (inches)	Pixel Count	Measured (inches)	Pixel Count	Measured (inches)	Pixel Count
1	0.497	236.950	0.492	224.557	0.603	285.925
2	0.490	231.409	0.570	264.948	0.720	336.021
3	0.563	264.189	0.661	299.812	0.789	368.490
4	0.615	287.696	0.759	352.460	0.908	418.863
5	0.688	317.110	0.867	400.176	0.945	433.221
6	0.711	331.232	1.004	459.212	1.119	512.387
7	0.786	360.673	1.003	459.874	1.115	517.404
8	0.837	387.685	1.005	461.780	1.118	508.820
9	0.867	397.890	1.004	461.790	1.120	510.189
10	0.867	401.636	1.003	467.850	1.121	512.876

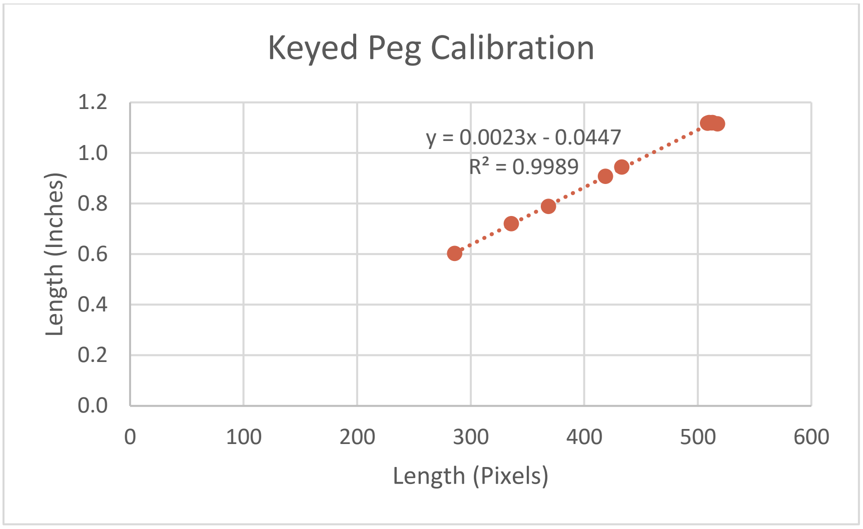


(a) Vision System Calibration Trendline



(b) Vision System Calibration Trendline

Figure 4-12: Graph of Trendlines for Calibration Equations of Three Peg Types



(c) Vision System Calibration Trendline

Figure 4-13 Continued: Graph of Trendlines for Calibration Equations of Three Peg Types

4.4 Arduino Code

Programming of the Jetson TX1 for pulse width modulation (PWM) outputs was tedious and required deeper understanding of the Jetson environment. A solution was developed for simplicity and time restrictions to enable PWM functionality controlled by the Jetson TX1. An Arduino Mega 2560 was added to the design with communication through the GPIO headers to the Jetson TX1.

The Arduino was responsible for controlling the servo to decide if a peg entered a conforming or a non-conforming bin. The Arduino was also responsible for turning the bin carousel when each bin was filled to the predetermined quantity. The Arduino code contained a series of if statements based on an input voltage reading. An interrupt activated on the rising and falling edges of that digital pin began the processing of the if statements. The decision on peg conformality was passed by the PLC. Present peg quantity in the carousel bin was tracked by the PLC and the decision to rotate to the next bin was passed to the Arduino in order to control the stepper motor. The PLC communication was based on grounding digital pins D4 and D5 of the Arduino which were attached to pullup resistors through software. The Arduino pseudo code is found in Appendix D along with the Arduino IDE code.

4.5 Mechanical Post Build Revisions

The following section discusses the mechanical revisions after full assembly of the sorting system. These revisions were made based on developmental testing of each section of the overall system.

4.5.1 Revised Reject Collection Method and Reject Bin

Developmental testing exposed an issue in counting the correct number of pegs in the conforming bins. The fiber optics located at the end of the conveyor were responsible for detecting when a peg fell into a carousel bin or the reject bin. To count correctly, the PLC had to know the decision of the camera system and be able to correlate the decision to which peg was currently falling into the reject bin. To decrease the probability of counting a non-conforming peg as conforming, a mechanical revision was made to ensure that no non-conforming pegs would be able to pass through the fiber optics.

The reject bin was removed from the drop-off area at the end of the conveyor. The reject bin was then redesigned to be fixed to the side of the conveyor. A reject slide was cut from quarter-inch acrylic and mounted to the aluminum frame. The reject slide would guide the non-conforming pegs into the revised reject bin. The additional space necessary for the reject slide was accounted for by attaching the servo support plate to the camera enclosure. This also minimized the distance between the camera system and the flipper. A smaller distance eliminates the requirement for a delay of the camera system waiting to communicate to the servo to rotate the flipper. Figure 4-13 displays the revised reject bin and additional reject slide.

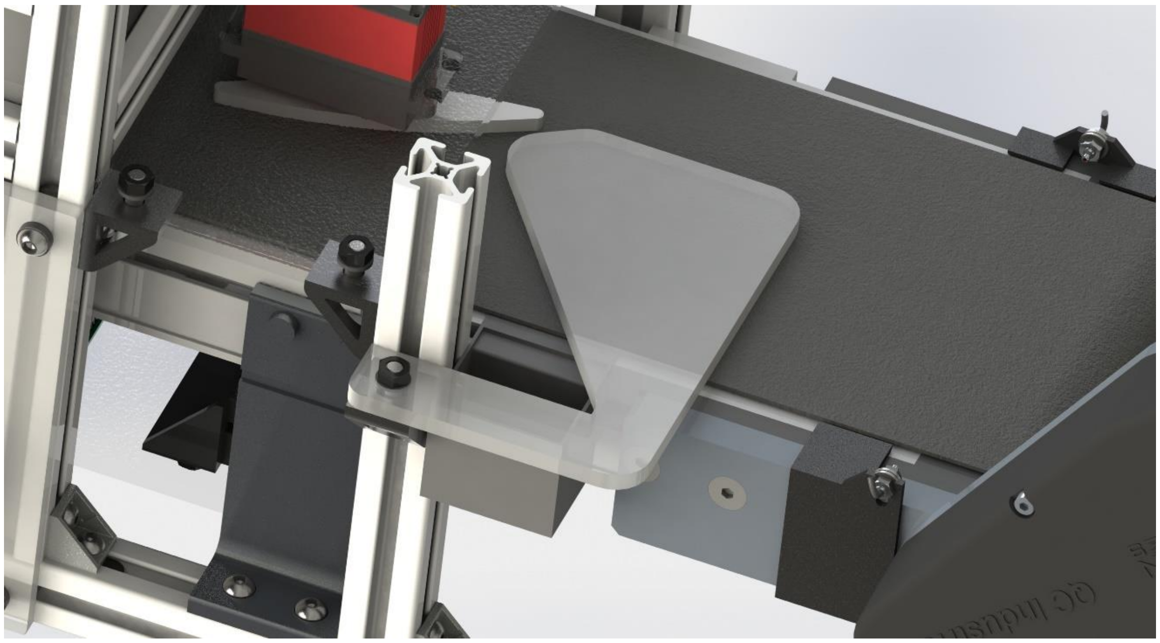


Figure 4-14: Revised Reject Bin and Reject Slide

4.5.2 Revised Bin Carousel

Pegs falling from the end of the conveyor during developmental testing were observed missing the conforming bins and falling out of the system. Pegs bouncing out of the bins was also observed during system development. A revision of the bin carousel was made to solve both issues. The revised carousel bins were created which were wider to accommodate movement of the peg during transportation. The carousel plate, which confines the containers, was redesigned to accommodate the larger size of the new carousel bins. The revised bin carousel assembly is shown in Figure 4-14.

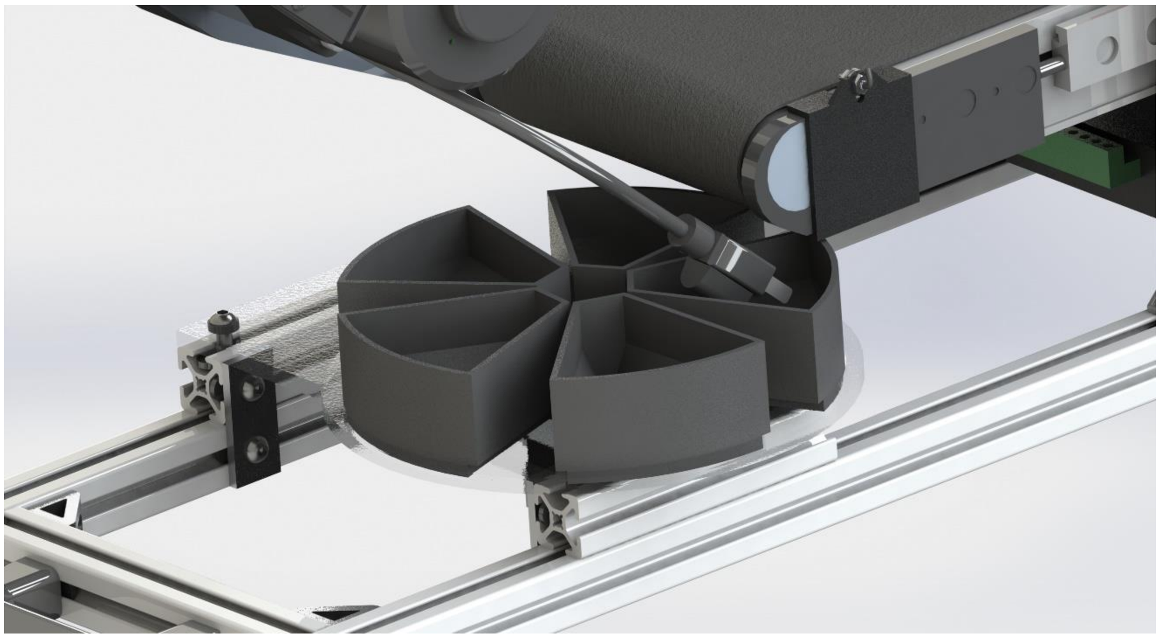


Figure 4-15: Revised Bin Carousel

4.5.3 Revised Hopper Separator Plate

Testing of the hopper assembly with the small peg separator often led to the release of two bundled pegs at one time onto the conveyor belt. Although the camera system was able to calculate the height of both pegs simultaneously, the fiber optics were not capable of counting two pegs when they fell into the carousel bins. A temporary solution of rejecting both pegs in order to ensure correct counting was initially implemented. A revision to the hopper was necessary to fix the root cause of bundled pegs on the conveyor. Two pegs were able to seat together into a single slot of the separator plate. Thickness of the separator plate was the root cause. A thinner acrylic plate would have been fragile under the weight of hundreds of pegs. Lafayette Instruments supplied 31mil sheet metal for separator plate fabrication. The metal was cut with a waterjet with the same design as the acrylic separator plate. Testing showed the 31mil sheet to be the correct thickness for allowing only one peg to release on the conveyor at a time (Echard, 2018) A new issue arose when pegs were stuck beneath the metal separator plate. Warp in the sheet metal created voids between the agitator and the separator plate allowed pegs to slip between the two. J-B Weld Qwikweld metal bonding epoxy was used in order to adhere the acrylic agitator into the cuts of the separator plate creating a flush and strong connection. The combined piece allowed the pegs to be release one at a time without pegs slipping underneath the plate. Figure 4-15 shows the agitator and separator plate assembly.

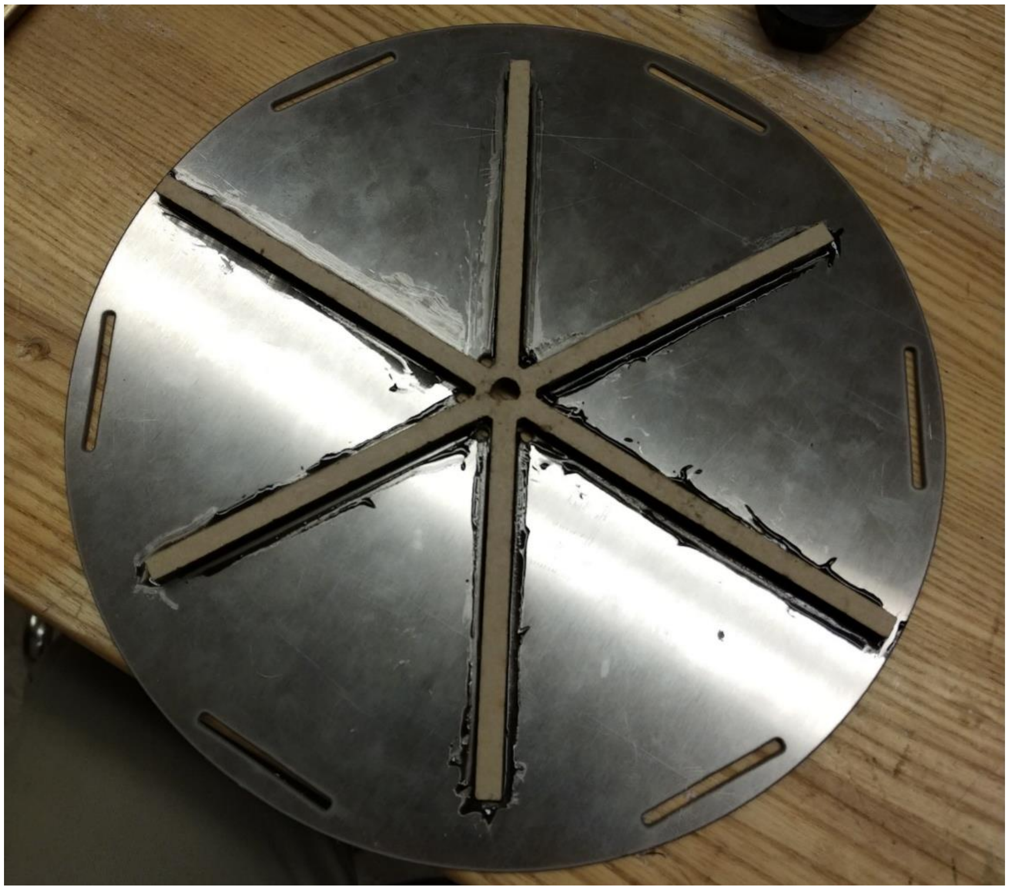


Figure 4-16: Metal Separator Plate with Agitator

A second agitator and separator plate were to be used for the larger peg types. The separator plate for the medium and keyed peg type was cut from 62mil acrylic. The agitator was then adhered to the separator plate. The separator plate and agitator are shown in Figure 4-16.

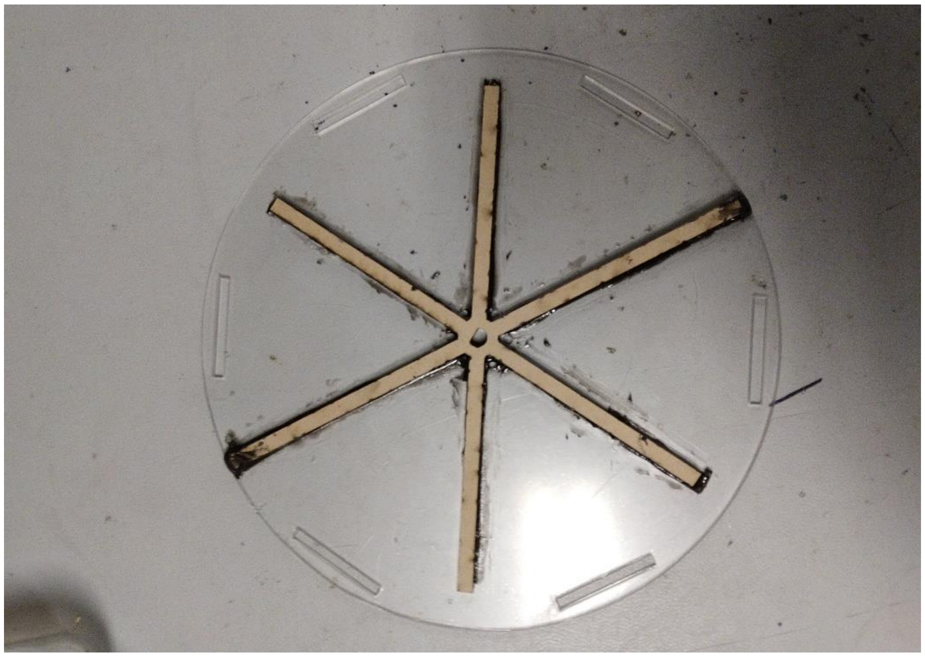


Figure 4-17: Medium and Keyed Separator Plate

4.5.4 Revised Flipper

Testing of the small pegs had no issue sliding the peg the correct direction using the flipper. Medium and keyed pegs that were slightly rotated often got stuck against the side of the flipper for several seconds before continuing to the carousal bins. A new flipper was designed and cut from quarter-inch acrylic which decreased the possibility of the pegs getting stuck against the flipper. The new flipper design is shown in Figure 4-17.

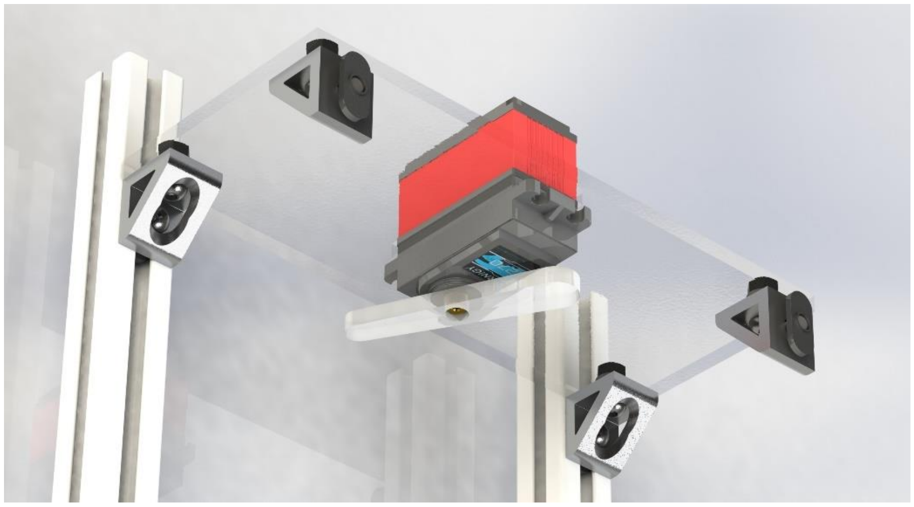


Figure 4-18: Revised Flipper for Selection Servo

4.6 Test Results and Analysis

The following section discusses the testing results of each peg type through the system. The results were then analyzed, and a statistical analysis was given to evaluate the overall accuracy and reliability of the sorting system.

4.6.1 Testing Procedure

Testing was performed in a modular fashion. The results were broken down into the performance of each individual section of the system and then an overall performance measurement was deduced from the results.

The pegs were hand fed into the hopper to ensure that they were caught by the rotating slots. The clearance between the slot and the hopper wall allowed pegs to sit without being captured and carried to the top of the hopper. A single conforming peg was first tested through the system to allow the auto exposure function to calibrate without affecting the test results.

The performance of the individual sections was broken down as follows. The hopper was evaluated based its ability to rotate a peg to the top and successfully drop the peg onto the conveyor without it bouncing off. The conveyor is evaluated by its ability to transport the peg from the hopper to the carousel without the peg falling or rolling from the predetermined path. The camera system was evaluated for its ability to make the correct decision for the pegs, conforming or non-conforming. The selection servo was assessed by its ability to properly respond to the camera systems decision and its ability to position the peg the correct direction along the conveyor. The reject and carousel bins were assessed by successful peg deposition into the containers. Correct rotation of the bin carousal was noted based on successfully rotating after five pegs were collected in each bin. Rotation should occur twice per test.

The test was performed with a conveyor speed of 16.84 feet per second and with a hopper rotation speed of 7.58 revolutions per minute. These measurements were visually calculated with a stop watch. Measurements of conveyor speed and hopper rotation speed were each taken three times and then averaged.

The test for each peg type consisted of a total of twenty pegs being inserted into the system. First, five conforming pegs were inserting into the hopper slots to be sorted by the system. Next, five of the non-conforming pegs were randomly selected and inserted into the hopper slots. The methodology was repeated with five more conforming pegs and then the five final non-conforming pegs. Each peg was measured prior to entry into the hopper and recorded. The test was repeated five times for each peg type. The test results for each peg type are provided in Appendix E.

4.6.2 Vision System Statistical Analysis Methods

Statistical analysis was performed on the vision system using Microsoft Excel ("Office 365. Excel", 2016). Linear regression models were created for each type of peg based on all data points representing peg length collected from the vision system. The R^2 value was a critical factor in analyzing how well a system would predict future instances. An R^2 score of one means that the system was a perfect predictor (Collins-Thompson, 2018).

The pegs were also analyzed based on mean and standard deviation as obtained from the vision system peg data. The arithmetic mean of the error was used. A mean evaluates the accuracy of the model. The standard deviation represents the spread of values around the mean (Gravetter & Wallnau, 1991). A smaller standard deviation value represented a higher precision system.

Finally, a paired difference t-test was performed for the vision system. The paired difference t-test evaluated the null hypothesis as no significant difference between the predicted and actual length of the pegs. The alternative hypothesis stated there was a significant difference between predicted and actual peg length. A t-statistic was calculated for a paired differential t-test. The t-statistic was determined using equation 4.1 (Gravetter & Wallnau, 1991).

$$t = \frac{\bar{D} - \mu_D}{s_{\bar{D}}} \tag{4.1}$$

The calculated t-statistic was compared to the t-statistic of the confidence interval to accept or reject the null hypothesis.

4.6.3 Small Peg Results

The small pegs were tested first. The combined results from the five tests are provided in Table 4-3. The system was successful in sorting of the 100 total pegs. An anomaly occurred in test four when a non-conforming peg was analyzed incorrectly. The peg was analyzed as 0.37 inches shorter in length than the actual value. The peg was a non-conforming peg and was still analyzed as a non-conforming peg. The anomaly had no impact on the successful sorting results.

Table 4-3: Test Results of Different Components of System with Small Pegs

Pegs #	Hopper	Conveyor	Camera	Selection Servo	Container	Rotation
20	20	20	20	20	20	2
20	20	20	20	20	20	2
20	20	20	20	20	20	2
20	20	20	19	20	20	2
20	20	20	20	20	20	2

A linear regression model is given in Figure 4-18 showing the ability of the model to predict future peg lengths. A R^2 value of 0.9651 was found using the length of the 100 pegs. The R^2 value was close to the value of one which showed a very high prediction accuracy of the camera system. The outlier affected the overall results and the R^2 value. The R^2 value after removal of the outlier is 0.9991 which shows that the camera system was a good predicter of peg length.

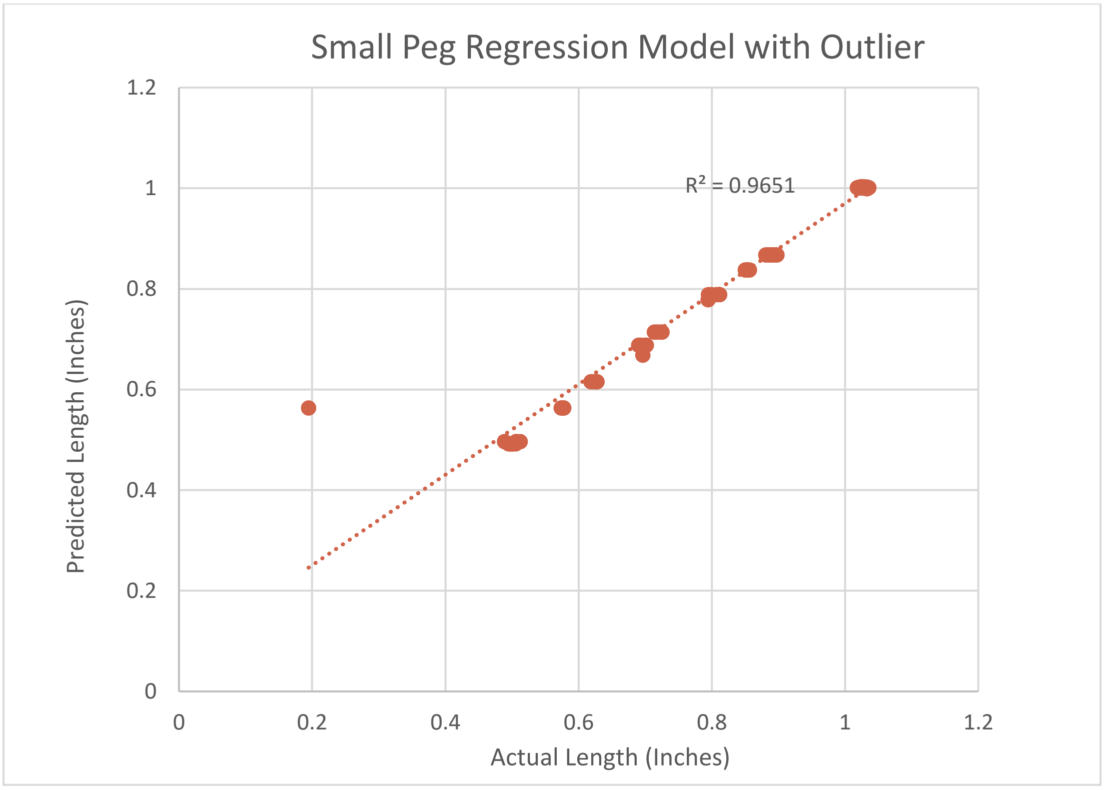


Figure 4-19: Small Peg Linear Regression Model

The mean value (average) for the difference of the predicted and measured peg length was found to be 0.0161 inches. The standard deviation was found to be 0.040 inches. The mean displayed an offset of the pegs predicted value from the actual value. The mean plus the standard deviation were within the ±0.05 inch tolerance of the system. The low standard deviation shows that the camera system was very consistent in measurements.

A paired difference t-test was performed at a 95% confidence level to evaluate the camera systems performance. The null hypothesis was defined as H_0: $\mu = 0$. The alternative hypothesis was expressed as H_a: $\mu \neq 0$. The t-statistic was calculated (equation 4.1) and was found to be 4.03. The t-statistic was also found using a t-distribution table which was 1.984. The null hypothesis was rejected and the alternative hypothesis that there was a difference between the predicted and actual length of the peg was accepted. The null hypothesis was rejected, however, all conforming pegs predicted test values still fell within the tolerances of a conforming peg.

4.6.4 Medium Peg Test Results

The medium pegs were tested next. The overall test results are displayed in Table 4-4. The system was 97% successful in the correct sorting of the 100 pegs. A non-conforming peg became wedged between the two positioning plates on the conveyor prior to entry into the lighting enclosure. The peg was manually centered between the plates to not affect the next pegs. A separate peg did not make it through the camera system as it bounced from the conveyor when landing from the hopper. A final peg actuated the vision system but was analyzed improperly resulting in a value of zero inches. This peg was moved to the reject container.

Table 4-4: Test Results of Different Components of System with Medium Pegs

Pegs #	Hopper	Conveyor	Camera	Selection Servo	Container	Rotation
20	20	20	19	20	20	2
20	20	19	20	20	20	2
20	20	20	20	20	20	2
20	20	20	20	20	20	2
20	20	19	20	20	20	2

A linear regression model is given in Figure 4-19. A R^2 value of 0.9987 was found using the length of the 98 pegs which went through the vision system. The R^2 value was close to the value of one which represented a very high prediction accuracy of the vision system.

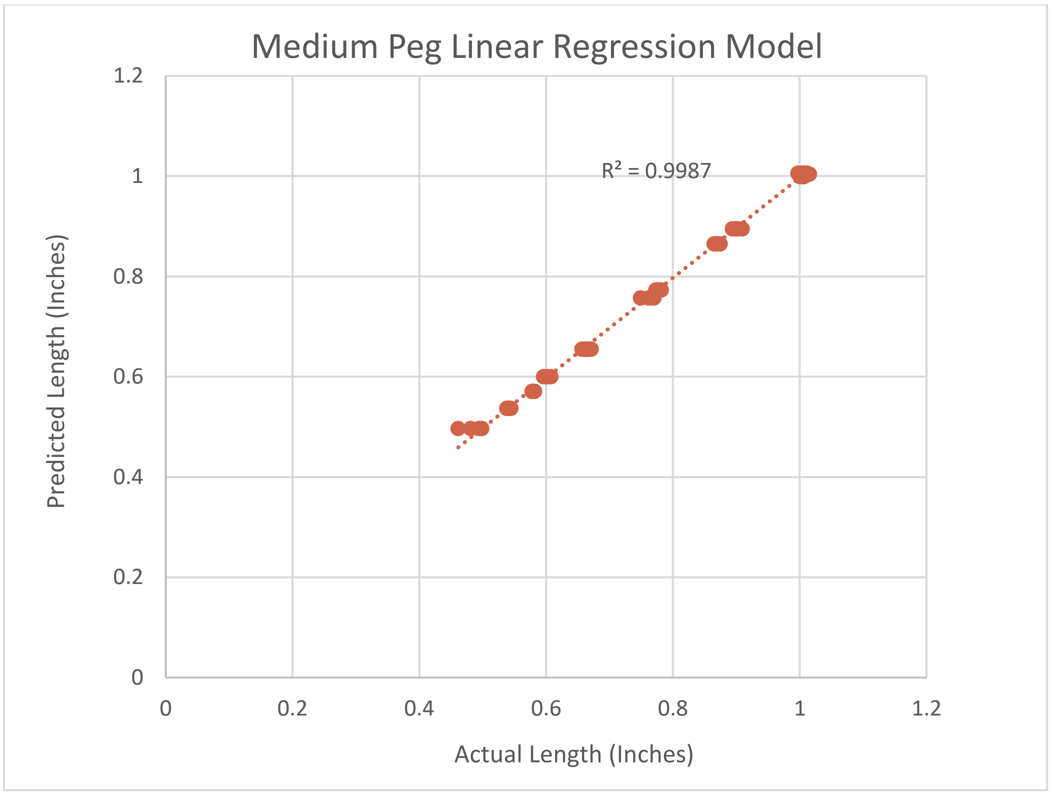


Figure 4-20: Medium Peg Regression Model

The mean value (average) for the difference of predicted and actual peg length was calculated as 0.0029 inches. The standard deviation was found to be 0.0066 inches. The mean was near zero representing a high accuracy of the vision system while the standard deviation was small, meaning the error between predicted and actual length was consistently small.

A paired difference t-test was performed at a 95% confidence interval. The null hypothesis was defined as H_0: $\mu = 0$. The alternative hypothesis was defined as H_a: $\mu \neq 0$. The t statistic was calculated (equation 4.1) and was found to be 4.4. Looking up the t-statistic using a t-distribution table gave a value of 1.984. The null hypothesis was rejected and the alternative hypothesis that there was a difference between the predicted and actual length of the peg was accepted. The null hypothesis was rejected, however, all conforming pegs predicted test values fell within the tolerances of a conforming peg and no issues occurred in testing due to the predicted peg dimensions being out of the conforming range.

4.6.5 Keyed Peg Test Results

The keyed pegs were tested next. The overall test results are displayed below in Table 4-3. The system was 99% successful in the sorting of the 100 pegs. A non-conforming peg became wedged between the two positioning plates on the conveyor during test four. The peg forced the next peg out of order and this was noted as to not affect the statistical analysis or the mean difference.

Table 4-5: Test Results of Different Components of System with Keyed Pegs

Pegs #	Hopper	Conveyor	Camera	Selection Servo	Container	Rotation
20	20	20	20	20	20	2
20	20	20	20	20	20	2
20	20	20	20	20	20	2
20	20	19	20	20	20	2
20	20	20	20	20	20	2

A linear regression model is given in Figure 4-20. A R^2 value of 0.9964 was found using the length of the 100 pegs which went through the vision system. The R^2 value was close to the value of one which represented a very high prediction accuracy of the vision system. Small and symmetrical deviation was found along the regression line.

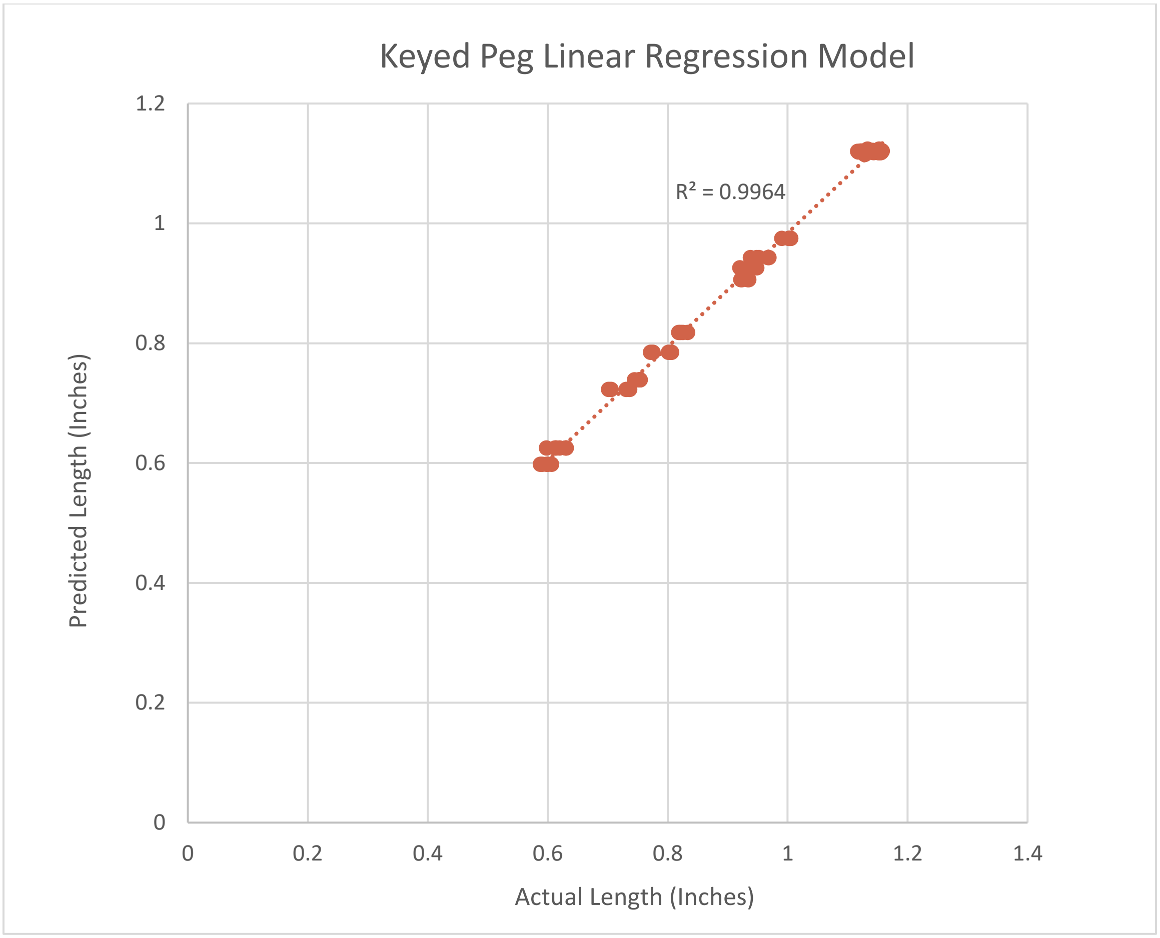


Figure 4-21: Keyed Peg Regression Model

The mean value (average) for the difference of predicted and actual peg length was calculated as 0.0151 inches. The standard deviation was found to be 0.0145 inches. The mean was near zero representing a high accuracy of the vision system while the standard deviation was very small, meaning the error between predicted and actual is consistently low.

A paired difference t-test was performed at a 95% confidence interval. The null hypothesis was defined as H_0: $\mu = 0$. The alternative hypothesis was defined as H_a: $\mu \neq 0$. The t statistic was calculated (equation 4.1) and was found to be 10.44. Looking up the t statistic using a t-distribution table was 1.984. The null hypothesis was rejected and the alternative hypothesis that there was a difference between the predicted and actual length of the peg was accepted. The t-value was higher in this test than desired, however, the error had no effect on the camera systems ability to correctly identify the pegs during testing.

4.7 Post Build and Testing Conclusion

The post build revisions were necessary to ensure proper functionality and accuracy of the sorting system. The revised reject bin placement and reject slide ensured proper counting of conforming pegs. The revised bin carousal and carousal bins ensured pegs were properly captured from the conveyor. Finally, the new separator plates with adhered agitators ensured that pegs fell individually onto the conveyor. The overall revised system is shown in Figure 4-19.

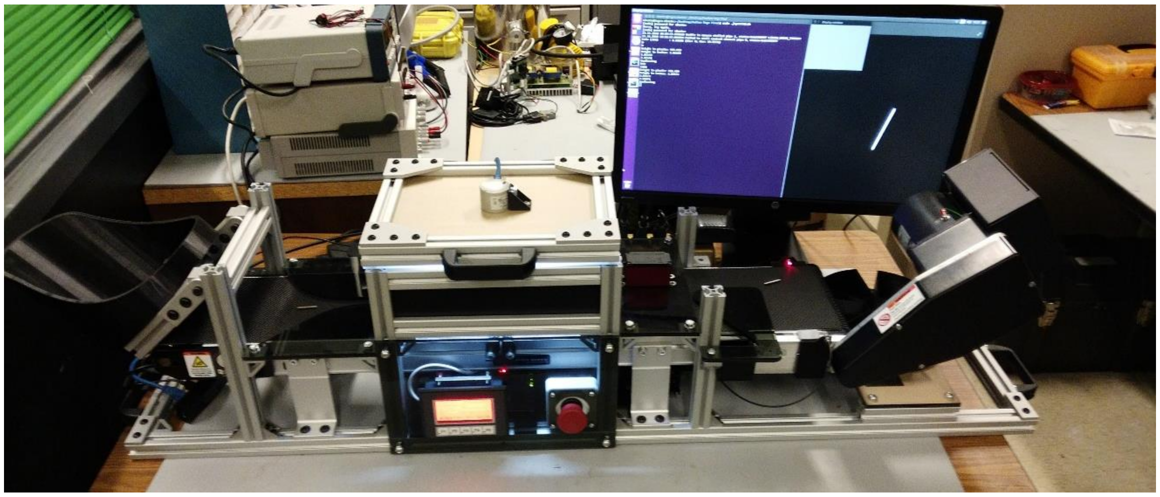


Figure 4-22: Assembled System with Post Build Revisions

Testing of the three peg types showed a 99% full system functionality for the small pegs, 97% for the medium pegs and 99% for the keyed pegs. All three tests met the requirement for 95% reliability. The testing also showed that no pegs entered the conforming bins to manipulate the final peg count per carousal bin.

The accuracies for the prediction of each peg type are broken down in Table 4-6. The results concluded average accuracies above 95% for all peg types. The average accuracy was 97.71% for small pegs, 99.27% for medium pegs and 98.15% for keyed pegs.

Table 4-6: Prediction Accuracy of Individual Peg Types

	Small Peg	Medium Peg	Keyed Peg
Average (%)	97.71	99.27	98.15
Minimum (%)	95.74	92.85	95.70
Maximum (%)	99.92	100.00	99.96

CHAPTER 5. SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

The following section describes recommendations and improvements for increasing the overall reliability and accuracy or the sorting system. These improvements are deduced from the testing results and observations.

5.1 Mechanical Recommendations

The hopper is currently driven by a DC gearmotor and a DC to DC converter to change its speed. Relying on voltage to control the motor speed allows the hopper speed to be affected by the weight of the pegs. A future recommendation is to replace the gearmotor with a motor which operates with a feedback loop. This would allow for the hopper to not be affected when a large number of pegs is inserted at a single time.

Testing proved the issue of non-conforming pegs being able to get stuck between the positioner plates. New plates should be designed to not allow a gap small enough for these pegs to get stuck between. Another solution would be to offset the plates pushing pegs all the way to one side of the conveyor before they run into the next plate. The next plate would then center the peg prior to entering the lighting enclosure.

Calibration of the fiber optics was very sensitive prior to testing. Caution was used to ensure the pegs were caught without catching the dimples of the conveyor as well. The falloff optic mount opposite did not slide correctly over the conveyor and was held on by tape during testing. This part should be redesigned to latch securely to the conveyor side. Both optic mounts were to slide tightly onto the conveyor sides. The PLA material was susceptible to warping over time and the optic sensor had to be calibrated before testing. Another material such as aluminum or ABS plastic may overcome to potential warping of the falloff optic mounts.

Finally, although an emergency stop button is present in the system as shown in Figure 5-1, it is currently nonfunctional. A recommendation would be to wire the emergency stop button to the PLC to allow full system shutdown if a problem occurs. The PLC code should be altered to accommodate the additional emergency stop button.

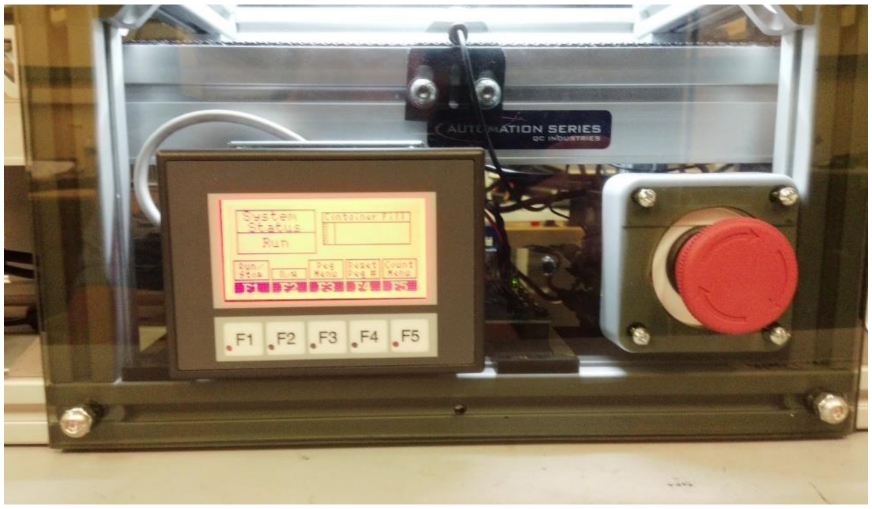


Figure 5-1: HMI and Emergency Stop Button

5.2 Software Recommendations

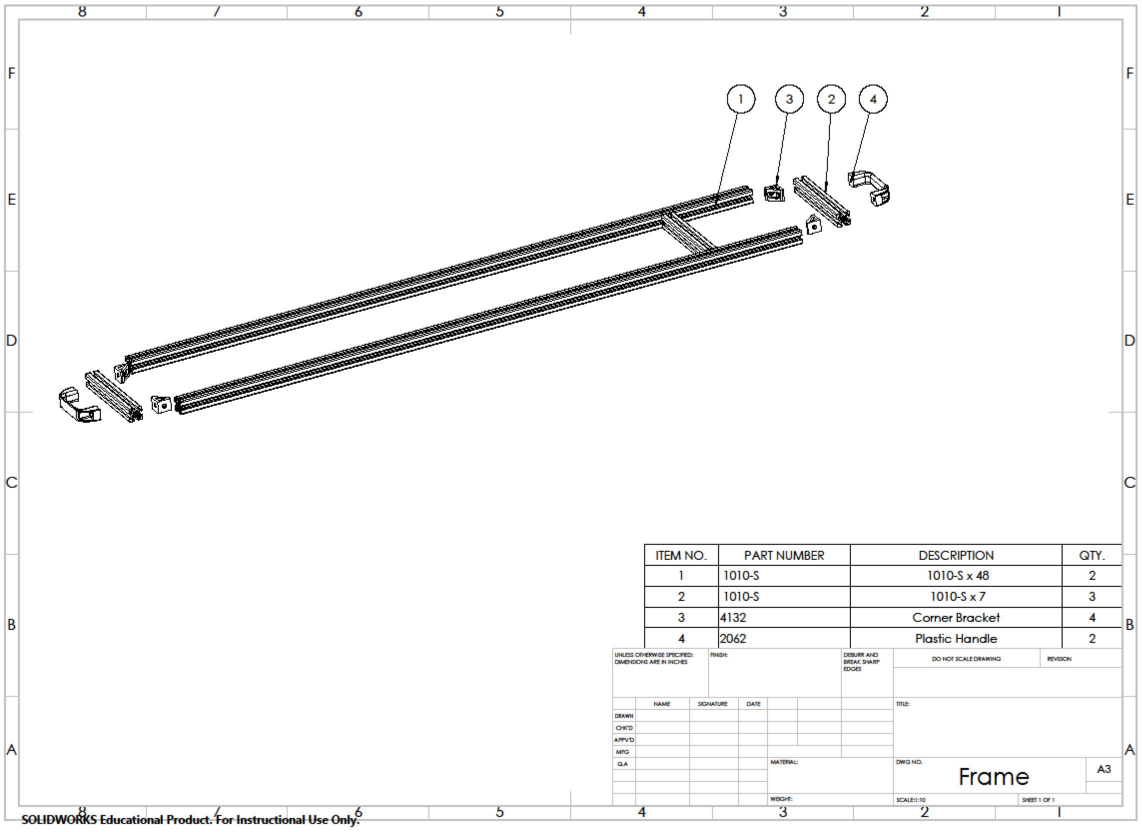
Currently the HMI is used to turn on and off the hopper as well as allow peg count selection with communication to the PLC. A recommendation for future improvement of the user interface would be to make the peg type selection menu fully functional and communicate with the Jetson TX1 to determine which program needs to be ran. The PLC would be used for the communication side and could pull one of several GPIO pins low on the Jetson to identify which peg type is being sorted. A single Jetson Code file could then be used for all peg types.

5.3 Summary

The automated sorting system for pegs using computer vision techniques was designed, constructed and tested in this study. The system results verified reliability of the small pegs at 99%, medium pegs at 97% and keyed pegs at 99%. The overall system was portable, inexpensive and the results show that the sorting system met the 95% reliability requirement.

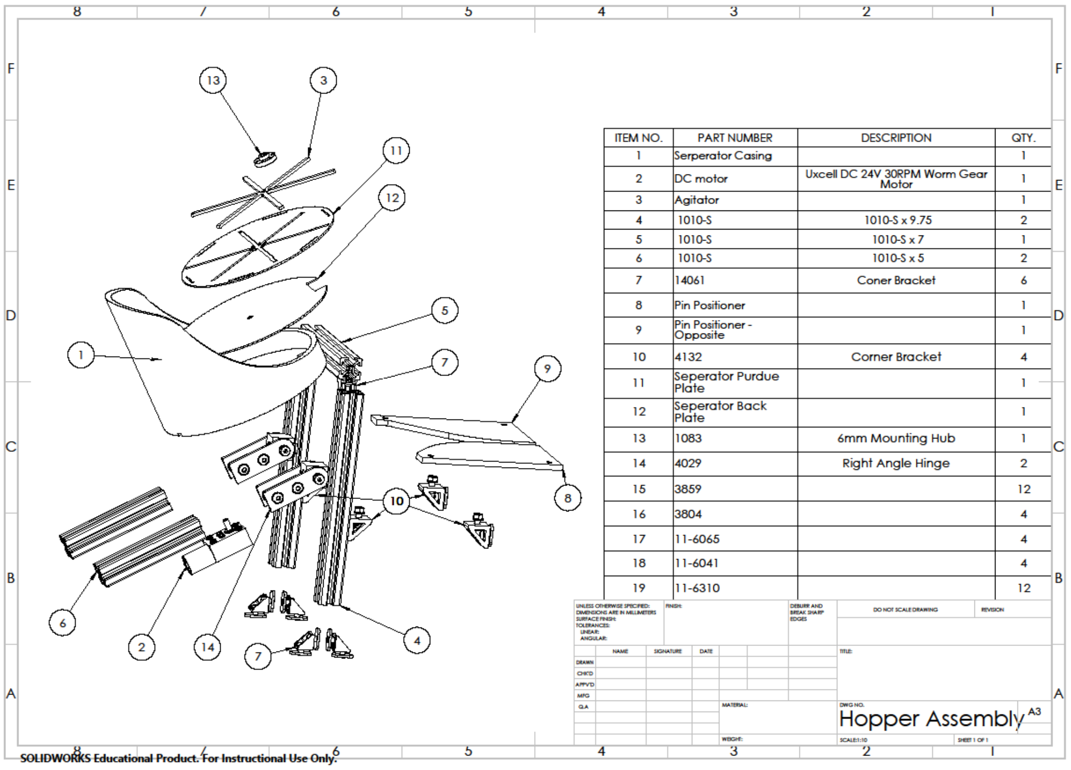
APPENDIX A. SUBASSEMBLY EXPLODED DRAWINGS

A.1 Frame Exploded Subassembly



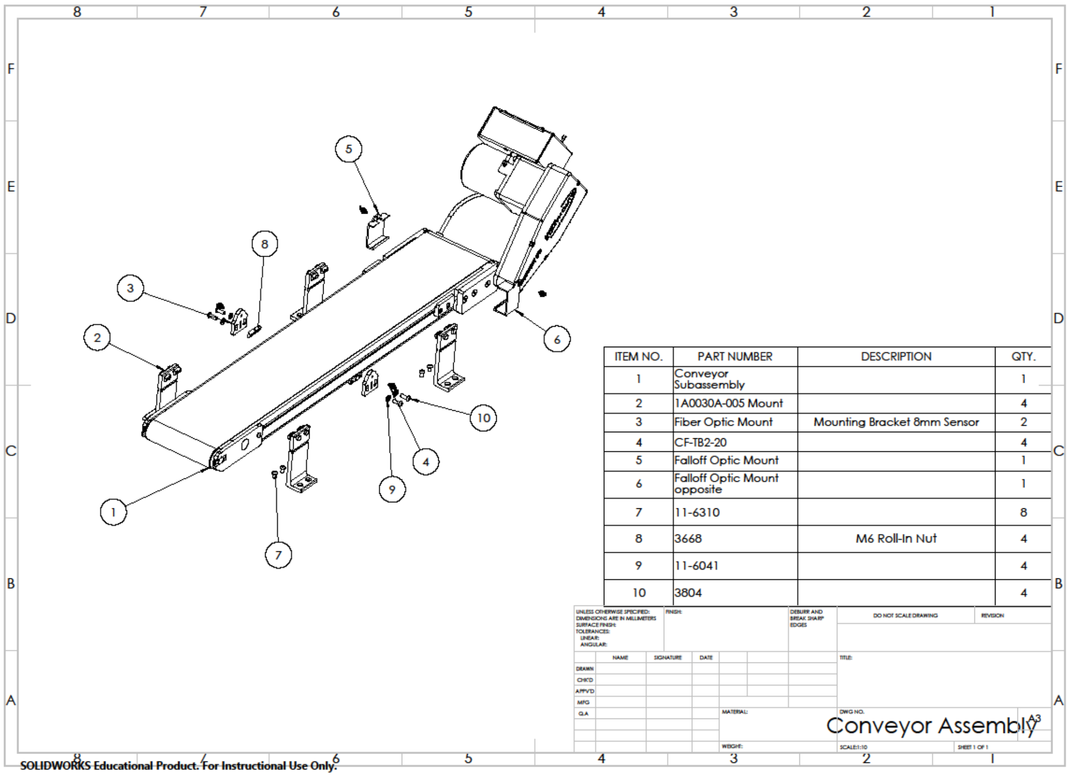
ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	1010-S	1010-S x 48	2
2	1010-S	1010-S x 7	3
3	4132	Corner Bracket	4
4	2062	Plastic Handle	2

A.2 Hopper Exploded Subassembly



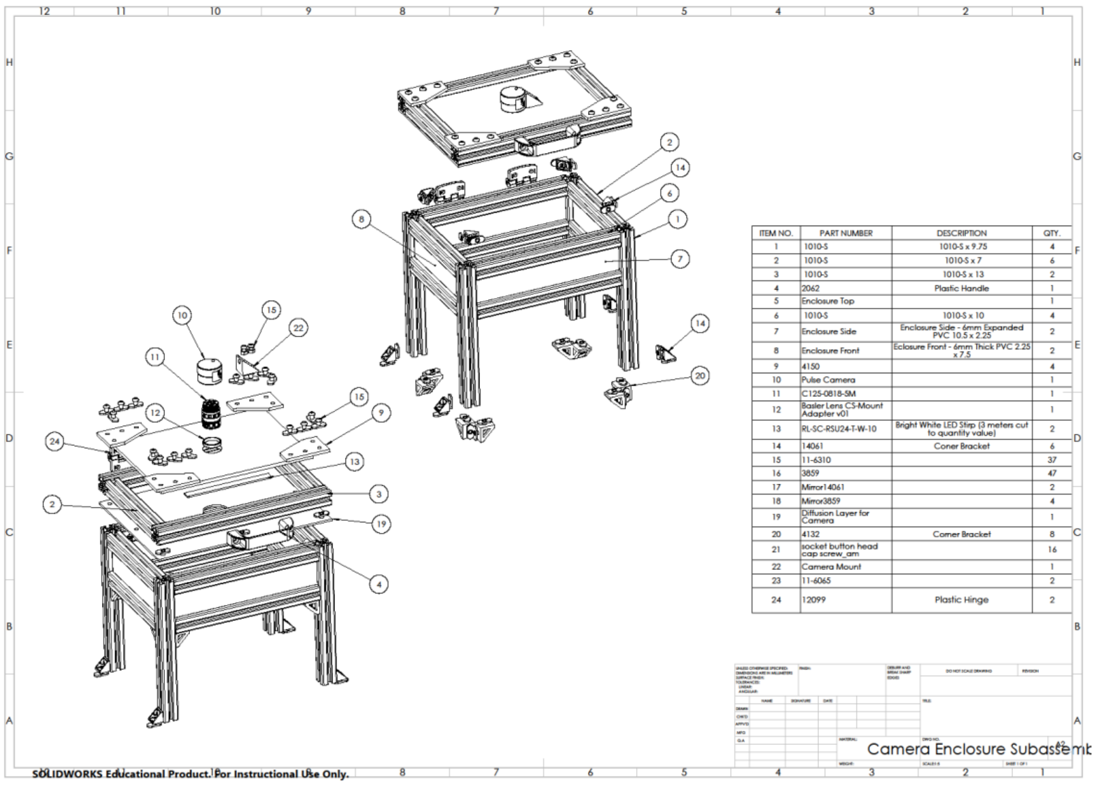
ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Serperator Casing		1
2	DC motor	Uxcell DC 24V 30RPM Worm Gear Motor	1
3	Agitator		1
4	1010-S	1010-S x 9.75	2
5	1010-S	1010-S x 7	1
6	1010-S	1010-S x 5	2
7	14061	Coner Bracket	6
8	Pin Positioner		1
9	Pin Positioner - Opposite		1
10	4132	Corner Bracket	4
11	Seperator Purdue Plate		1
12	Seperator Back Plate		1
13	1083	6mm Mounting Hub	1
14	4029	Right Angle Hinge	2
15	3859		12
16	3804		4
17	11-6065		4
18	11-6041		4
19	11-6310		12

A.3 Conveyor Exploded Subassembly



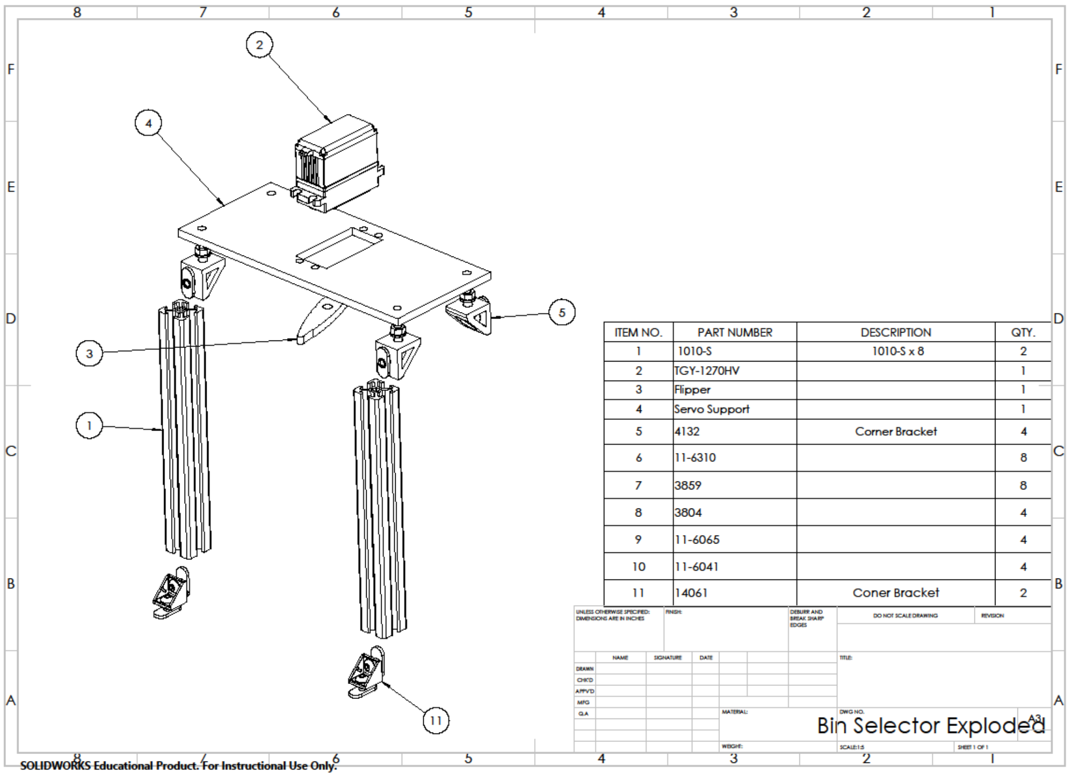
ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Conveyor Subassembly		1
2	1A0030A-005 Mount		4
3	Fiber Optic Mount	Mounting Bracket 8mm Sensor	2
4	CF-TB2-20		4
5	Falloff Optic Mount		1
6	Falloff Optic Mount opposite		1
7	11-6310		8
8	3668	M6 Roll-In Nut	4
9	11-6041		4
10	3804		4

A.4 Enclosure and Camera Exploded Subassembly


Camera Enclosure Subassemb
SOLIDWORKS Educational Product. For Instructional Use Only.

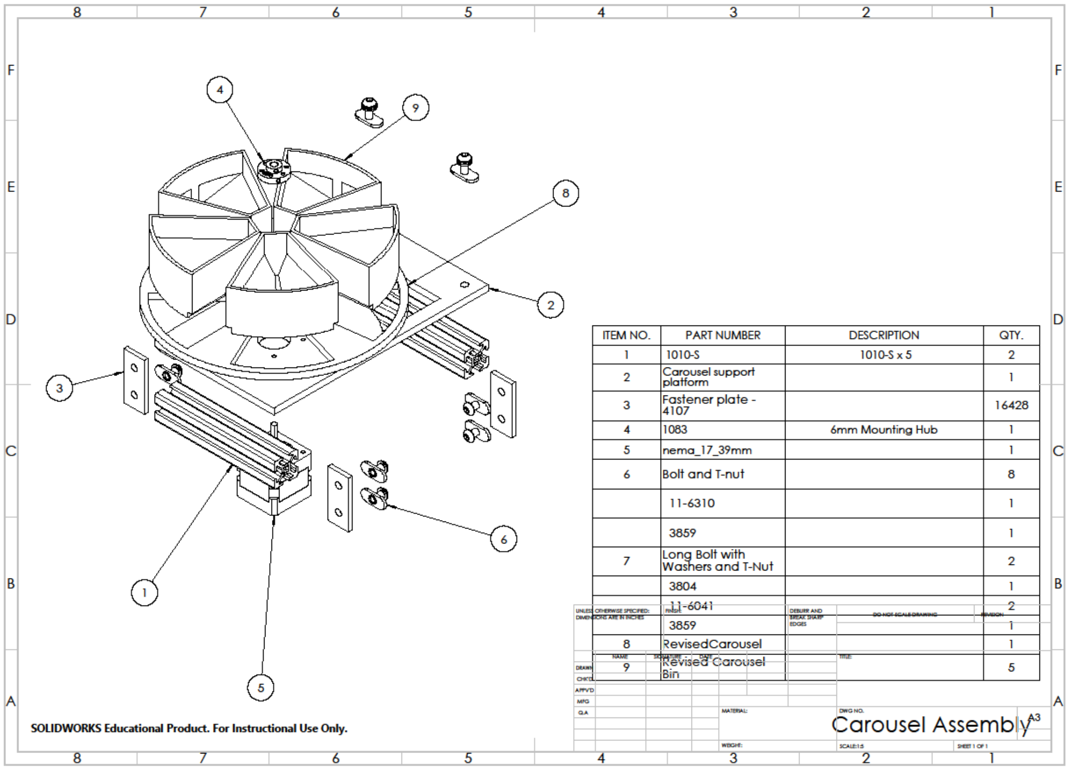
ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	1010-S	1010-S x 9.75	4
2	1010-S	1010-S x 7	6
3	1010-S	1010-S x 13	2
4	2062	Plastic Handle	1
5	Enclosure Top		1
6	1010-S	1010-S x 10	4
7	Enclosure Side	Enclosure Side - 6mm Expanded PVC 10.5 x 2.25	2
8	Enclosure Front	Eclosure Front - 6mm Thick PVC 2.25 x 7.5	2
9	4150		4
10	Pulse Camera		1
11	C125-0818-5M		1
12	Basler Lens CS-Mount Adapter v01		1
13	RL-SC-RSU24-T-W-10	Bright White LED Stirp (3 meters cut to quantity value)	2
14	14061	Coner Bracket	6
15	11-6310		37
16	3859		47
17	Mirror14061		2
18	Mirror3859		4
19	Diffusion Layer for Camera		1
20	4132	Corner Bracket	8
21	socket button head cap screw_am		16
22	Camera Mount		1
23	11-6065		2
24	12099	Plastic Hinge	2

A.5 Bin Selector Exploded Subassembly



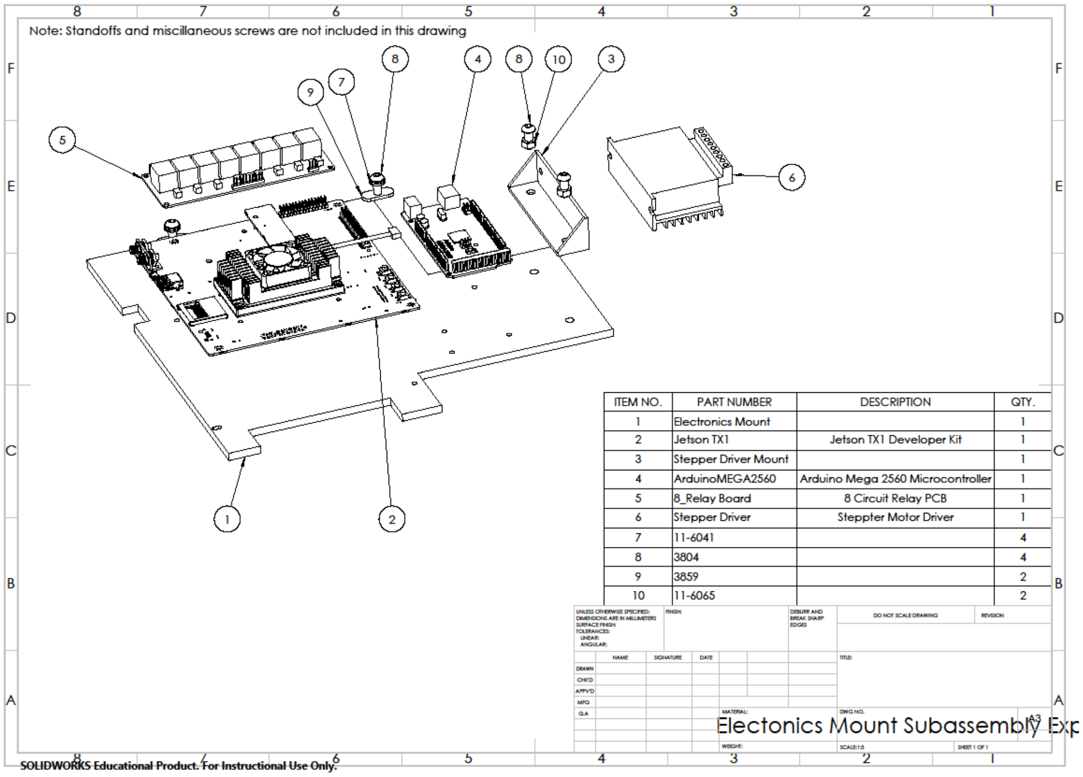
ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	1010-S	1010-S x 8	2
2	TGY-1270HV		1
3	Flipper		1
4	Servo Support		1
5	4132	Corner Bracket	4
6	11-6310		8
7	3859		8
8	3804		4
9	11-6065		4
10	11-6041		4
11	14061	Coner Bracket	2

A.6 Bin Carousel Exploded Subassembly



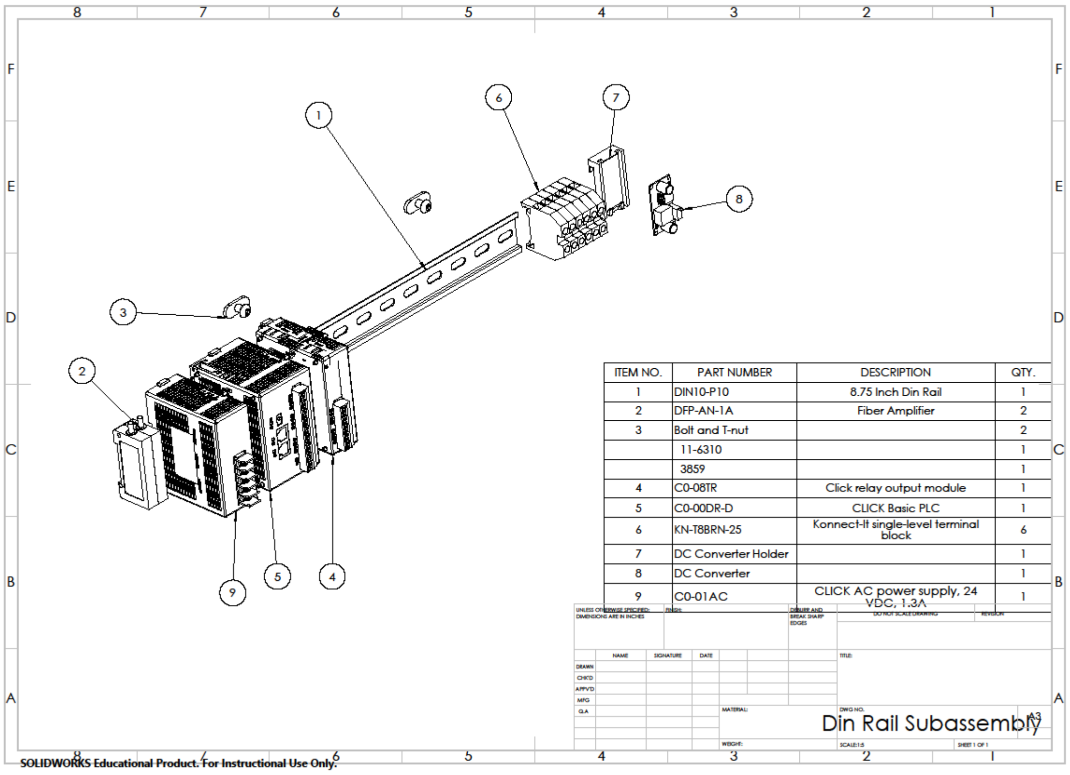
ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	1010-S	1010-S x 5	2
2	Carousel support platform		1
3	Fastener plate - 4107		16428
4	1083	6mm Mounting Hub	1
5	nema_17_39mm		1
6	Bolt and T-nut		8
	11-6310		1
	3859		1
7	Long Bolt with Washers and T-Nut		2
	3804		1
	11-6041		2
	3859		1
8	RevisedCarousel		1
9	Revised Carousel Bin		5

A.7 Electronic Plate Exploded Subassembly


Note: Standoffs and miscillaneous screws are not included in this drawing
1
2
3
4
5
6
7
8
9
10
Electonics Mount Subassembly Exp
A3
SOLIDWORKS Educational Product. For Instructional Use Only.

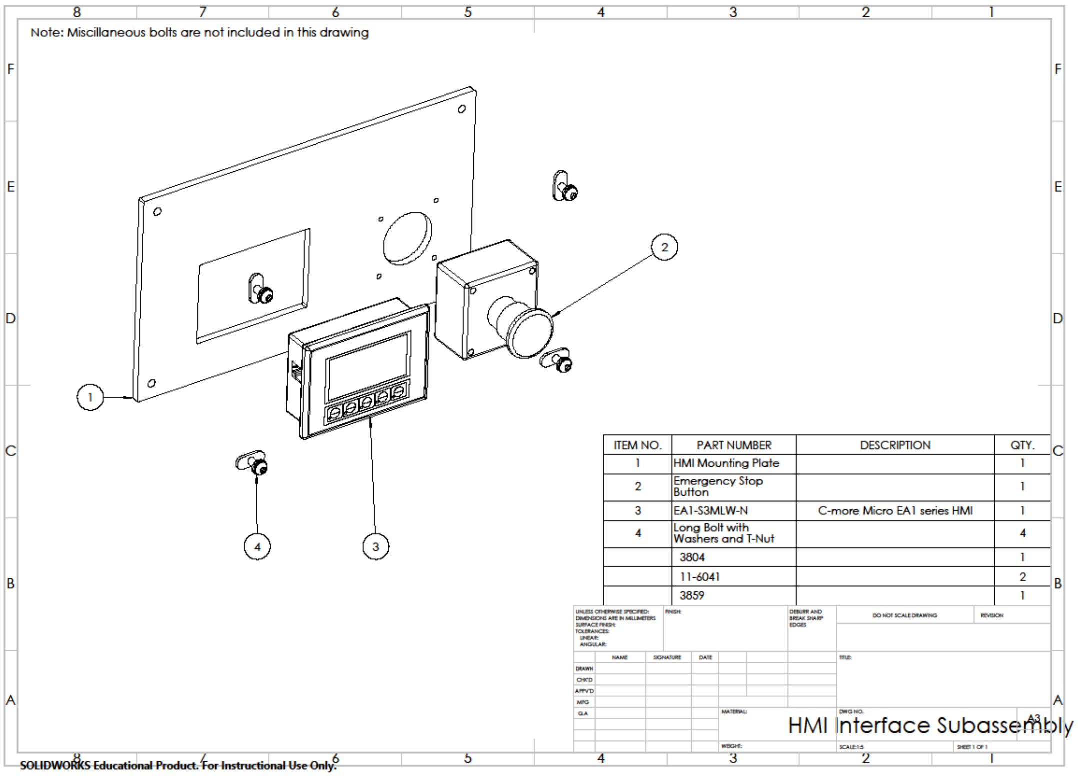
ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Electronics Mount		1
2	Jetson TX1	Jetson TX1 Developer Kit	1
3	Stepper Driver Mount		1
4	ArduinoMEGA2560	Arduino Mega 2560 Microcontroller	1
5	8_Relay Board	8 Circuit Relay PCB	1
6	Stepper Driver	Steppter Motor Driver	1
7	11-6041		4
8	3804		4
9	3859		2
10	11-6065		2

A.8 Din Rail PLC Exploded Subassembly



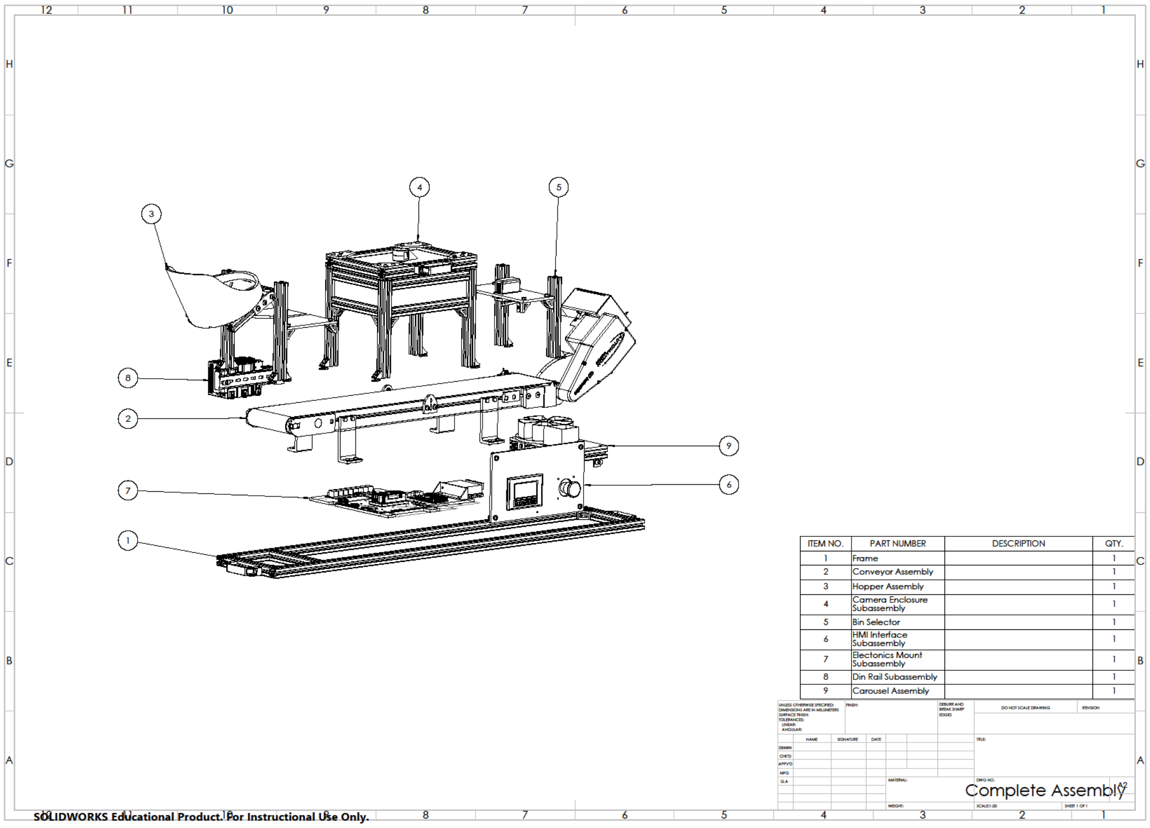
ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	DIN10-P10	8.75 Inch Din Rail	1
2	DFP-AN-1A	Fiber Amplifier	2
3	Bolt and T-nut		2
	11-6310		1
	3859		1
4	C0-08TR	Click relay output module	1
5	C0-00DR-D	CLICK Basic PLC	1
6	KN-T8BRN-25	Konnect-It single-level terminal block	6
7	DC Converter Holder		1
8	DC Converter		1
9	C0-01AC	CLICK AC power supply, 24 VDC, 1.3A	1

A.9 HMI Interface Exploded Subassembly


Note: Miscellaneous bolts are not included in this drawing
1
2
3
4
HMI Interface Subassembly
A3
SOLIDWORKS Educational Product. For Instructional Use Only.

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	HMI Mounting Plate		1
2	Emergency Stop Button		1
3	EA1-S3MLW-N	C-more Micro EA1 series HMI	1
4	Long Bolt with Washers and T-Nut		4
	3804		1
	11-6041		2
	3859		1

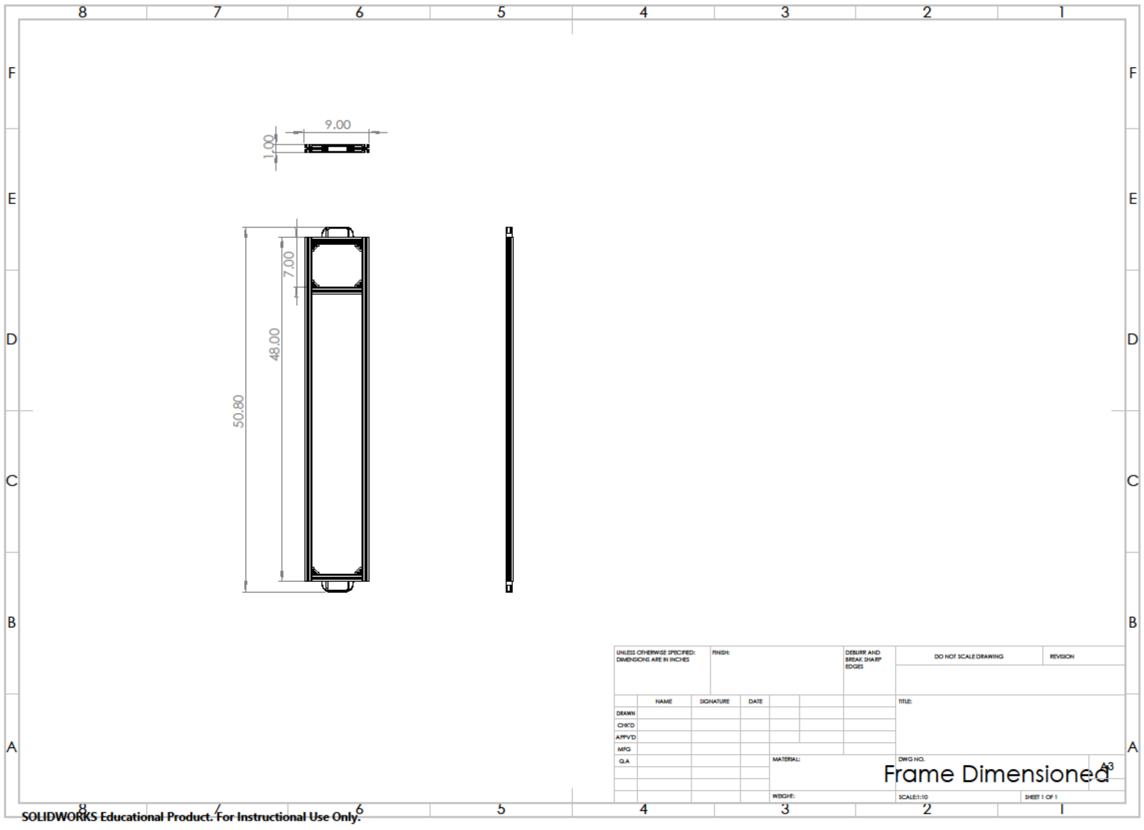
A.10 Overall Exploded Assembly



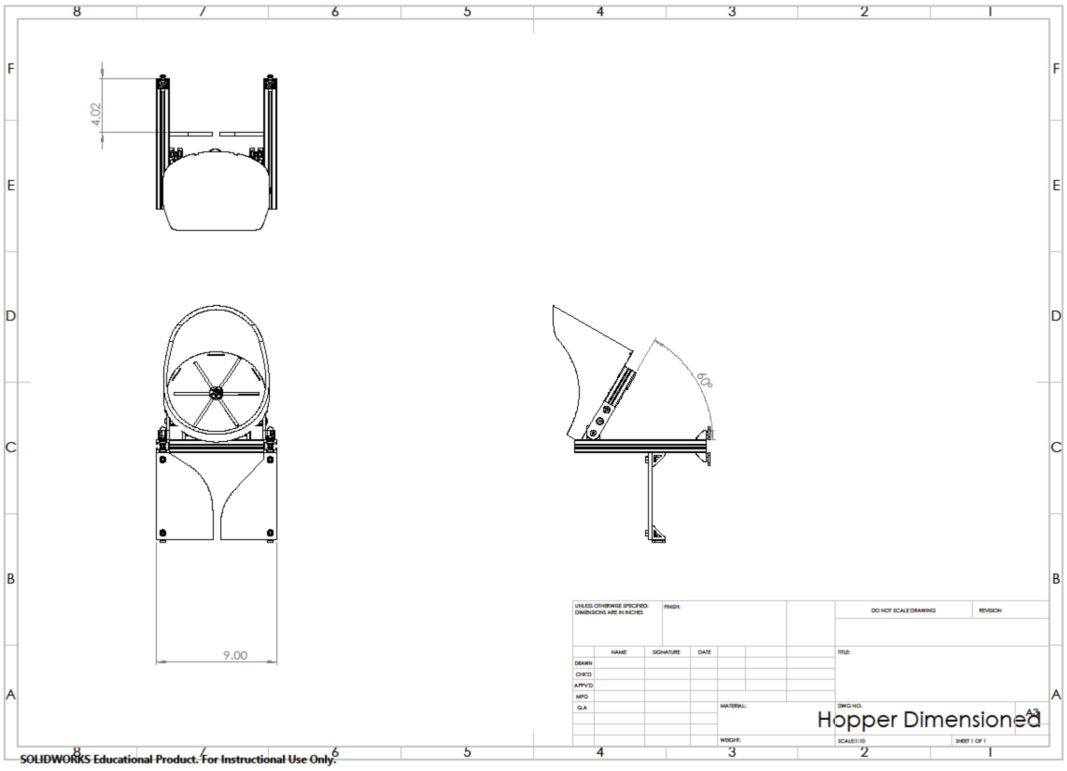
ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Frame		1
2	Conveyor Assembly		1
3	Hopper Assembly		1
4	Camera Enclosure Subassembly		1
5	Bin Selector		1
6	HMI Interface Subassembly		1
7	Electonics Mount Subassembly		1
8	Din Rail Subassembly		1
9	Carousel Assembly		1

APPENDIX B. SUBASSEMBLY DRAWINGS WITH DIMENSIONS

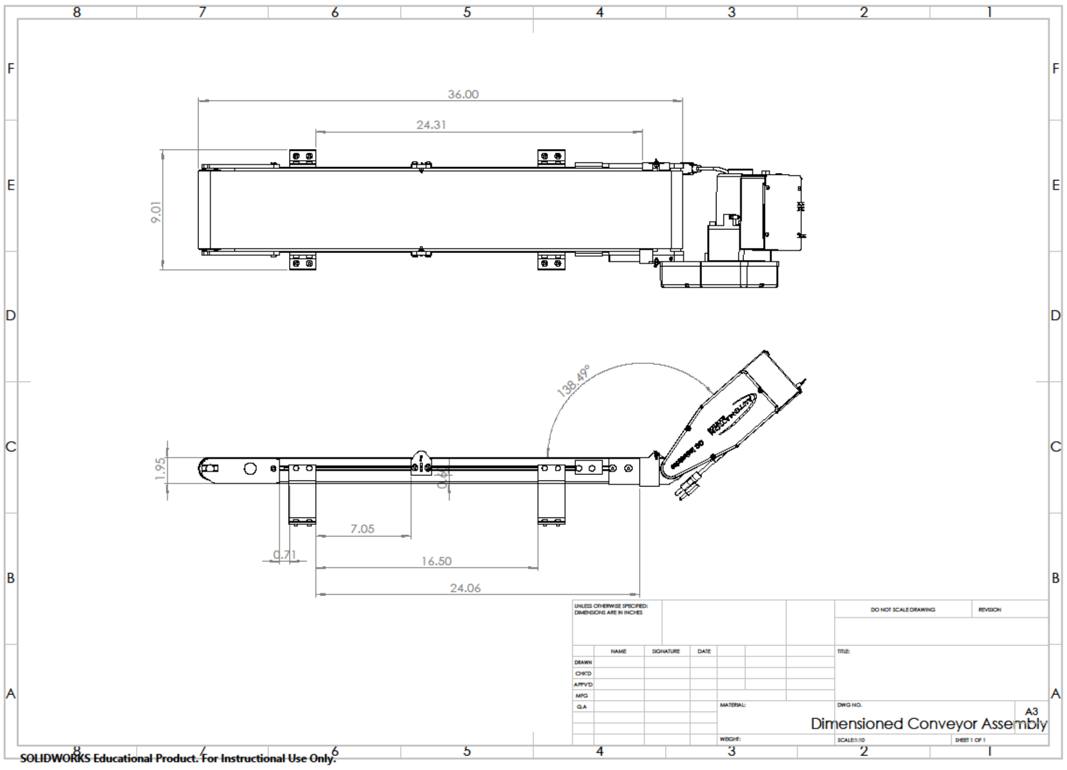
B.1 System Frame



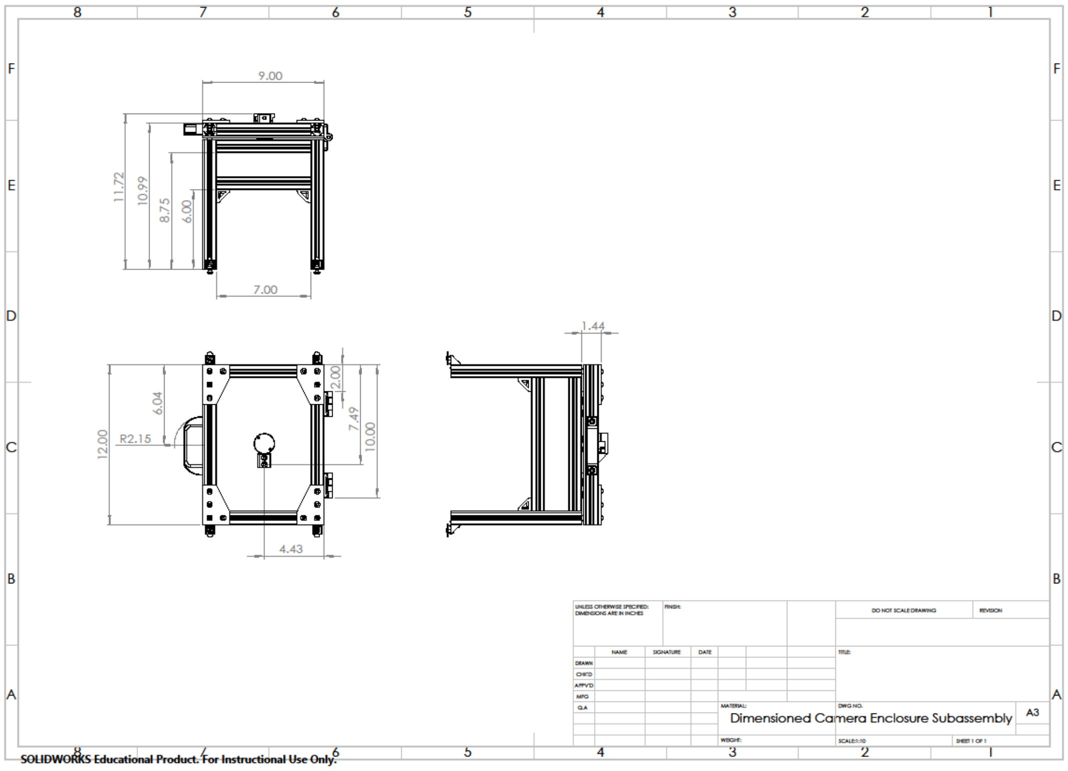
B.2 Hopper Subassembly



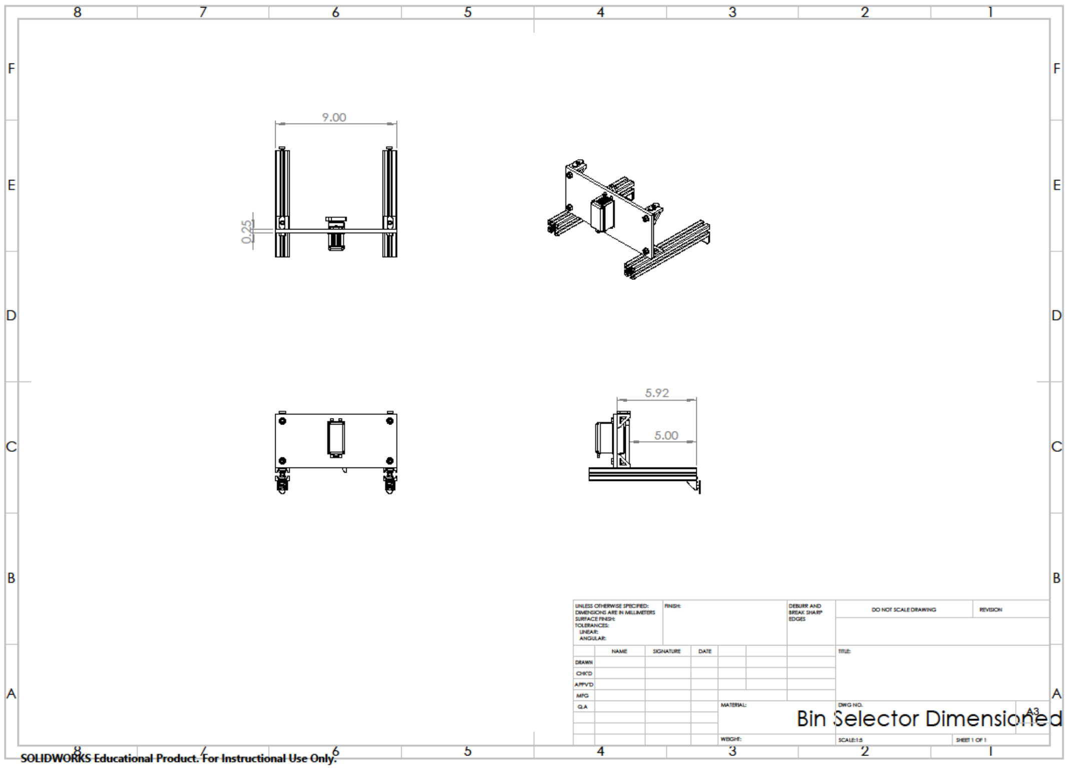
B.3 Conveyor Subassembly



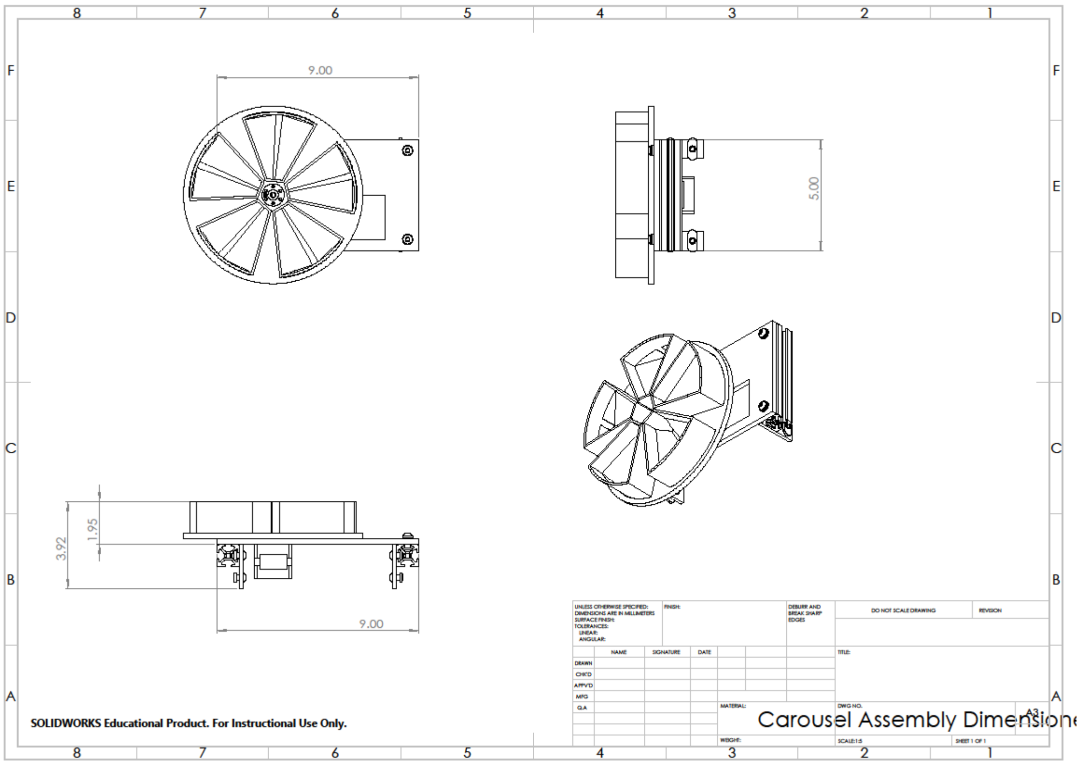
B.4 Lighting Enclosure and Camera System


9.00
11.72
10.99
8.75
6.00
7.00
1.44
6.04
2.00
R2.15
12.00
7.49
10.00
4.43
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES
DO NOT SCALE DRAWING
Dimensioned Camera Enclosure Subassembly
A3
SCALE:1:10
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

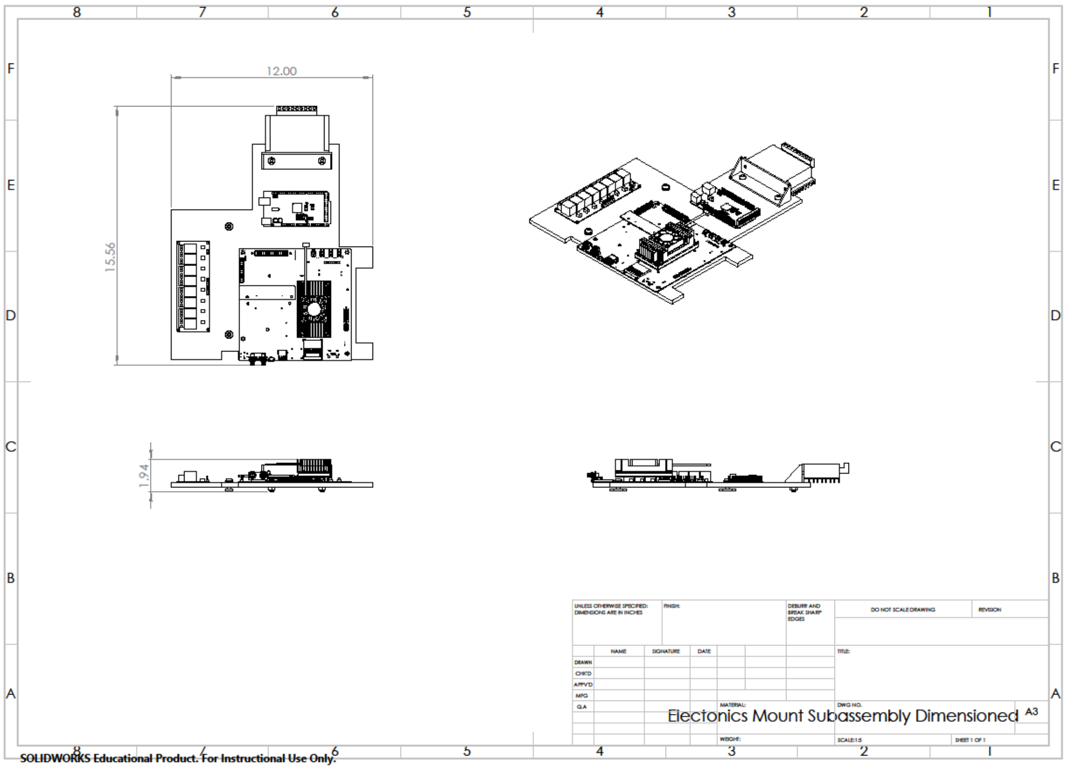
B.5 Bin Selector Subassembly


9.00
0.25
5.92
5.00
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS
DO NOT SCALE DRAWING
Bin Selector Dimensioned
A3
SCALE:1:5
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

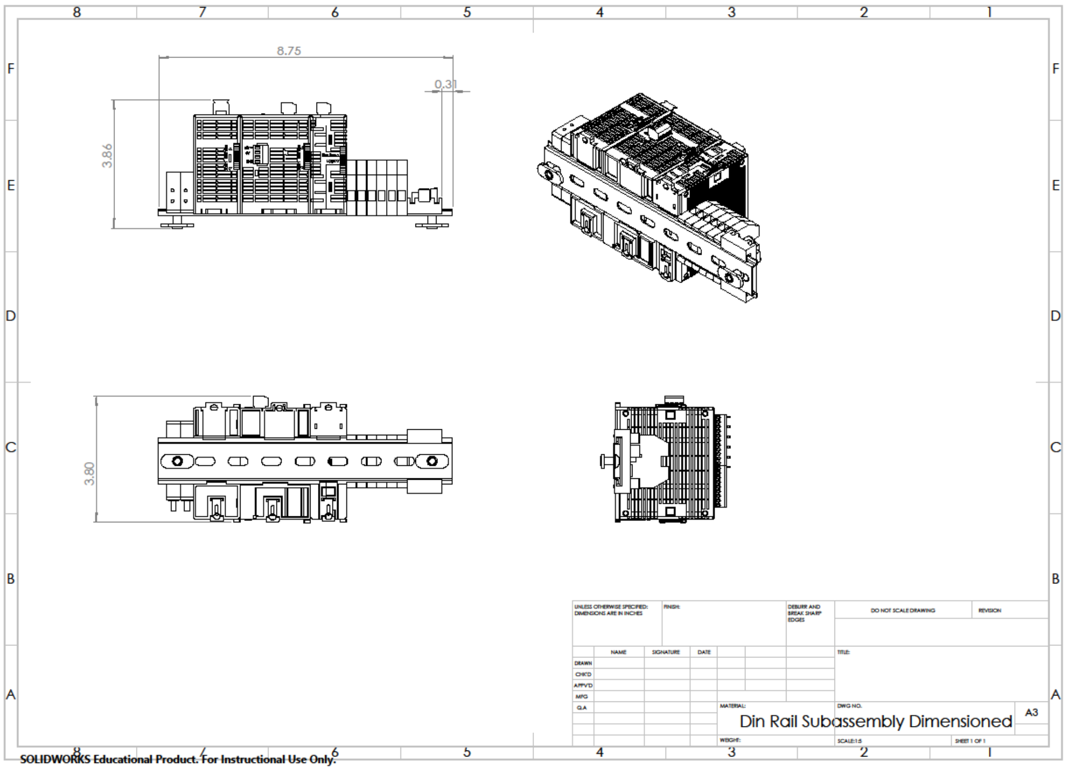
B.6 Bin Carousel Subassembly


9.00
5.00
1.95
3.92
9.00
SOLIDWORKS Educational Product. For Instructional Use Only.
Carousel Assembly Dimensione
A3

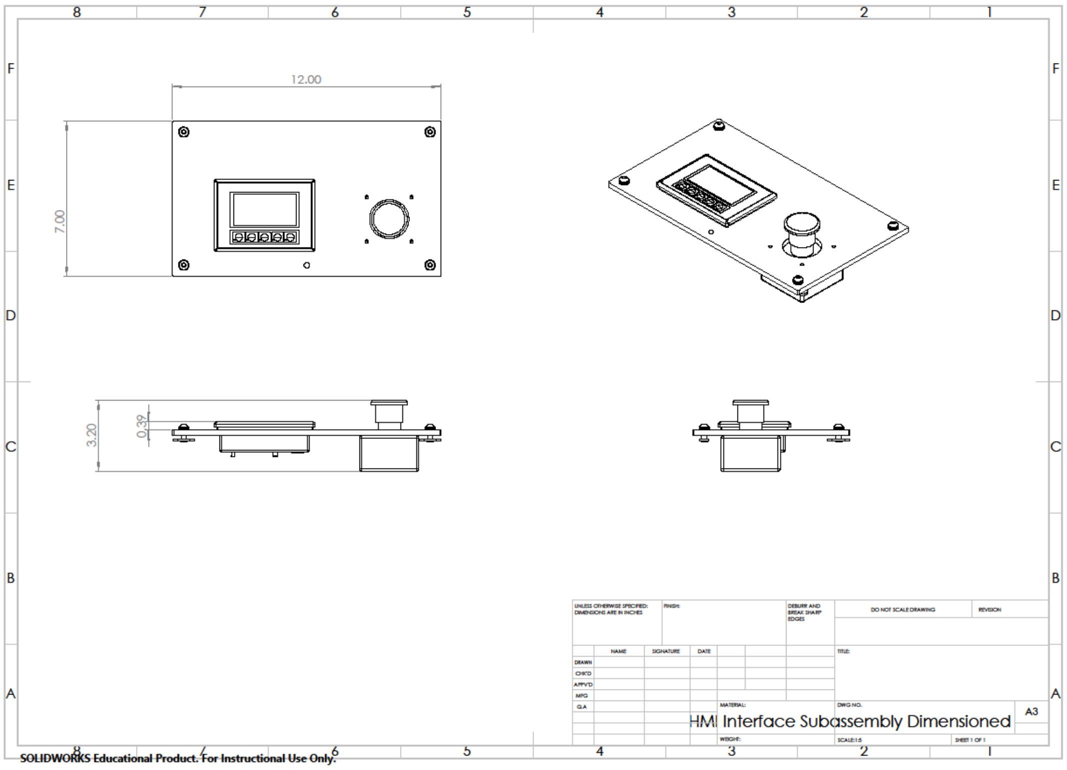
B.7 Electronics Plate Subassembly


12.00
15.56
1.94
Electonics Mount Subassembly Dimensioned
A3
SOLIDWORKS Educational Product. For Instructional Use Only.

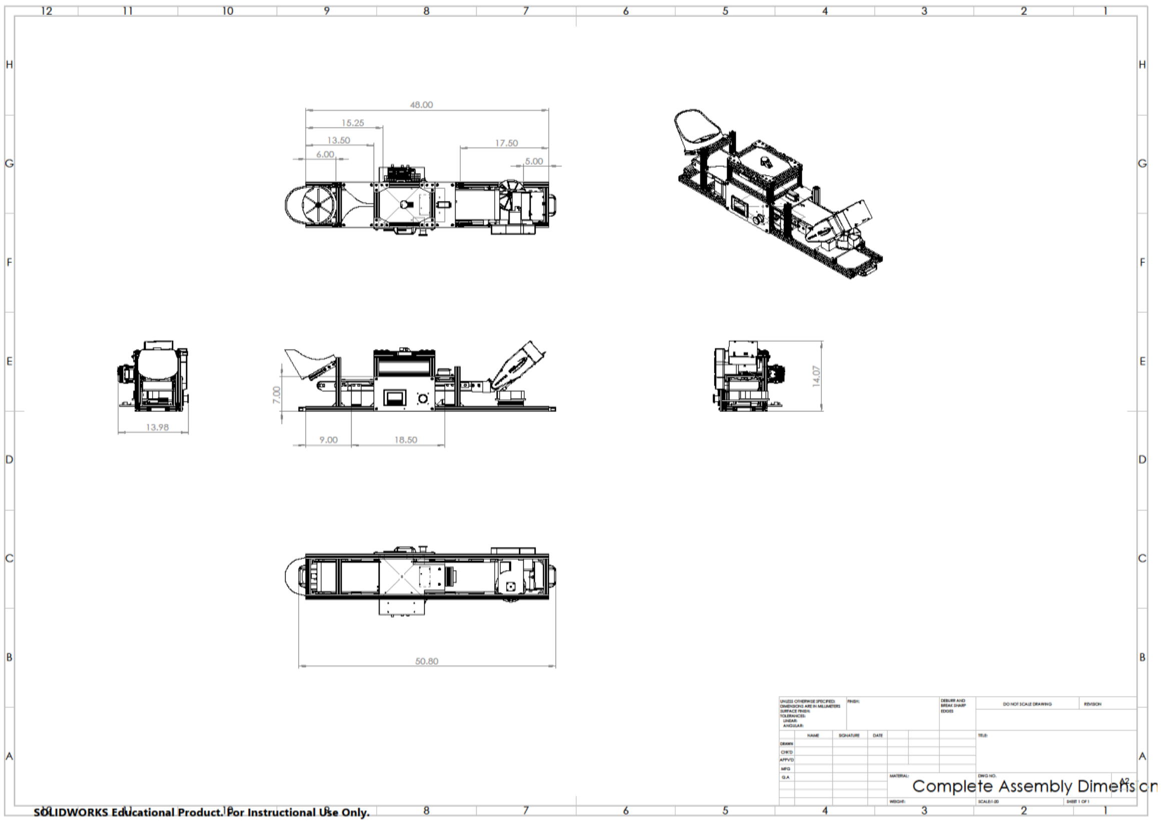
B.8 Din Rail PLC Subassembly


8.75
0.31
3.86
3.80
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
TITLE:
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
DWG NO.
Din Rail Subassembly Dimensioned
A3
WEIGHT:
SCALE:1:5
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

B.9 HMI Interface Subassembly


12.00
7.00
3.20
0.39
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
TITLE:
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
DWG NO.
HMI Interface Subassembly Dimensioned
A3
WEIGHT:
SCALE:1:5
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

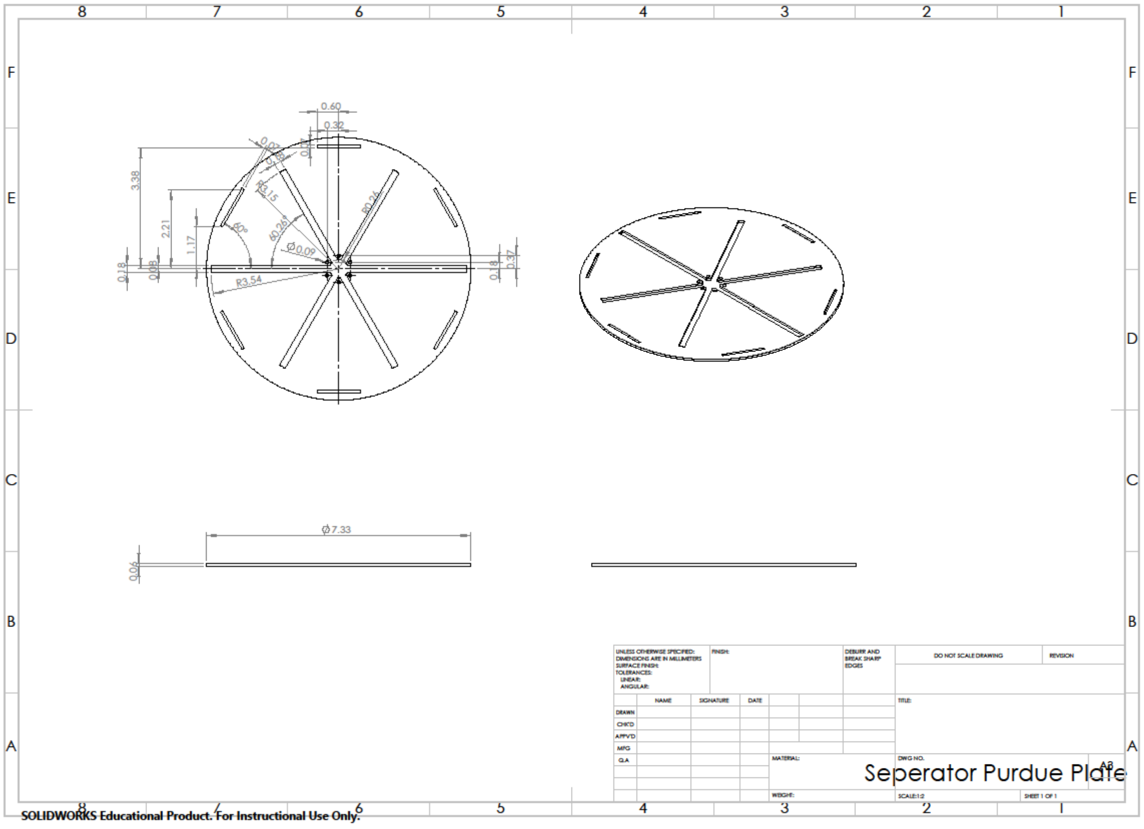
B.10 Overall Subassembly


48.00
15.25
13.50
6.00
17.50
5.00
13.98
7.00
9.00
18.50
14.07
50.80
Complete Assembly Dimensi
SOLIDWORKS Educational Product. For Instructional Use Only.

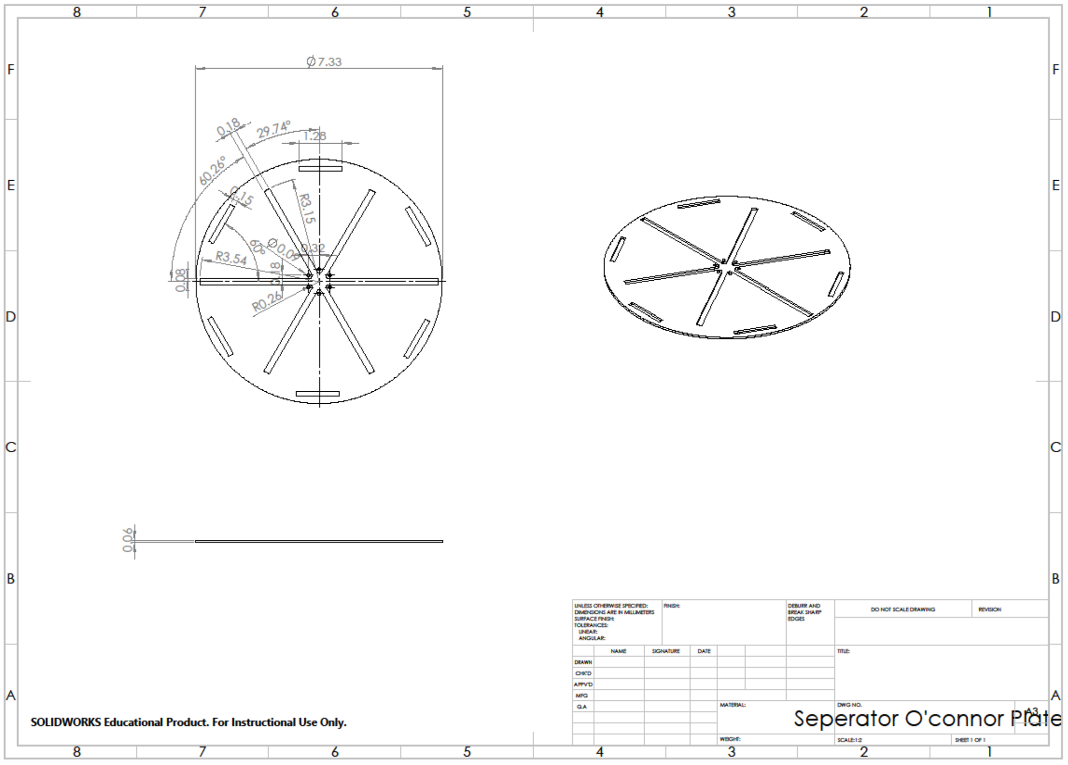
APPENDIX C. CUSTOM PART DRAWINGS WITH DIMENSIONS

This section contains drawings with dimensions for all parts that were cut from acrylic, 3D printed, or a model was not available from the supplier of the part. 3D models are not shown for parts where a supplier has provided a model.

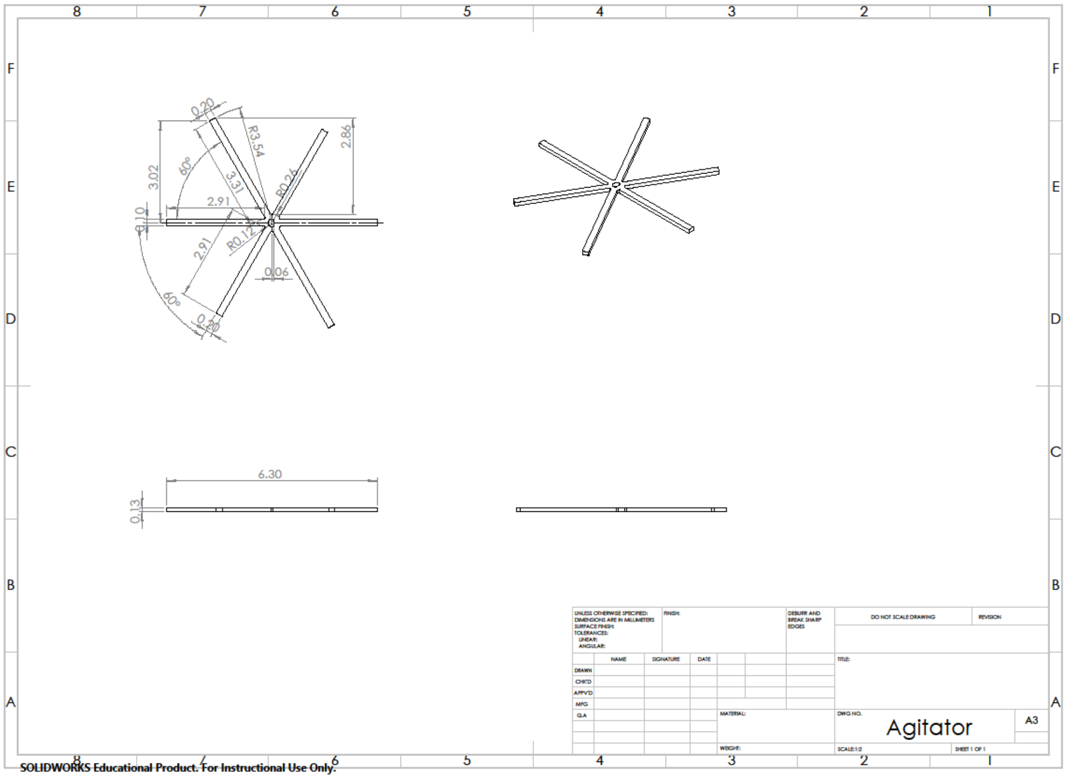
C.1 Separator Small Peg Plate



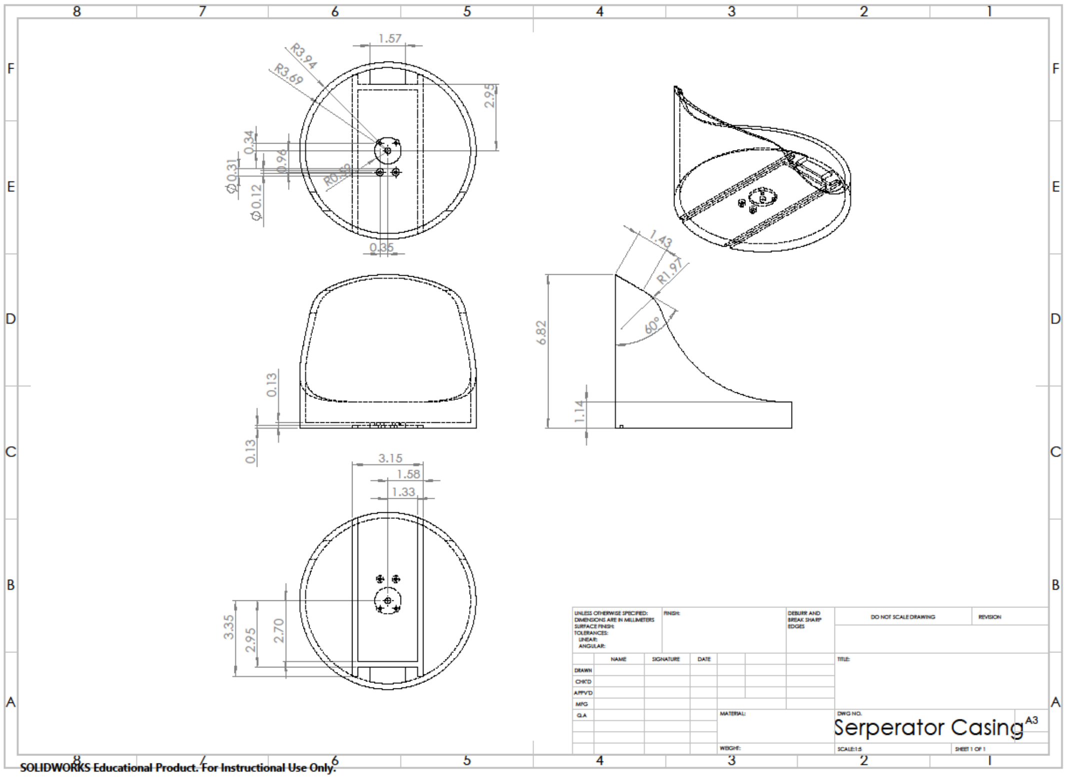
C.2 Separator Medium and Keyed Peg Plate



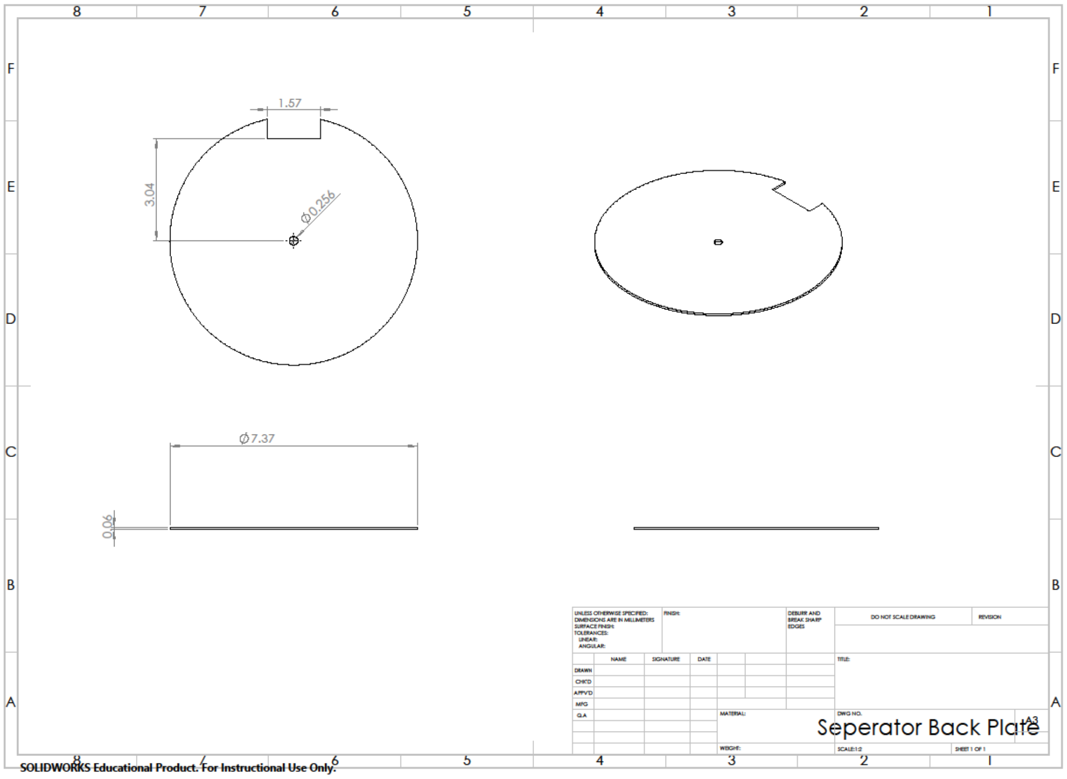
C.3 Agitator



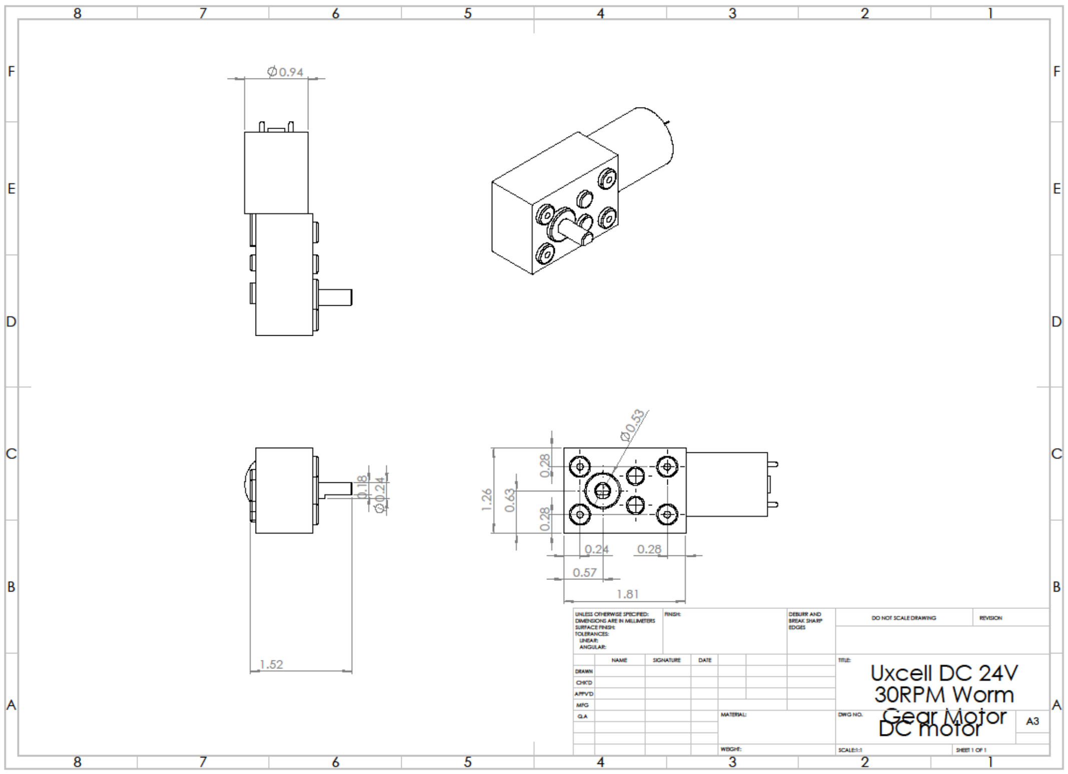
C.4 Separator Casing



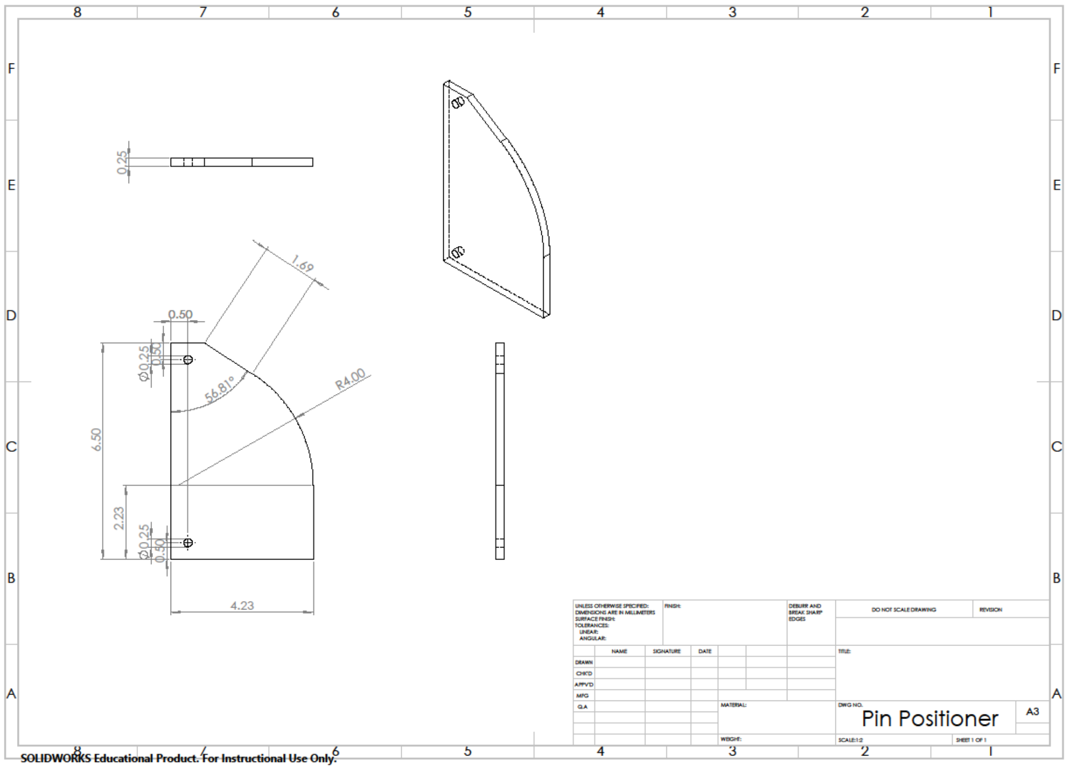
C.5 Separator Back Plate



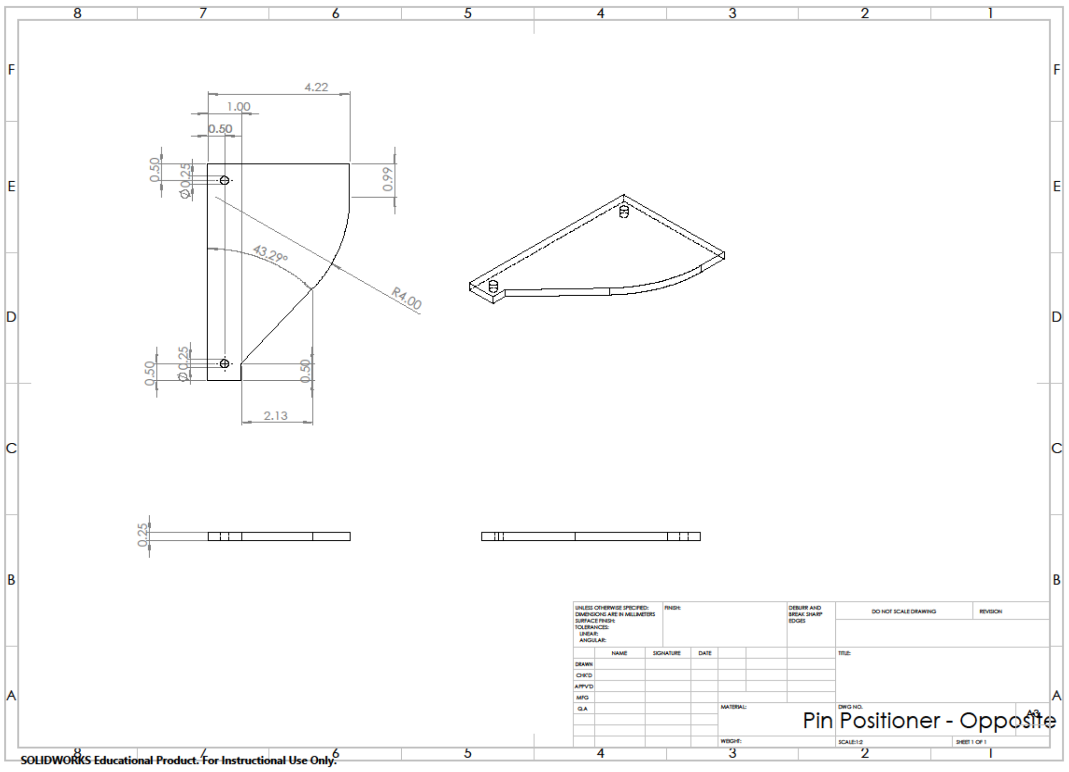
C.6 DC Motor



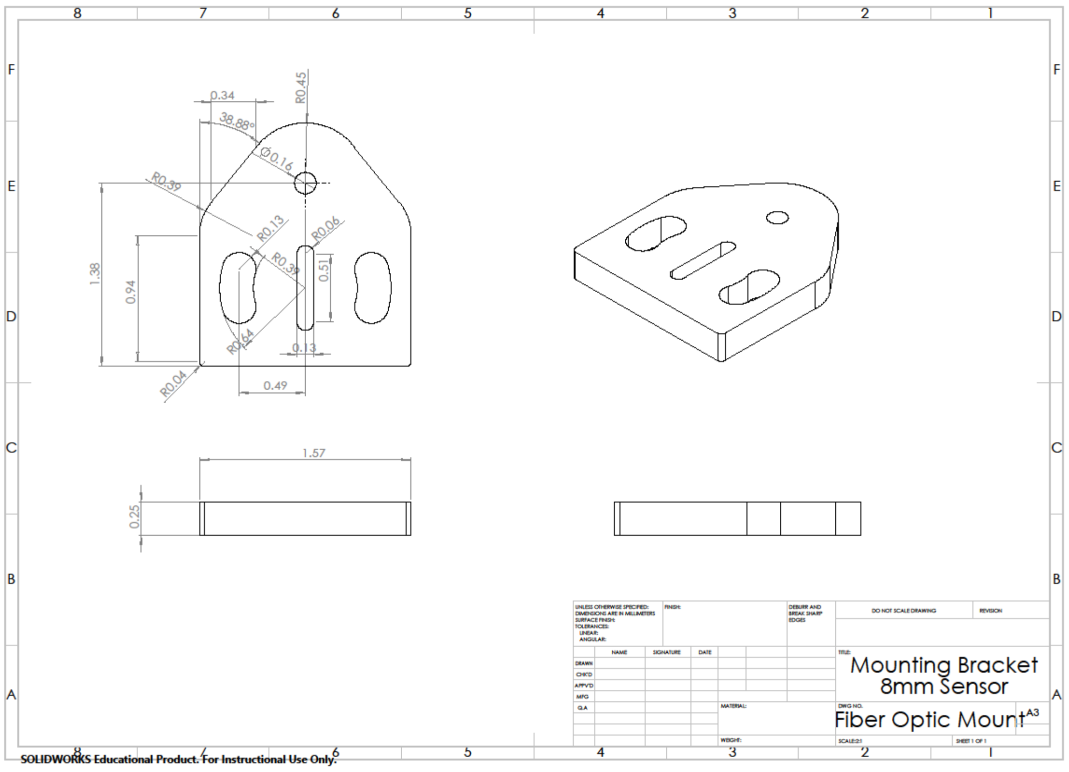
C.7 Pin Positioner



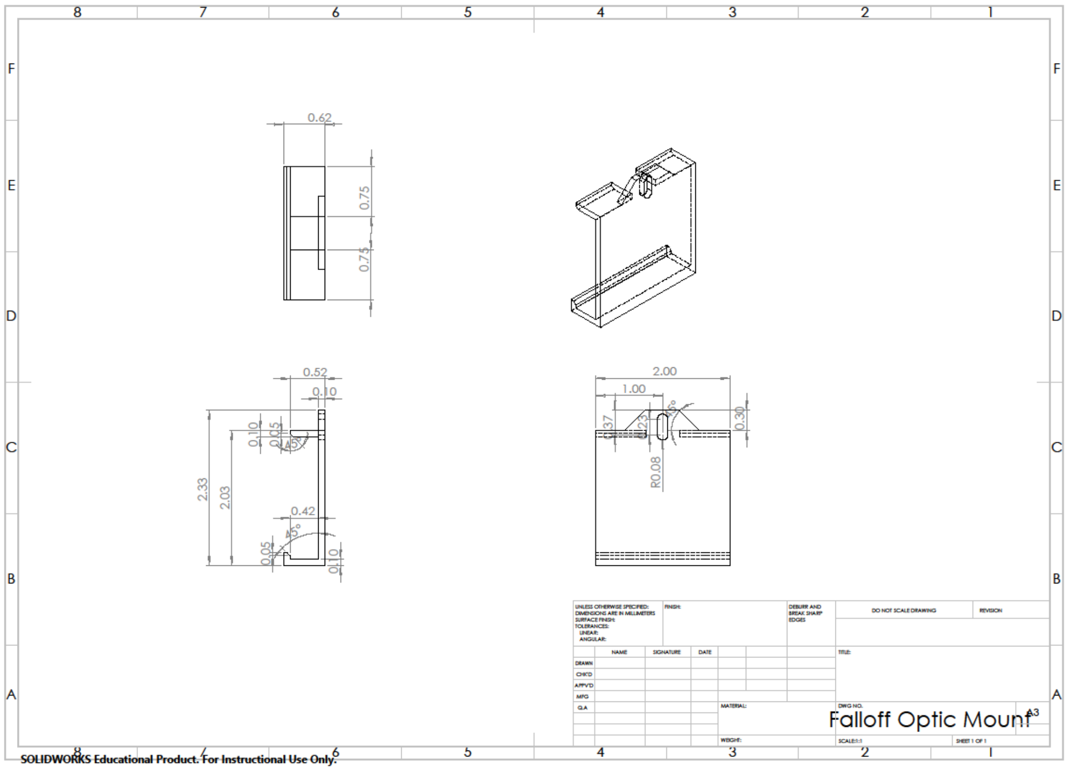
C.8 Pin Positioner Opposite


4.22
1.00
0.50
0.50
Ø0.25
0.99
43.29°
R4.00
0.50
Ø0.25
0.50
2.13
0.25
Pin Positioner - Opposite
A3
SOLIDWORKS Educational Product. For Instructional Use Only.

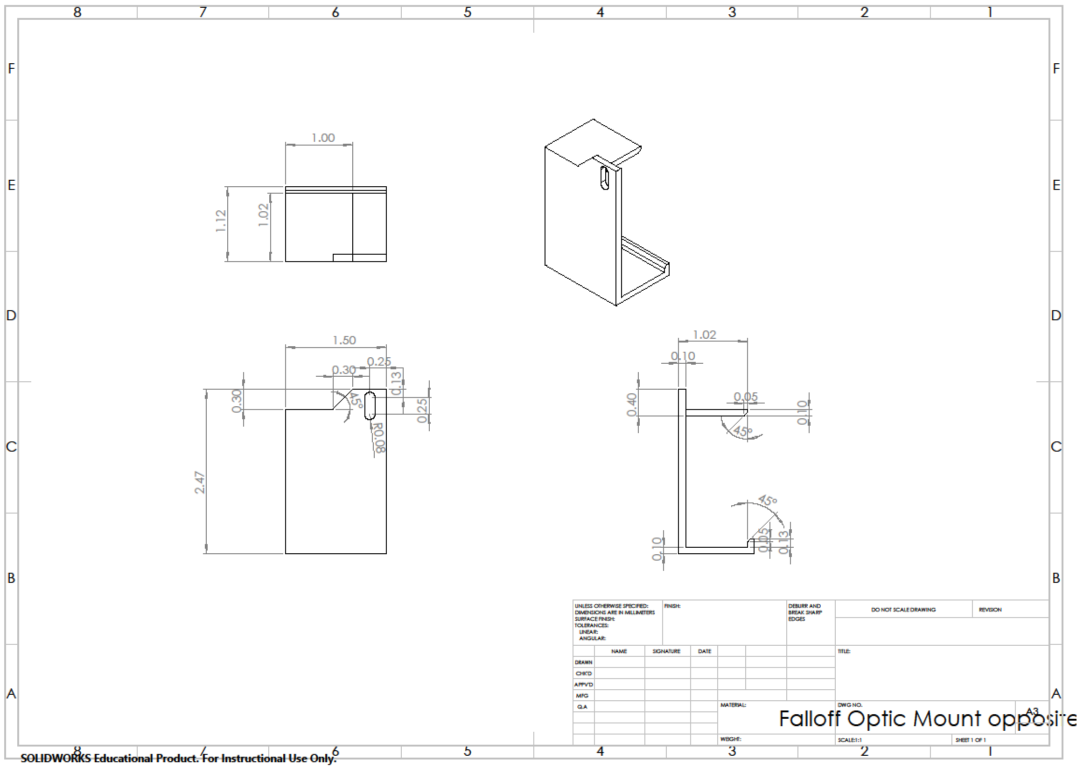
C.9 Lighting Enclosure Fiber Optic Mounts



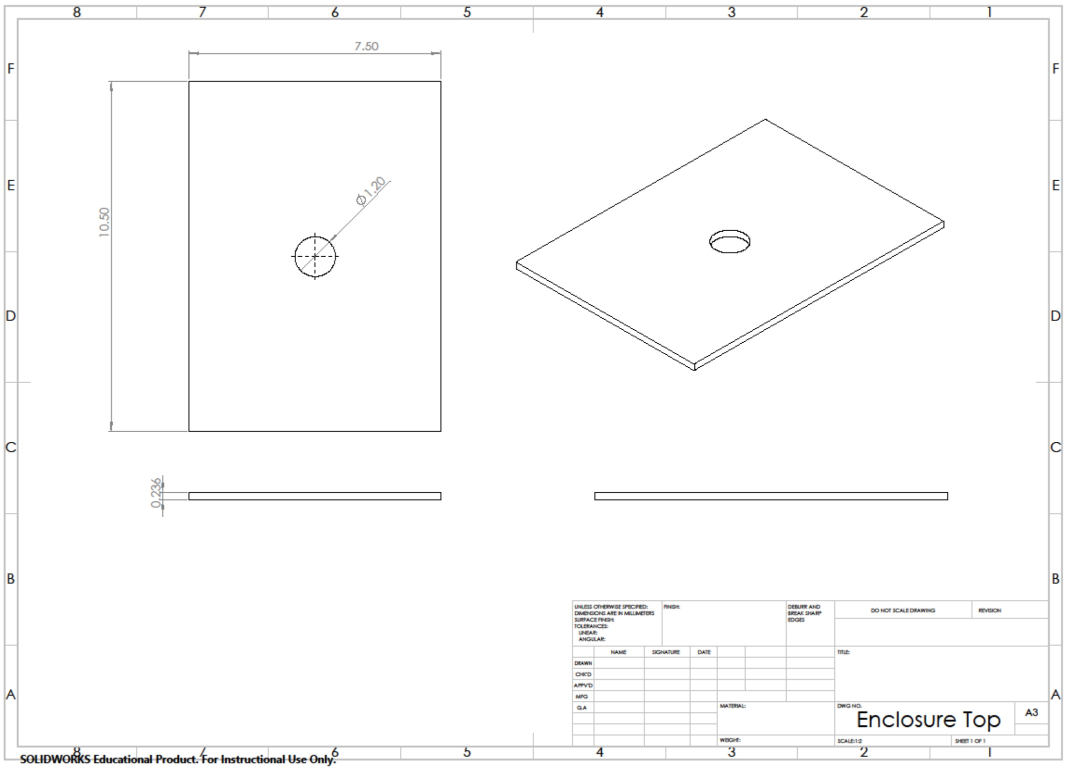
C.10 Falloff Optic Mount



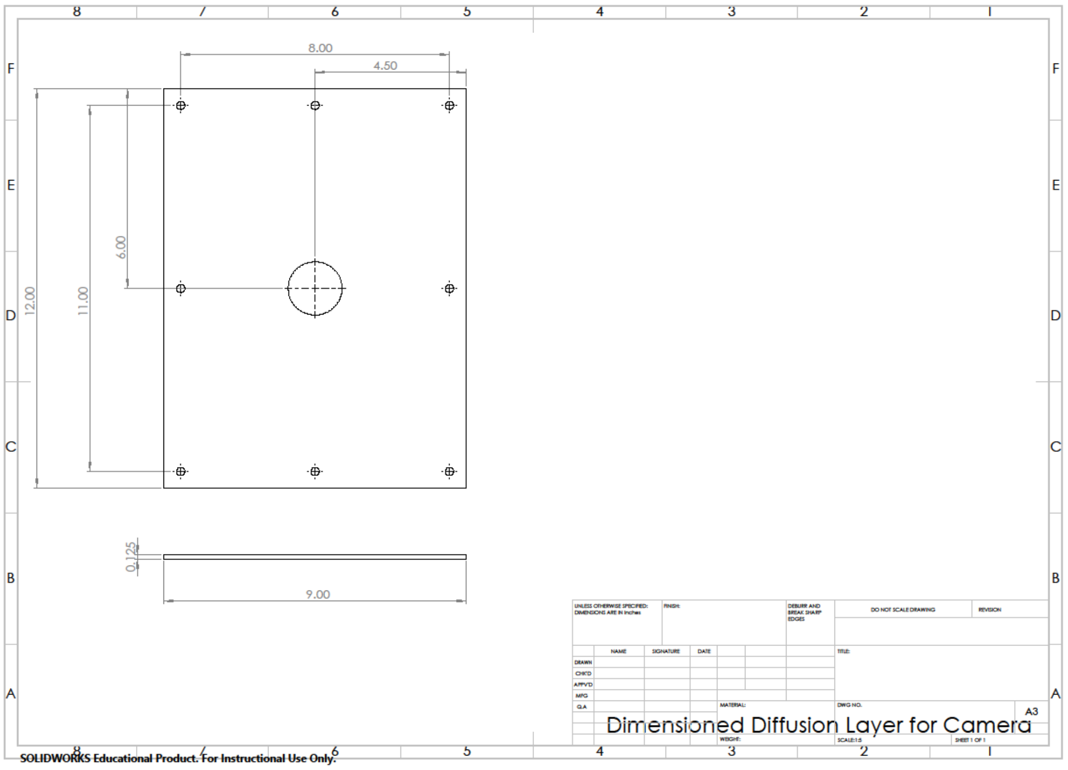
C.11 Falloff Optic Mount Opposite


1.00
1.12
1.02
1.50
0.30
0.25
0.13
0.30
0.25
45°
R0.08
2.47
1.02
0.10
0.40
0.05
0.10
45°
0.10
0.05
0.13
Falloff Optic Mount opposite
A3
SOLIDWORKS Educational Product. For Instructional Use Only.

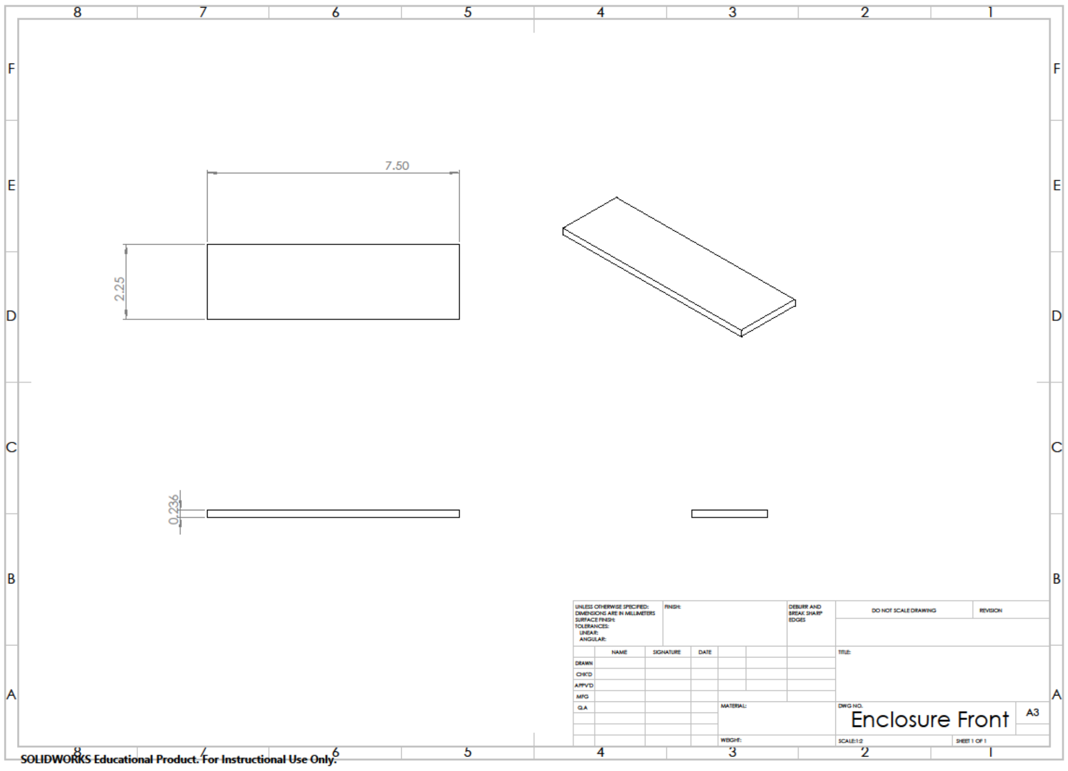
C.12 Enclosure Top



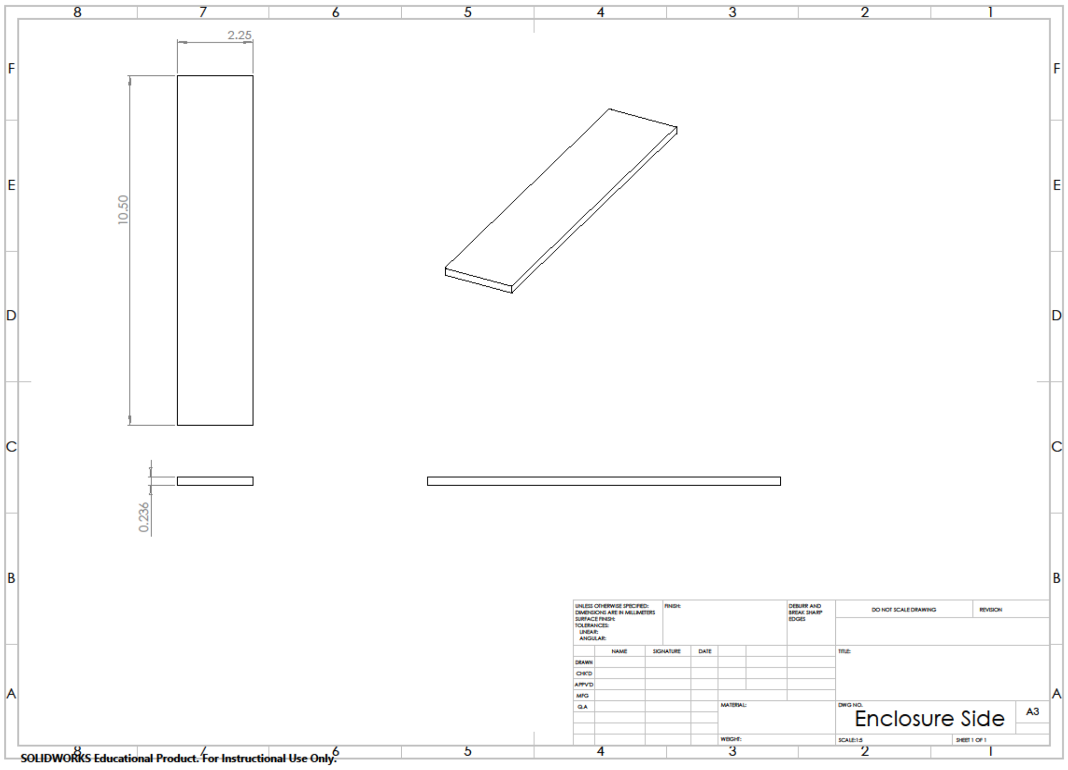
C.13 Diffusion Plate


8.00
4.50
12.00
11.00
6.00
0.125
9.00
Dimensioned Diffusion Layer for Camera
A3
SCALE:1:5
SOLIDWORKS Educational Product. For Instructional Use Only.

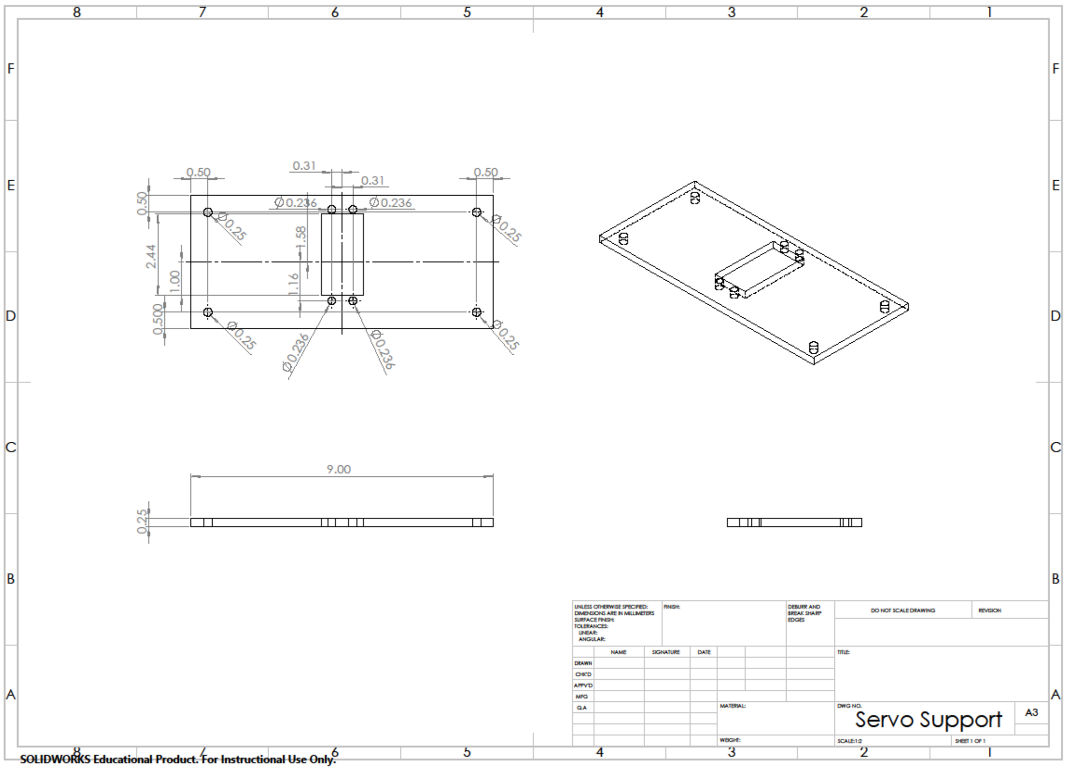
C.14 Enclosure Front



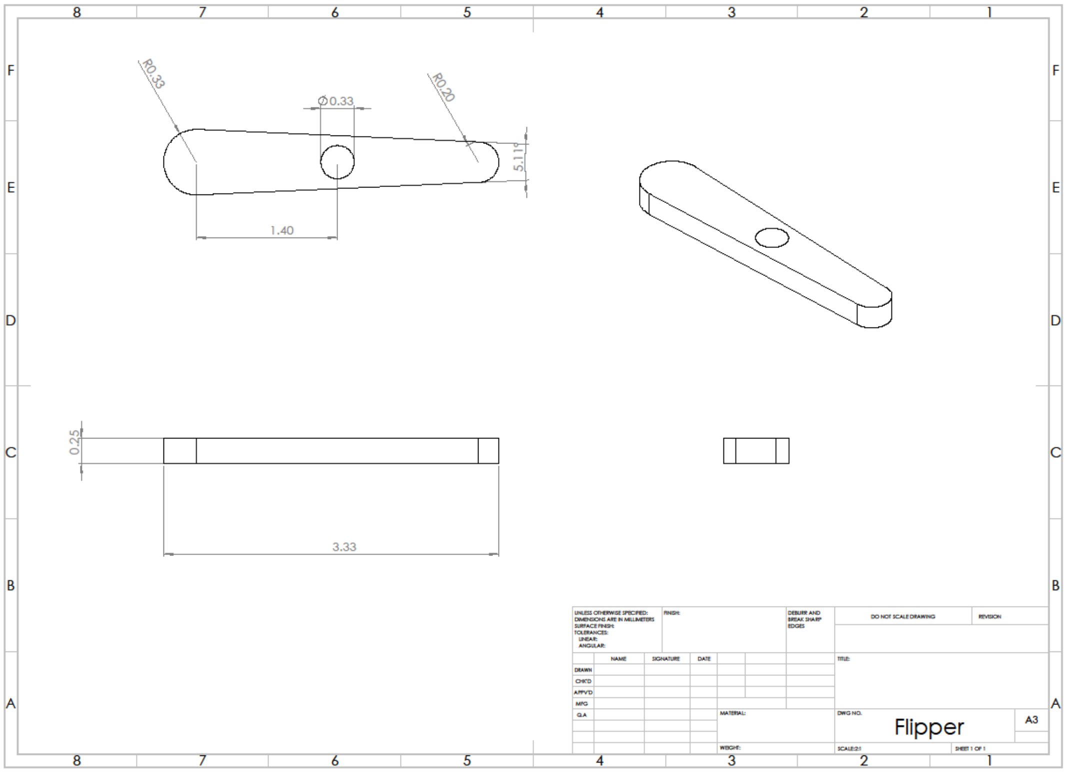
C.15 Enclosure Side



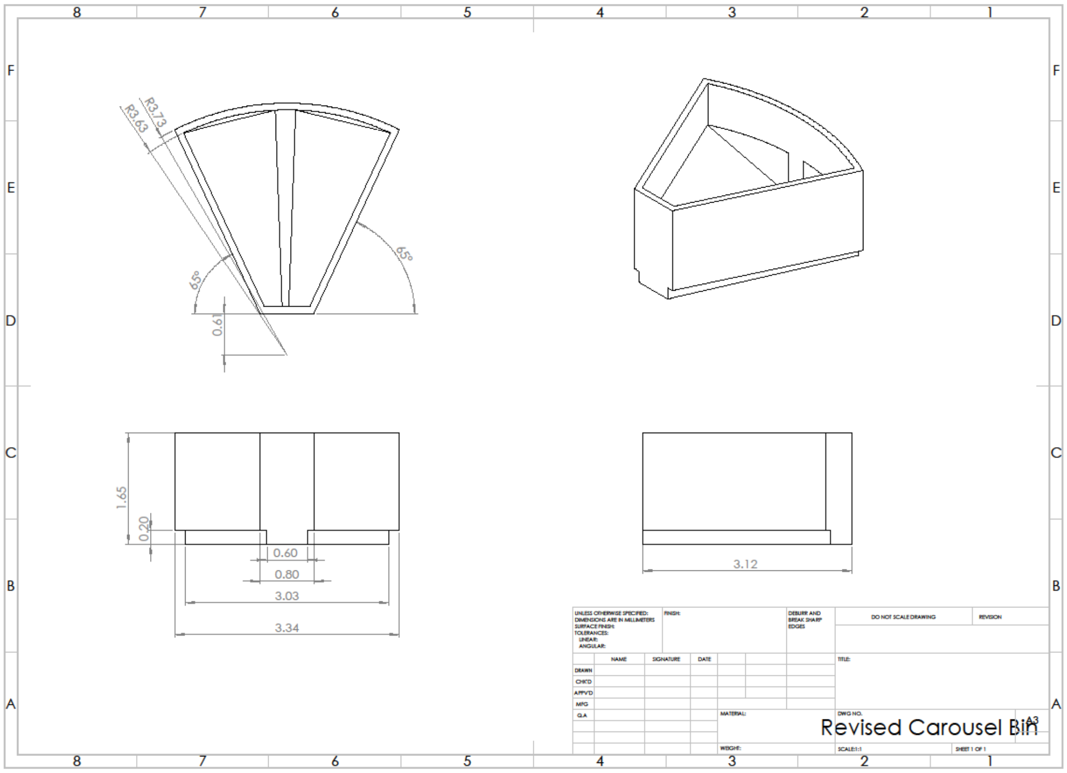
C.16 Servo Support



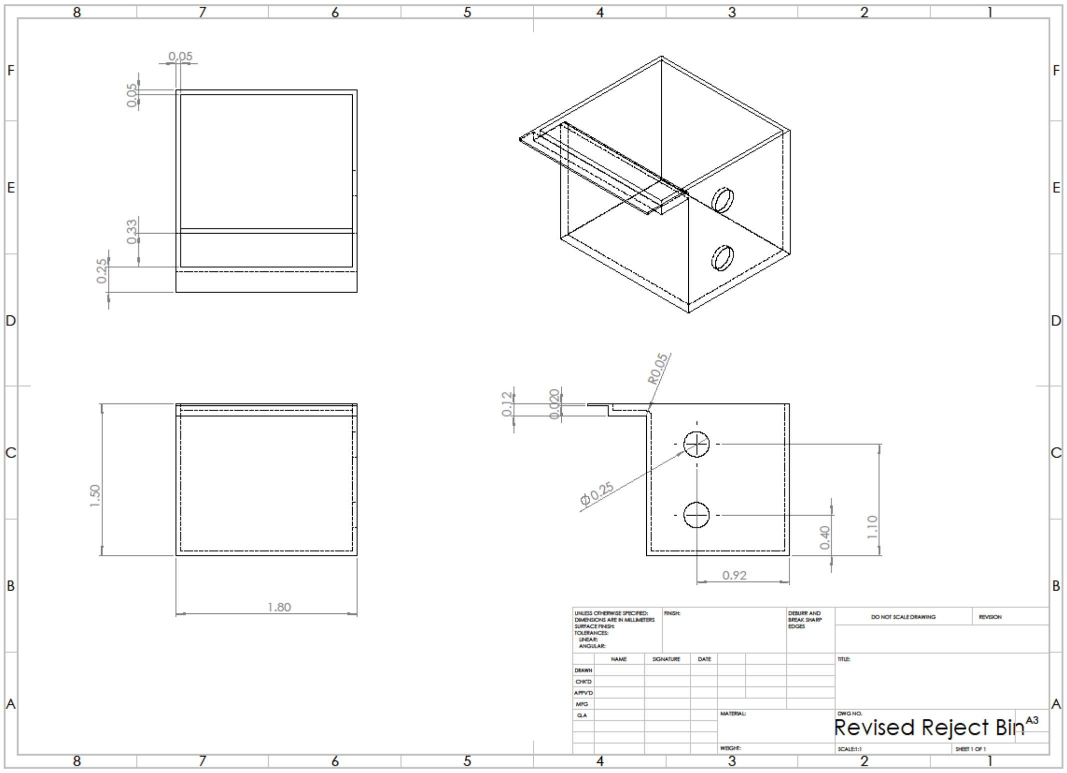
C.17 Flipper



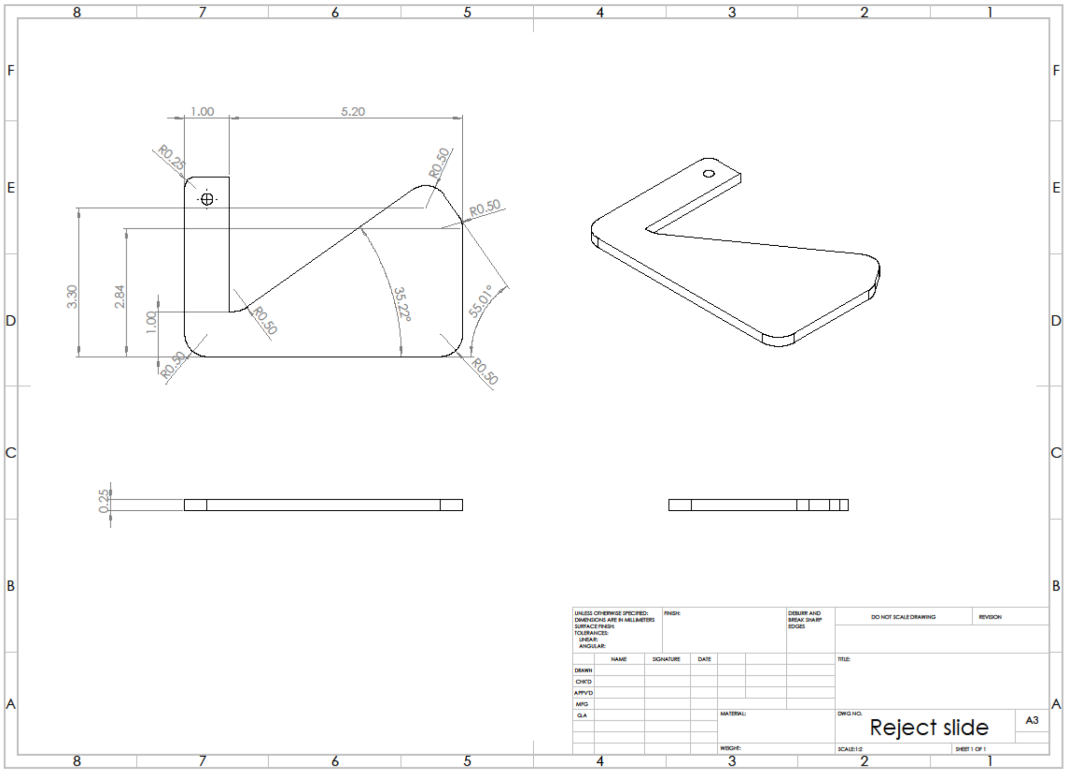
C.18 Carousel Bin



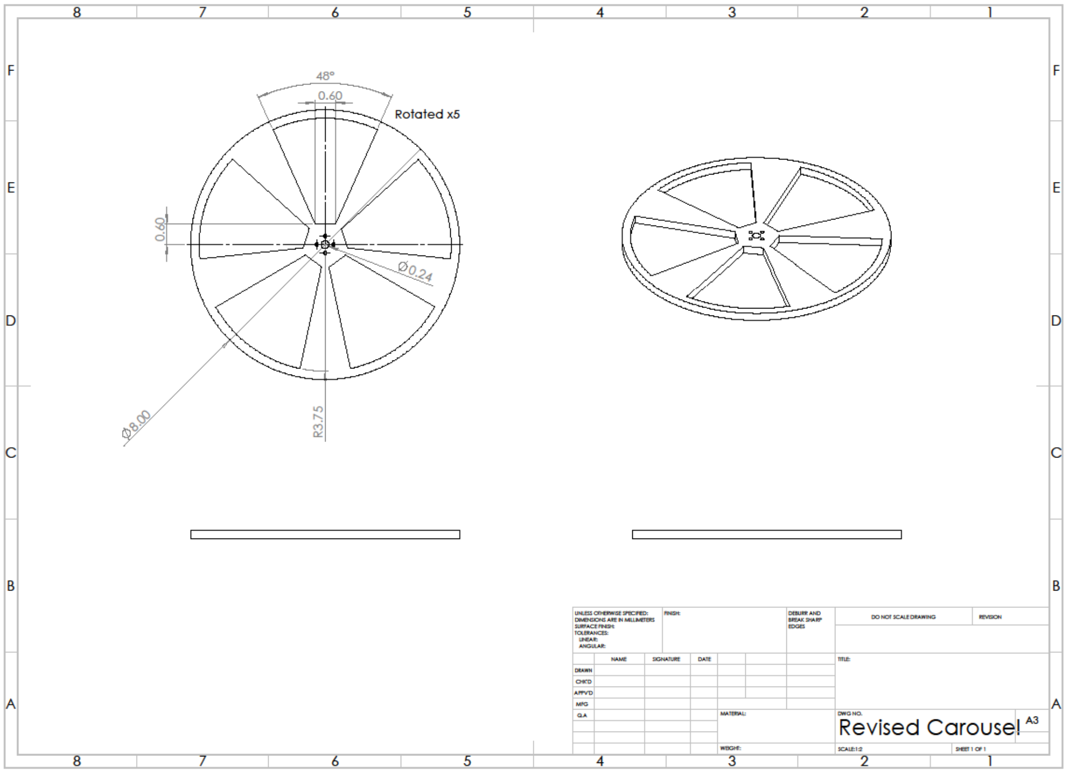
C.19 Reject Bin



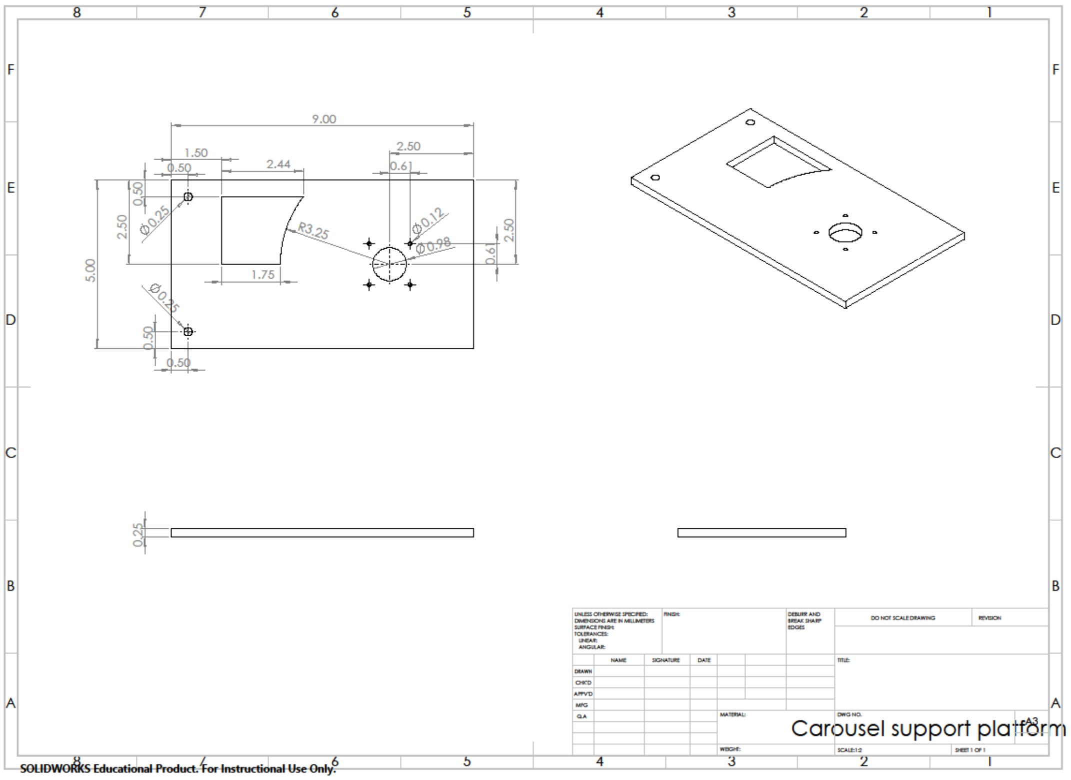
C.20 Reject Slide



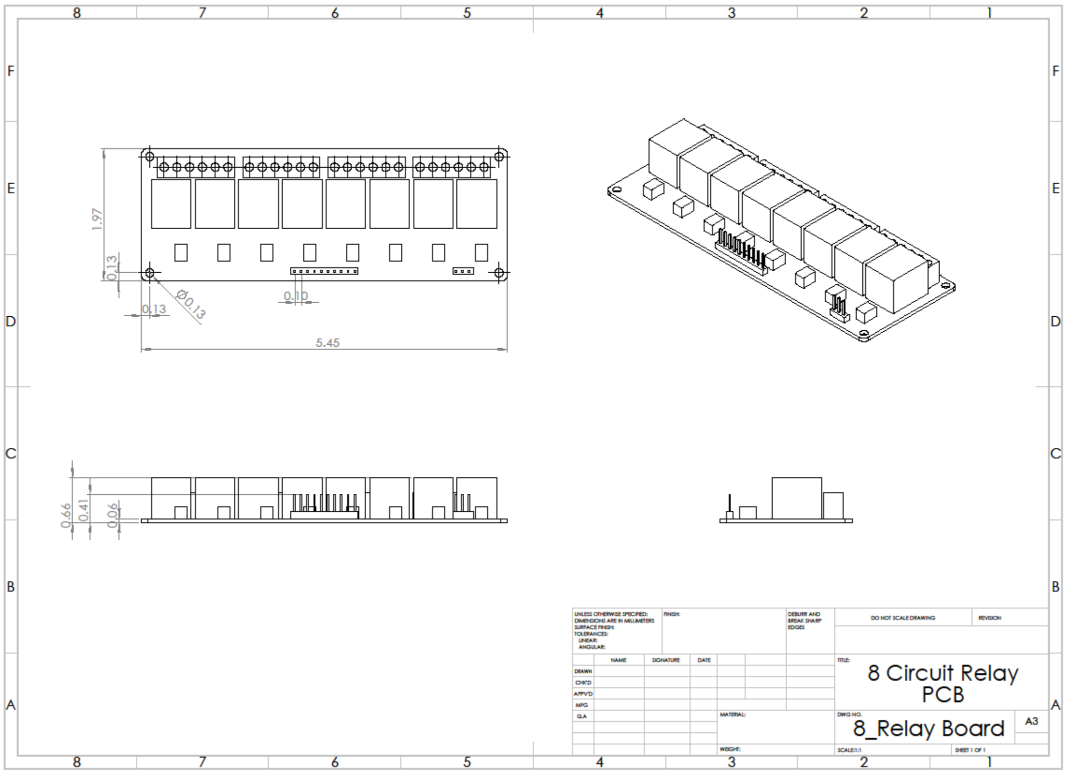
C.21 Carousel



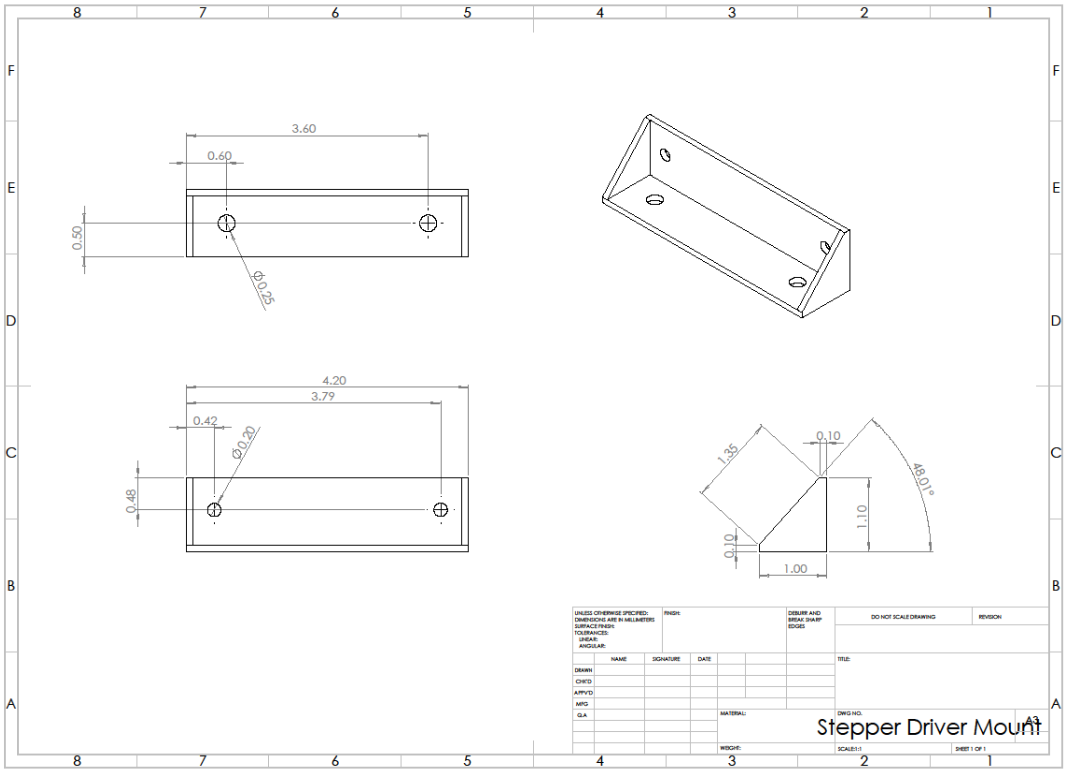
C.22 Carousel Support Platform



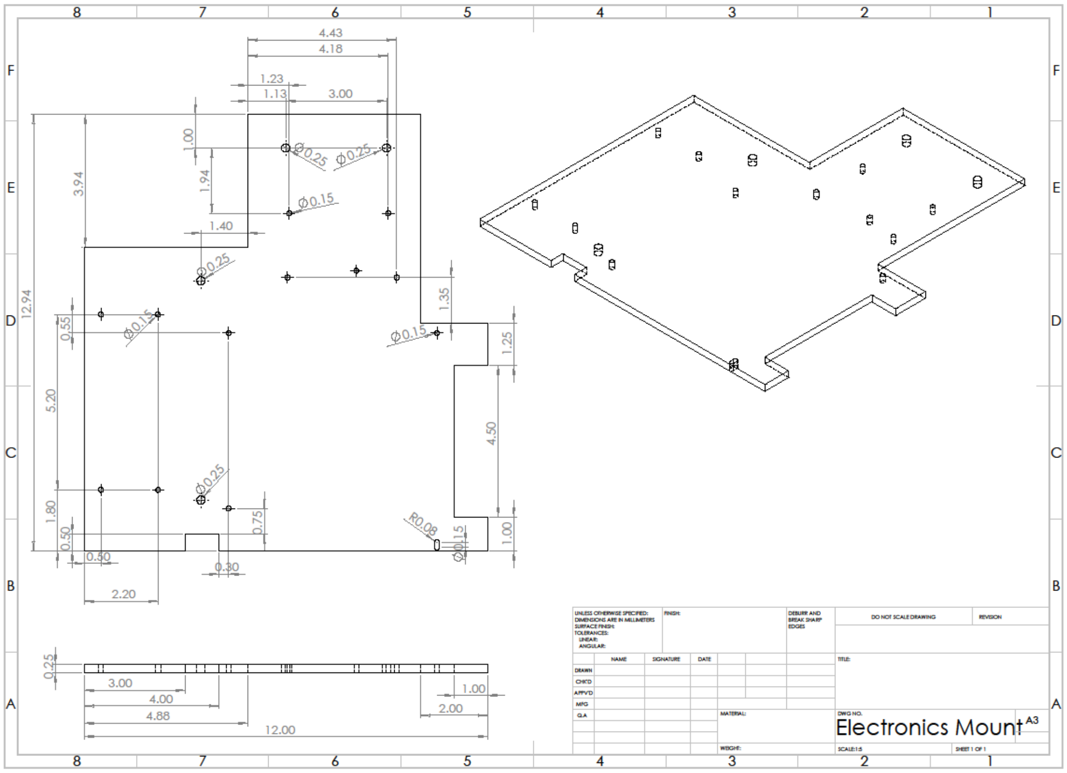
C.23 Eight Relay Board



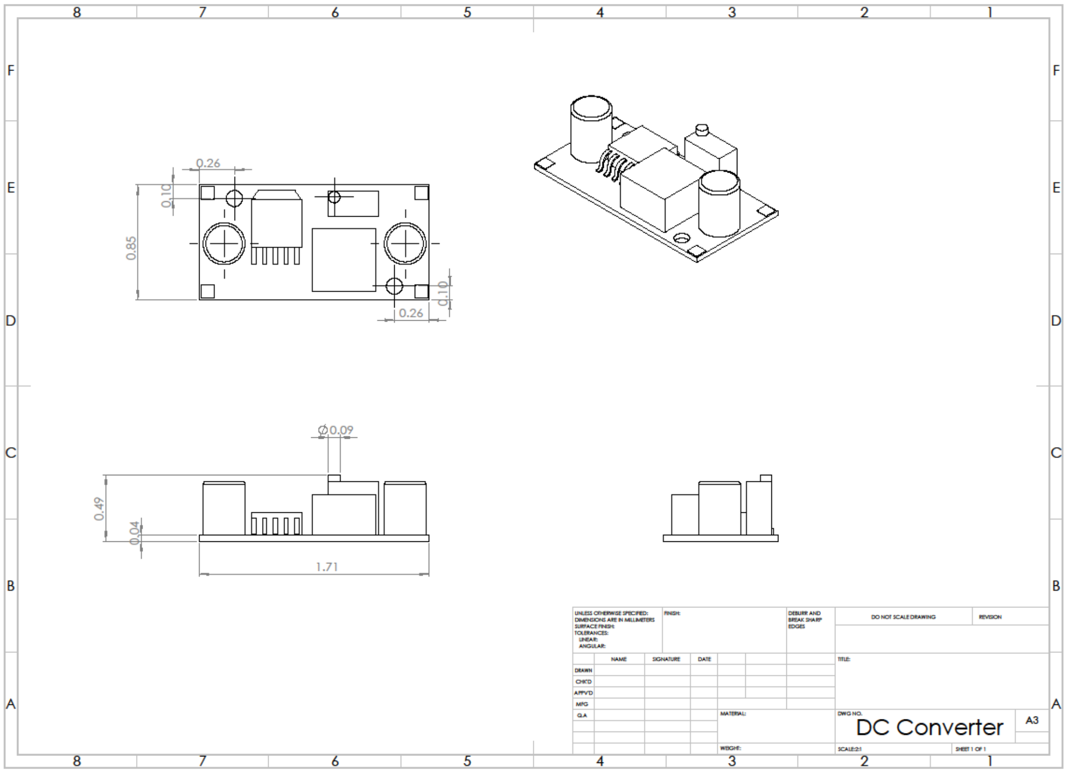
C.24 Stepper Driver Mount



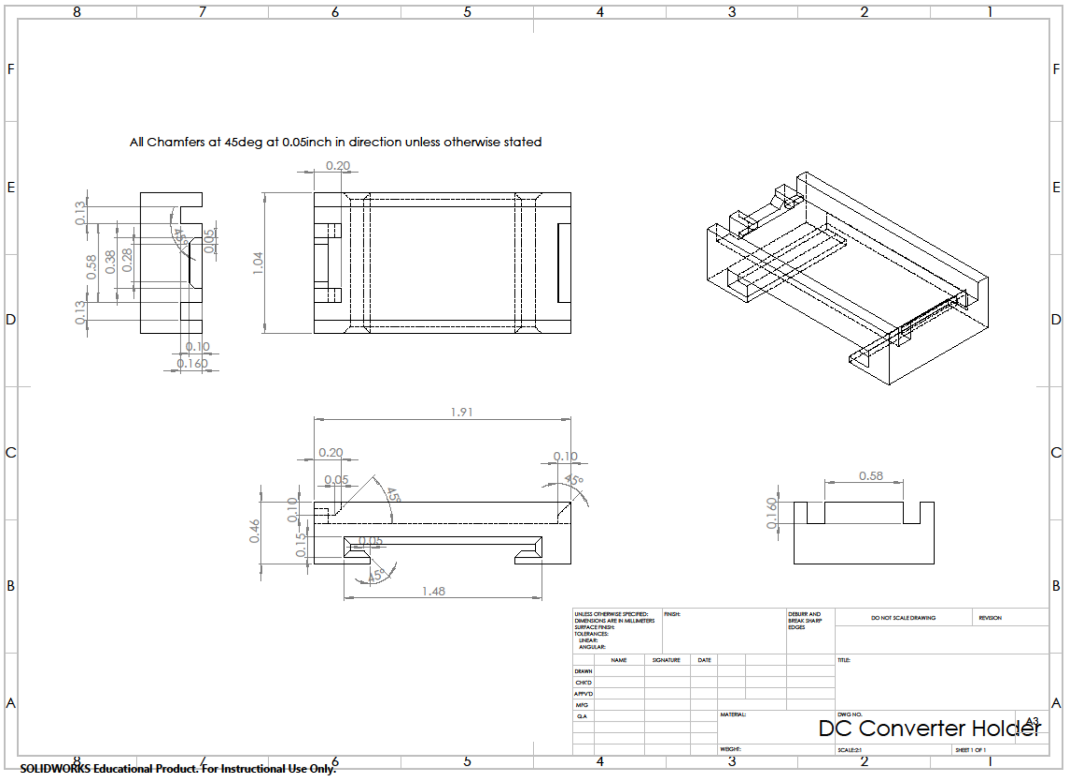
C.25 Electronics Mount



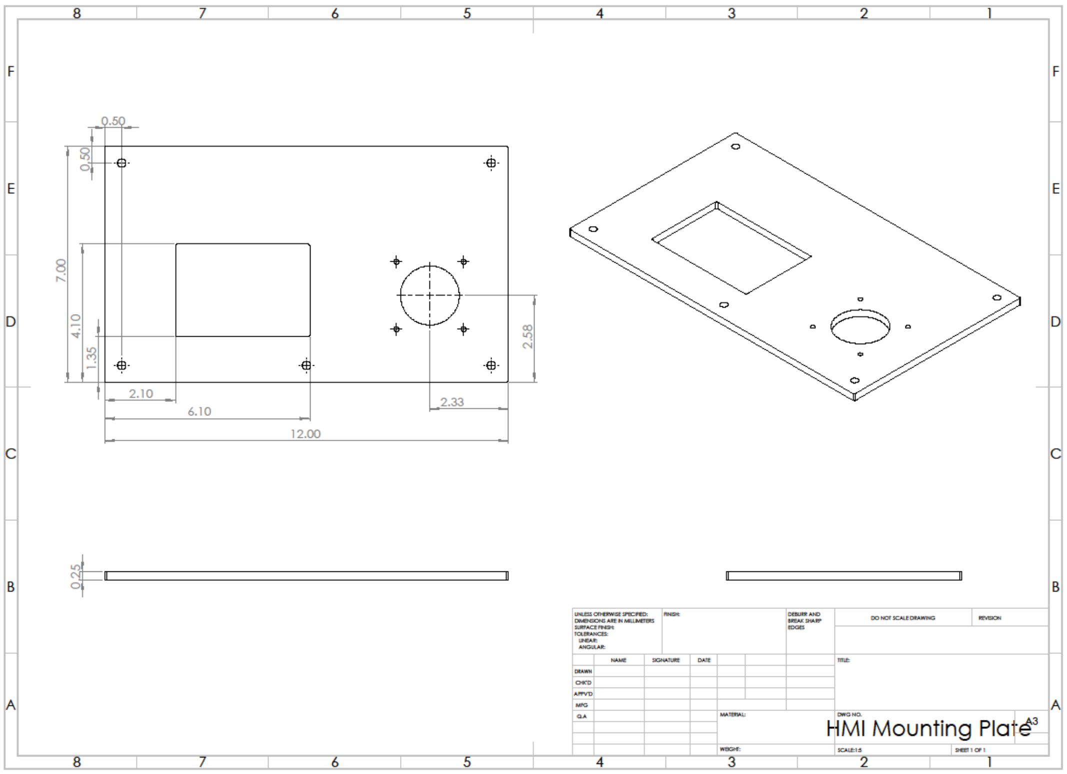
C.26 DC Converter



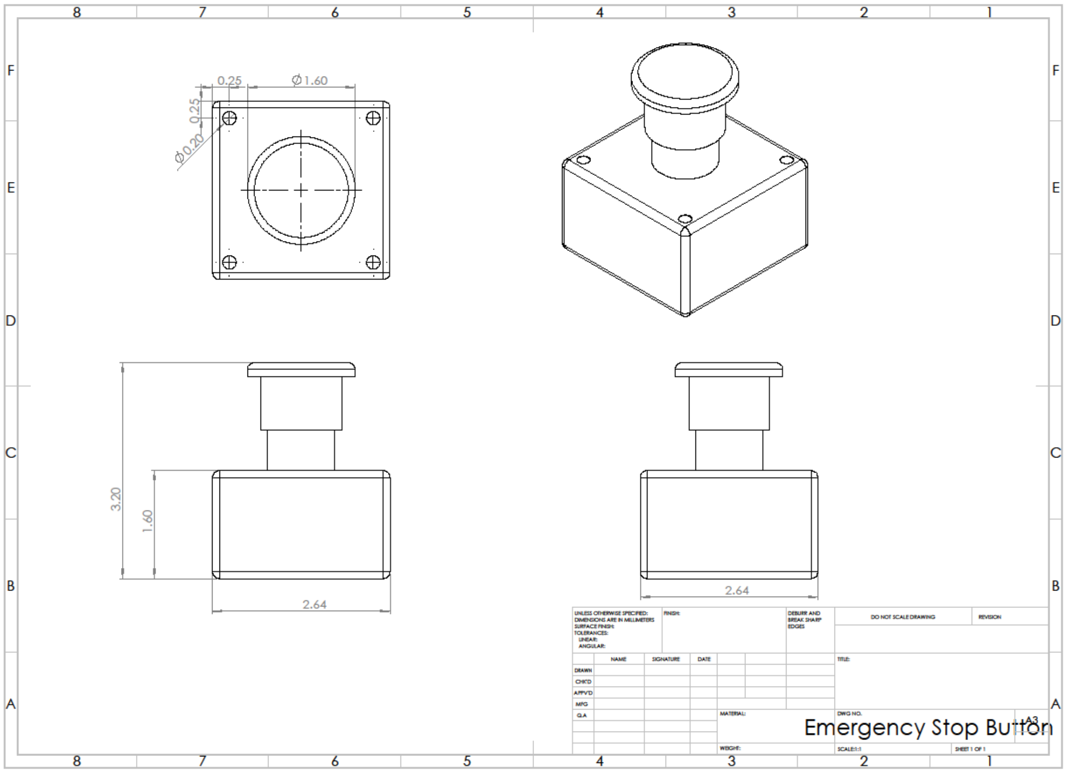
C.27 DC Converter Holder


All Chamfers at 45deg at 0.05inch in direction unless otherwise stated
0.13
0.58
0.38
0.28
0.13
0.05
0.10
0.160
0.20
1.04
1.91
0.20
0.05
45°
0.10
45°
0.46
0.10
0.15
0.05
45°
1.48
0.58
0.160
DC Converter Holder
A3
SOLIDWORKS Educational Product. For Instructional Use Only.

C.28 HMI Mounting Plate



C.29 Emergency Stop Button



APPENDIX D: EMBEDDED SYSTEMS AND PLC CODE

D.1 Arduino Pseudo Code

```
/*Initialize Variables*/
    myservo     /*object to control servo*/
    toggle      /*variable to toggle servo*/
    i           /*variable to count carousel stepper pulses*/
/*Initialize digital Pins*/
    Pin 9       /*attached servo object*/
    Pin 9       /*set servo to non-conforming position*/
    Pin 2       /*stepper decision input with pullup*/
    Pin 3       /*servo decision input with pullup*/
    Pin 3       /*set pin as interrupt and initialize as rising*/
    Pin 5       /*ENA- output to stepper motor driver*/
    Pin 6       /*DIR- output to stepper motor driver*/
    Pin 7       /*PUL- output to stepper motor driver*/
--Loop:
    Reset Variable i
    If Pin 2 goes high
        While i less than 320
            Toggle pin 7 with 2ms delay
            Increment i
--Interrupt
    Set toggle to Pin 3 value
    If toggle is high
        Turn servo for conforming peg
        Set interrupt for falling edge
    If toggle is low
        Turn servo for non-conforming peg
        Set interrupt for rising edge
```

D.2 Arduino IDE Code

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo
int toggle = 0; //create varaible to toggle servo
int i = 0;      //Variable to count carousel stepper pulses

void setup() {
    //Serial.begin(9600); //turn on serial communication for debugging
    myservo.attach(9);  // attaches the servo on pin 9 to the servo object
    pinMode(2, INPUT_PULLUP);  // set pin to stepper input with pullup
    pinMode(3, INPUT_PULLUP);  // set pin to servo input with pullup
    pinMode(5, OUTPUT);  //set pin ENA-(ENA) as output
    pinMode(6, OUTPUT);  //set pin DIR-(DIR) as output
    pinMode(7, OUTPUT);  //set pin PUL-(PUL) as output
    digitalWrite(5, HIGH);  //turn on stepper driver
    digitalWrite(6, HIGH);  //set stepper driver direction

   attachInterrupt(digitalPinToInterrupt(3), pin3_ISR, RISING);  //wait for conforming peg
   myservo.write(120);   //sets the servo for non-conforming
}
void loop() {
  i = 0;
  if(digitalRead(2) == HIGH){

    //rotate carousel to next bin 72deg
    while(i < 320){
      digitalWrite(7, HIGH); //move stepper one pulse
      delay(2);
      digitalWrite(7, LOW); //reset stepper pulse pin
      delay(2);
      i++;
    }//end while
    while(digitalRead(2) == HIGH){
      //wait until pin goes low again
    }//end while

  }//end if
}//end main

void pin3_ISR() {
  toggle = digitalRead(3);
  //Serial.print(toggle);  //print servo decision to serial monitor
  if(toggle == 1){
    myservo.write(70); // sets the servo position according to the scaled value
    //toggle = 1;
    attachInterrupt(digitalPinToInterrupt(3), pin3_ISR, FALLING);
  }
  else{
    myservo.write(120);  // sets the servo position according to the scaled value
    //toggle = 0;
    attachInterrupt(digitalPinToInterrupt(3), pin3_ISR, RISING);
  }
}//end interrupt
```

D.3 Basler Provided Sample Code

```
/* openCVGrab: A sample program showing to convert Pylon images to opencv MAT.

	Copyright 2017 Matthew Breit <matt.breit@gmail.com>

	Licensed under the Apache License, Version 2.0 (the "License");
	you may not use this file except in compliance with the License.
	You may obtain a copy of the License at

	http://www.apache.org/licenses/LICENSE-2.0

	Unless required by applicable law or agreed to in writing, software
	distributed under the License is distributed on an "AS IS" BASIS,
	WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
	See the License for the specific language governing permissions and
	limitations under the License.

	THIS SOFTWARE REQUIRES ADDITIONAL SOFTWARE (IE: LIBRARIES)
IN ORDER TO COMPILE
	INTO BINARY FORM AND TO FUNCTION IN BINARY FORM. ANY SUCH
ADDITIONAL SOFTWARE
	IS OUTSIDE THE SCOPE OF THIS LICENSE.
*/
// Include files to use OpenCV API.
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

// Include files to use the PYLON API.
#include <pylon/PylonIncludes.h>
```

```
// Use sstream to create image names including integer
#include <sstream>

// Namespace for using pylon objects.
using namespace Pylon;

// Namespace for using GenApi objects
using namespace GenApi;

// Namespace for using opencv objects.
using namespace cv;

// Namespace for using cout.
using namespace std;

// Number of images to be grabbed.
static const uint32_t c_countOfImagesToGrab = 1000;

int main(int argc, char* argv[])
{
   // The exit code of the sample application.
   int exitCode = 0;

   // Automagically call PylonInitialize and PylonTerminate to ensure the pylon runtime
system
   // is initialized during the lifetime of this object.
   Pylon::PylonAutoInitTerm autoInitTerm;

   try
   {
```

```
// Create an instant camera object with the camera device found first.
    cout << "Creating Camera..." << endl;
    CInstantCamera camera(CTlFactory::GetInstance().CreateFirstDevice());

// or use a device info object to use a specific camera
    //CDeviceInfo info;
    //info.SetSerialNumber("21694497");
    //CInstantCamera camera( CTlFactory::GetInstance().CreateFirstDevice(info));
    cout << "Camera Created." << endl;
// Print the model name of the camera.
cout << "Using device " << camera.GetDeviceInfo().GetModelName() << endl;

// The parameter MaxNumBuffer can be used to control the count of buffers
// allocated for grabbing. The default value of this parameter is 10.
camera.MaxNumBuffer = 10;

    // create pylon image format converter and pylon image
    CImageFormatConverter formatConverter;
    formatConverter.OutputPixelFormat= PixelType_BGR8packed;
    CPylonImage pylonImage;

    // Create an OpenCV image
    Mat openCvImage;

// Start the grabbing of c_countOfImagesToGrab images.
// The camera device is parameterized with a default configuration which
// sets up free-running continuous acquisition.
camera.StartGrabbing( c_countOfImagesToGrab);

// This smart pointer will receive the grab result data.
CGrabResultPtr ptrGrabResult;
```

```
        // Camera.StopGrabbing() is called automatically by the RetrieveResult() method
        // when c_countOfImagesToGrab images have been retrieved.
        while ( camera.IsGrabbing())
        {
           // Wait for an image and then retrieve it. A timeout of 5000 ms is used.
           camera.RetrieveResult( 5000, ptrGrabResult, TimeoutHandling_ThrowException);

           // Image grabbed successfully?
           if (ptrGrabResult->GrabSucceeded())
           {
              // Access the image data.
              cout << "SizeX: " << ptrGrabResult->GetWidth() << endl;
              cout << "SizeY: " << ptrGrabResult->GetHeight() << endl;
              const uint8_t *pImageBuffer = (uint8_t *) ptrGrabResult->GetBuffer();
              cout << "Gray value of first pixel: " << (uint32_t) pImageBuffer[0] << endl <<
endl;

                     // Convert the grabbed buffer to pylon imag
                     formatConverter.Convert(pylonImage, ptrGrabResult);
                     // Create an OpenCV image out of pylon image
                     openCvImage= cv::Mat(ptrGrabResult->GetHeight(), ptrGrabResult-
>GetWidth(), CV_8UC3, (uint8_t *) pylonImage.GetBuffer());

                     // Create a display window
                     namedWindow( "OpenCV Display Window",
CV_WINDOW_NORMAL);//AUTOSIZE //FREERATIO
                     // Display the current image with opencv
                     imshow( "OpenCV Display Window", openCvImage);
                     // Define a timeout for customer's input in ms.
```

```
                    // '0' means indefinite, i.e. the next image will be displayed after closing the
window
                    // '1' means live stream
                    waitKey(1);

                }
                else
                {
                    cout << "Error: " << ptrGrabResult->GetErrorCode() << " " << ptrGrabResult-
>GetErrorDescription() << endl;
                }
            }
        }
        catch (GenICam::GenericException &e)
        {
            // Error handling.
            cerr << "An exception occurred." << endl
            << e.GetDescription() << endl;
            exitCode = 1;
        }

        // Comment the following two lines to disable waiting on exit.
        cerr << endl << "Press Enter to exit." << endl;
        while( cin.get() != '\n');

        return exitCode;
}
```

D.4 Vision System Pseudo Code

```
*/Initialization*/
Include Files
Include Namespaces

/*Variable Declarations*/
C_counterOfImagesToGrab   //stores captured pylon image
i          //loop counter
status   //comparison decision
dimensionsSwap        //variable for swapping height and width dimensions
heightAngled          //stores height of pegs
widthAngled           //stores width of pegs
scale                 //stores equation to convert pixels to inches
image                 //OpenCV image
camera_t              //USB instant camera

/*GPIO declaration*/
servoControl          //servo control pin to Arduino
stepperControl        //stepper control to Arduino
imageActuation        //Camera Fiber optic input

/*Function Declarations*/
actuationInput()      //waits for camera actuation
imageAnalysis()       //Analyze image and returns peg height
comparison()          //Compares peg height to criteria
stepperServo()        //sends decision to Arduino

--main:
        Set camera parameters
        while{
```

```
        actuationInput()
        Grab image
        Convert to OpenCV image
        imageAnalysis()
        comparison()
        stepperServo()
    }//End while loop
```

--actuationInput:

```
    Wait for GPIO pin to go HIGH
```

--imageAnalysis:

```
    Convert image to black and white
    Apply median filter
    Apply binary threshold
    Find contours
    For{
        Sort images in order by area
        Save number of contours with area over 1000 pixels
    }
    Find length and width of largest contour
    Swap dimensions if width > height
    Convert height from pixels to inches
    Return height
```

--Comparison:

```
    If (height within range){
        Return 1
    }Else{
        Return 2
    }
```

--stepperServo:

```
If(input==1){
    Send high to Arduino
}Else{
    Send low to Arduino
}
```

D.5 Vision System C++ Code

```
//Include files and dependencies to use JETSON GPIO
#include <jetsonGPIO.c>
#include <jetsonGPIO.h>

//Include misc. dependencies
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <termios.h>
#include <time.h>
#include <sys/time.h>
#include <iostream>
#include <string>
#include <unistd.h>

// Include files to use OpenCV API.
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/features2d/features2d.hpp"

// Include files to use the PYLON API.
```

```
#include <pylon/PylonIncludes.h>
#include <pylon/usb/PylonUsbIncludes.h>

// Include sstream to create image names including integer
#include <sstream>

// Namespace for using pylon objects.
using namespace Pylon;

// Settings for using Basler USB cameras.
#include <pylon/usb/BaslerUsbInstantCamera.h>
typedef Pylon::CBaslerUsbInstantCamera Camera_t;
using namespace Basler_UsbCameraParams;

// Namespace for using GenApi objects
using namespace GenApi;

// Namespace for using opencv objects.
using namespace cv;

// Namespace for using cout.
using namespace std;

//Function Declarations
void actuationInput();
double imageAnalysis();
int comparison (double height);
void stepperServo(int status);

//Variable Declaration
static const uint32_t c_countOfImagesToGrab = 3;
```

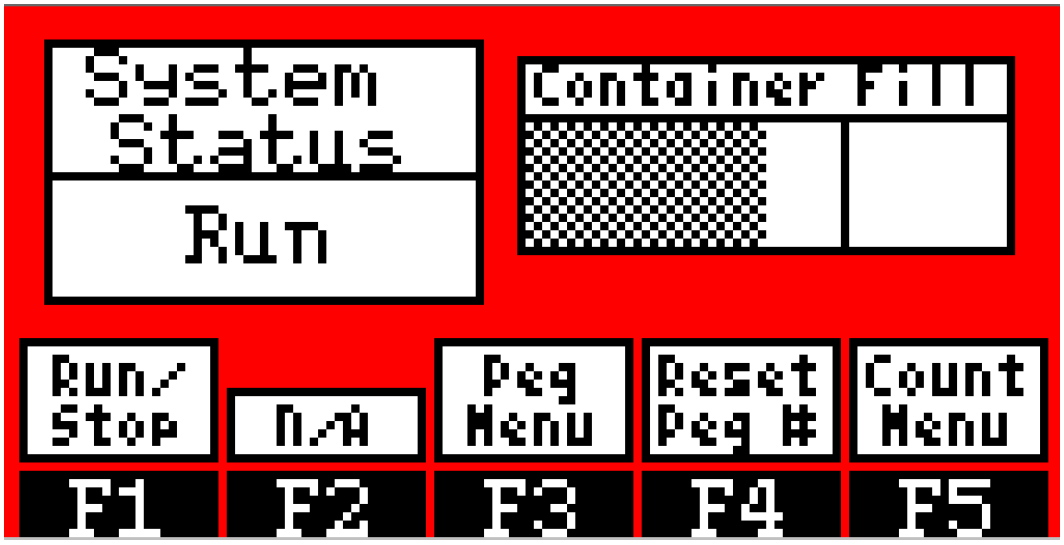
```
int i = 0;
int status = 0;
double dimensionSwap = 0; //initialize variable to swap dimensions
double heightAngled = 0;  //stores height of pegs
double widthAngled = 0;   //stores width of pegs
double scale = 0;         //equation to convert pixels to inches
Mat image;// Create an OpenCV image
jetsonTX1GPIONumber servoControl = gpio219;     // Servo cntrl to Arduino
jetsonTX1GPIONumber stepperControl = gpio186;     // Stepper cntrl to Arduino
jetsonTX1GPIONumber imageActutation = gpio38;    // Camera Fiber optic signal (pulled
up)

typedef Pylon::CBaslerUsbInstantCamera Camera_t;
using namespace Basler_UsbCameraParams;

int main(int argc, char* argv[])
{
        // The exit code of the sample application.
        int exitCode = 0;
        //set up GPIO
        gpioExport(servoControl);
        gpioExport(stepperControl);
        gpioSetDirection(servoControl,outputPin);
        gpioSetDirection(stepperControl,outputPin);
        PylonInitialize();

        namedWindow( "Display window", WINDOW_AUTOSIZE );// Create a window
for display.

        // create pylon image format converter and pylon image
        CImageFormatConverter formatConverter;
        formatConverter.OutputPixelFormat= PixelType_BGR8packed;
        CPylonImage pylonImage;

        // Create an instant camera object with the camera device found first.
        //cout << "Creating Camera..." << endl;
        Camera_t camera( CTlFactory::GetInstance().CreateFirstDevice());
        //cout << "Camera Created." << endl;

        CGrabResultPtr ptrGrabResult; // pointer for receiving grab result data

        INodeMap& nodemap = camera.GetNodeMap();
        //cout << "Collected Node Map" << endl;

        camera.Open();

  //set camera parameters
  // Set the exposure time ranges for luminance control
camera.AutoExposureTimeLowerLimit.SetValue(camera.AutoExposureTimeLowerLimit
.GetMin());

      // Access the enumeration type node GainAuto.
      CEnumerationPtr gainAuto( nodemap.GetNode( "GainAuto"));
      if ( IsWritable( gainAuto))
      {
        gainAuto->FromString("Off");
      }

      // Check to see which Standard Feature Naming Convention (SFNC) is used by the
camera device.
      if ( camera.GetSfncVersion() >= Sfnc_2_0_0)
      {
            // Access the Gain float type node. This node is available for USB camera devices.
            // USB camera devices are compliant to SFNC version 2.0.
            CFloatPtr gain( nodemap.GetNode( "Gain"));
            double newGain = gain->GetMin() + ((gain->GetMax() - gain->GetMin()) / 2);
            gain->SetValue(newGain);
            cout << "Gain (50%)       : " << gain->GetValue() << " (Min: " << gain->GetMin()
<< "; Max: " << gain->GetMax() << ")" << endl;
        }

    camera.AutoTargetBrightness.SetValue(0.2);

//camera.AutoExposureTimeUpperLimit.SetValue(camera.AutoExposureTimeLowerLim
it.GetMax());
camera.AutoExposureTimeUpperLimit.SetValue(10000);//set exposure time limit to
eliminate blur of moving images
    camera.ExposureAuto.SetValue(ExposureAuto_Once);

    // The current configuration is set to software trigger
     camera.RegisterConfiguration( new CSoftwareTriggerConfiguration,
RegistrationMode_ReplaceAll, Cleanup_Delete);

while (i<10000)
 {
                actuationInput();

                camera.StartGrabbing( c_countOfImagesToGrab);
                while( camera.IsGrabbing())
                {
                    camera.ExecuteSoftwareTrigger();
                    camera.RetrieveResult( 5000, ptrGrabResult,
            TimeoutHandling_ThrowException);
```

```
                }//end while

            // Convert the grabbed buffer to pylon image
                    formatConverter.Convert(pylonImage, ptrGrabResult);
                    // Create an OpenCV image out of pylon image
                    image = cv::Mat(ptrGrabResult->GetHeight(), ptrGrabResult->GetWidth(),
CV_8UC3, (uint8_t *) pylonImage.GetBuffer());

                    //imwrite( "Image1.jpg", image );

   double height = imageAnalysis();
   cout<<height<<endl;
   status = comparison(height);
   stepperServo(status);
   usleep(100000);
        i=i+1;
}
   gpioUnexport(servoControl);     // unexport the Servo Pin
   gpioUnexport(stepperControl);      // unexport the Stepper Pin
   gpioUnexport(imageActutation);      // unexport the fiber optic input

   PylonTerminate();
   return exitCode;
}
void actuationInput()
{
        unsigned int value = low;
        gpioExport (imageActutation);
        gpioSetDirection(imageActutation,inputPin);
        usleep(2);
        /*while(value =! low){ //wait for previous peg to clear optics
                usleep(2);
                gpioGetValue(imageActutation, &value);
        }*/

        while(value == low){ //wait for peg to enter optics
                usleep(2);
                gpioGetValue(imageActutation, &value);
        }
        gpioUnexport(imageActutation);
}
double imageAnalysis()
{
        Mat medianFilter;
        Mat bwImage;
        Mat grayImage;
        RotatedRect rectAngled;
         int largest_area = 0;
        int largest_contour_index = 0;
        int pegCount = 0;//counts pegs per image

        cvtColor(image, grayImage, CV_BGR2GRAY);
        //imwrite("GreyImage2.jpg", grayImage);
        medianBlur(grayImage, medianFilter, 1);
        threshold(medianFilter, bwImage, 170, 255, THRESH_BINARY);

        //imshow( "Display window", bwImage );            // Show our image inside it.
        //namedWindow( "Display window2", WINDOW_AUTOSIZE );// Create a
        window for display.
        //imshow( "Display window2", image );            // Show our image inside it.
        waitKey(1000);
        vector<vector<Point> > contours;
        vector<Vec4i> hierarchy;

        // Find contours
        findContours( bwImage, contours, hierarchy, CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE );

        //Sort countours base on size
        for (int contourSize = 0; contourSize<contours.size(); contourSize++){
                double area = contourArea(contours[contourSize],false);
                if(area>1000){
                        pegCount= pegCount+1;
                }
                //save index of largest contour to be used for length
                if(area>largest_area){
                largest_area = area;
                largest_contour_index = contourSize;
                rectAngled=minAreaRect(contours[contourSize]);
        }
  }
//Section to find dimensions using boundingRect function
//Rect rect = boundingRect(contours[0]);
//double heightPixels = rect.height;

//Find dimensions using minAreaRect which accounts for objects at an angle
heightAngled = rectAngled.size.height;//extract height in pixels
widthAngled = rectAngled.size.width;//extract width in pixels

//Switch height and width if they are flipped from .size.width and .size.height
if(heightAngled < widthAngled)
```

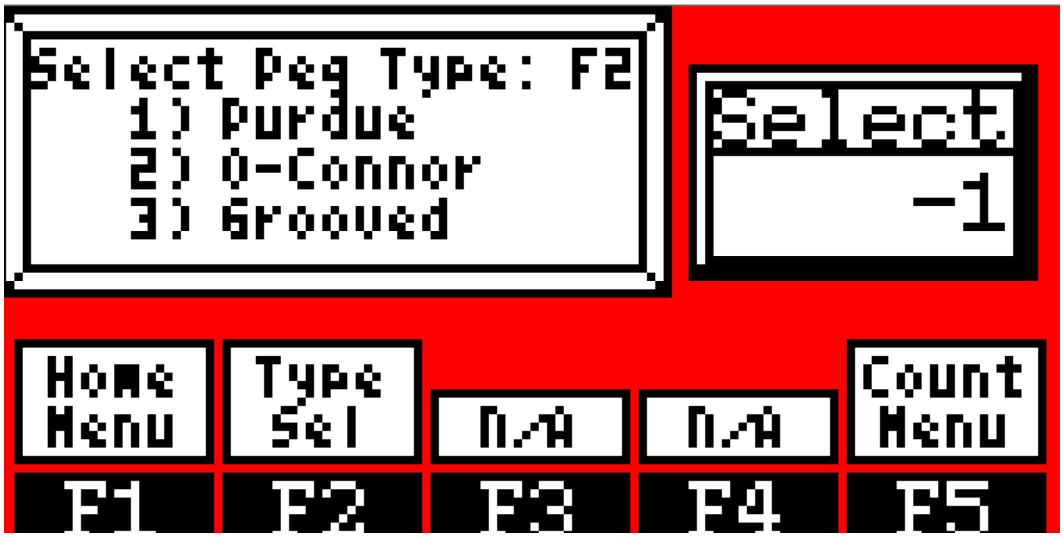
```
        {
                dimensionSwap = heightAngled;
                heightAngled = widthAngled;
                widthAngled = dimensionSwap;
         }

         //calibration equation, changes for each peg type
         double heightInch = heightAngled*0.0023-0.0447;//convert from pixels to inches

         //Display height and width of pegs
         //cout << "Width in pixels: " << widthAngled << endl;
         cout << "Height in pixels: " <<  heightAngled << endl;
         cout << "Height in inches: " << heightInch << endl;

        //contours allowed in each image before rejection, for small and medium pegs, pegCount
        cannot greater than 1
        if(pegCount>2){
                heightInch = 3;
        }
        return heightInch;
}
        int comparison (double height)
        {
                cout << height << endl;
                if (height <= 1.17 && height >= 1.07)
                        {
                                cout << "Conforming" << endl;
                                return 1;
                        }
                else
                {
```

```
                cout << "Non-Conforming" << endl;
                return 0;
        }
}
void stepperServo(int status)
{
        if(status == 1)
        {
                gpioSetValue(servoControl, 1);
                usleep(50);     // off for 50us
                //cout<< "Servo: 1" <<endl;
        }
        if(status == 0)
        {
                gpioSetValue(servoControl, 0);
                usleep(50);     // off for 50us
                //cout<< "Servo: 0" <<endl;
        }
        return;
}
```

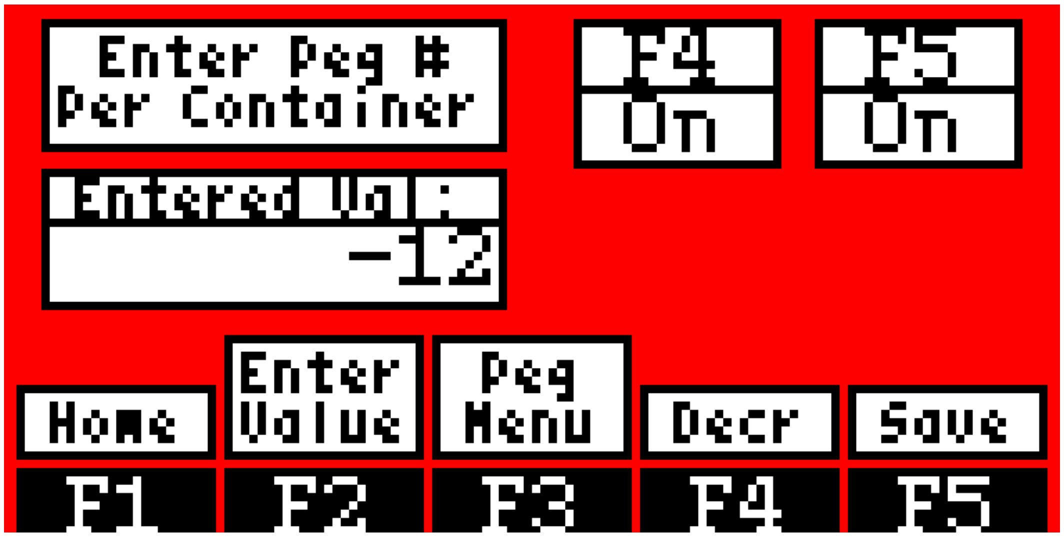
D.6 HMI Screens



(a) Monitor and Navigation Menu

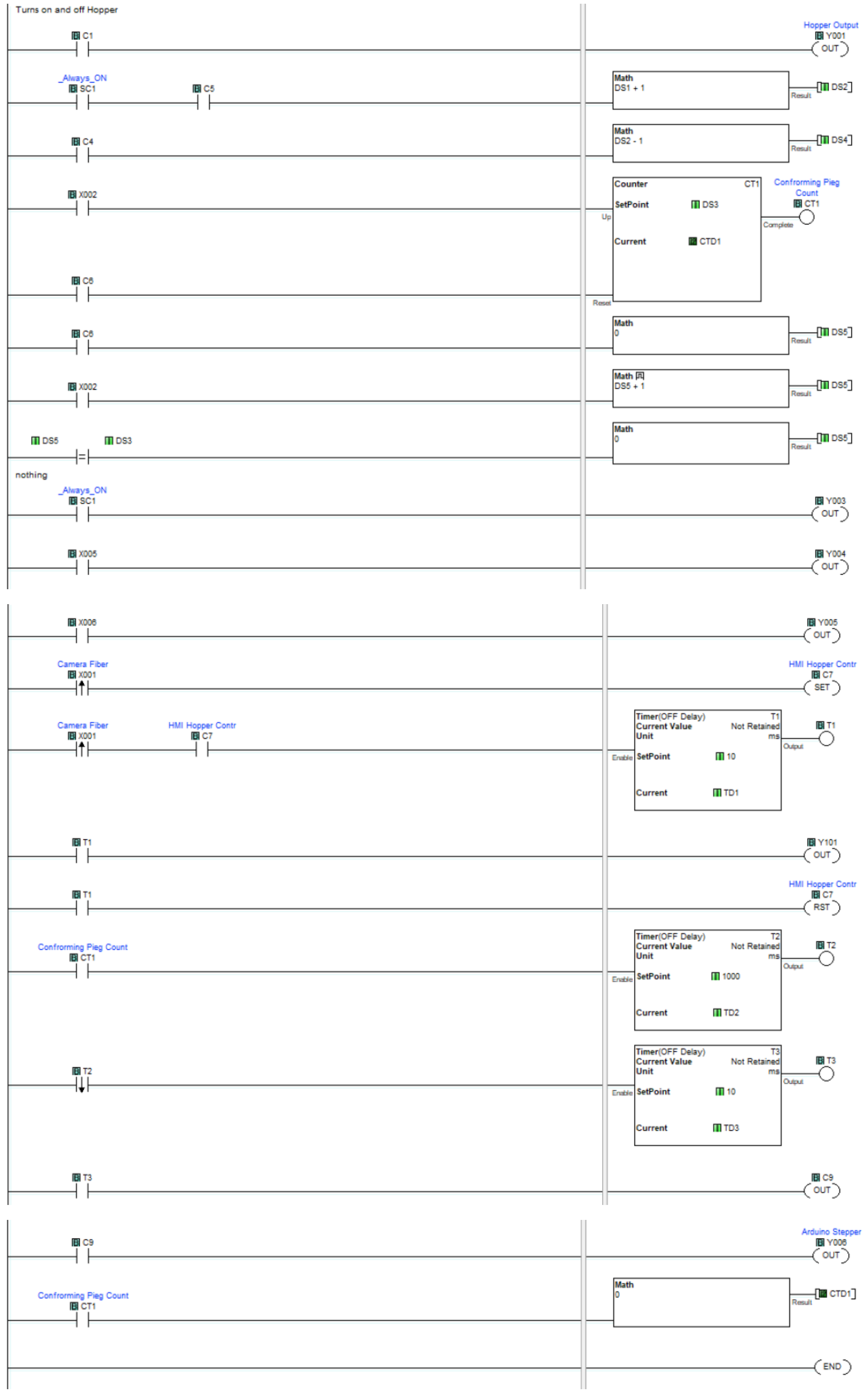


(b) Peg Type Selection Menu



(c) Pegs Count Selection Menu

D.7 PLC Ladder Logic



APPENDIX E: TESTING RESULTS

E.1 Small Peg Testing Data Test 1

Test 1:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	460.300	1.026	1.001	Conforming	-0.025
2	461.339	1.029	1.001	Conforming	-0.028
3	461.281	1.028	1.002	Conforming	-0.026
4	461.932	1.030	1.000	Conforming	-0.030
5	461.281	1.028	1.001	Conforming	-0.027
6	316.941	0.696	0.668	Non-Conforming	-0.028
7	282.963	0.618	0.615	Non-Conforming	-0.003
8	226.546	0.489	0.496	Non-Conforming	0.007
9	359.726	0.795	0.788	Non-Conforming	-0.007
10	402.430	0.893	0.867	Non-Conforming	-0.026
11	456.884	1.018	1.001	Conforming	-0.017
12	464.430	1.036	1.001	Conforming	-0.035
13	460.434	1.027	1.002	Conforming	-0.025
14	461.661	1.029	1.001	Conforming	-0.028
15	463.211	1.033	1.002	Conforming	-0.031
16	403.514	0.896	0.867	Non-Conforming	-0.029
17	263.577	0.574	0.563	Non-Conforming	-0.011
18	386.375	0.856	0.837	Non-Conforming	-0.019
19	328.832	0.724	0.714	Non-Conforming	-0.010
20	230.786	0.498	0.492	Non-Conforming	-0.006

E.2 Small Peg Testing Data Test 2

Test 2:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	460.136	1.026	1.002	Conforming	-0.024
2	459.391	1.024	1.001	Conforming	-0.023
3	462.395	1.031	1.001	Conforming	-0.030
4	462.407	1.031	1.001	Conforming	-0.030
5	460.302	1.026	1.002	Conforming	-0.024
6	359.430	0.794	0.778	Non-Conforming	-0.016
7	314.077	0.690	0.688	Non-Conforming	-0.002
8	398.116	0.883	0.867	Non-Conforming	-0.016
9	399.996	0.887	0.867	Non-Conforming	-0.020
10	233.958	0.506	0.492	Non-Conforming	-0.014
11	463.614	1.034	1.000	Conforming	-0.034
12	458.782	1.023	1.002	Conforming	-0.021
13	462.885	1.032	1.001	Conforming	-0.031
14	461.788	1.030	1.001	Conforming	-0.029
15	460.715	1.027	1.001	Conforming	-0.026
16	384.941	0.853	0.837	Non-Conforming	-0.016
17	232.222	0.502	0.492	Non-Conforming	-0.010
18	324.304	0.713	0.714	Non-Conforming	0.001
19	265.295	0.578	0.563	Non-Conforming	-0.015
20	284.047	0.621	0.615	Non-Conforming	-0.006

E.3 Small Peg Testing Data Test 3

Test 3:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	461.807	1.030	1.001	Conforming	-0.029
2	459.785	1.025	1.003	Conforming	-0.022
3	463.728	1.034	1.001	Conforming	-0.033
4	460.177	1.026	1.001	Conforming	-0.025
5	461.025	1.028	1.001	Conforming	-0.027
6	385.615	0.854	0.837	Non-Conforming	-0.017
7	264.995	0.577	0.563	Non-Conforming	-0.014
8	329.298	0.725	0.714	Non-Conforming	-0.011
9	229.264	0.495	0.492	Non-Conforming	-0.003
10	361.631	0.799	0.788	Non-Conforming	-0.011
11	458.074	1.021	1.000	Conforming	-0.021
12	458.073	1.021	1.001	Conforming	-0.020
13	463.379	1.033	1.001	Conforming	-0.032
14	462.891	1.032	1.001	Conforming	-0.031
15	460.639	1.027	1.001	Conforming	-0.026
16	404.460	0.898	0.867	Non-Conforming	-0.031
17	285.770	0.625	0.615	Non-Conforming	-0.010
18	319.181	0.702	0.688	Non-Conforming	-0.014
19	401.614	0.891	0.867	Non-Conforming	-0.024
20	236.741	0.512	0.496	Non-Conforming	-0.016

E.4 Small Peg Testing Data Test 4

Test 4:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	459.350	1.024	1.000	Conforming	-0.024
2	460.092	1.026	1.002	Conforming	-0.024
3	463.674	1.034	1.001	Conforming	-0.033
4	463.060	1.033	1.001	Conforming	-0.032
5	460.909	1.028	1.001	Conforming	-0.027
6	326.683	0.719	0.714	Non-Conforming	-0.005
7	230.171	0.497	0.492	Non-Conforming	-0.005
8	315.376	0.693	0.688	Non-Conforming	-0.005
9	286.549	0.627	0.615	Non-Conforming	-0.012
10	234.370	0.507	0.496	Non-Conforming	-0.011
11	461.945	1.030	1.002	Conforming	-0.028
12	460.030	1.026	1.003	Conforming	-0.023
13	462.805	1.032	1.000	Conforming	-0.032
14	463.248	1.033	0.999	Conforming	-0.034
15	458.071	1.021	1.001	Conforming	-0.020
16	264.184	0.575	0.563	Non-Conforming	-0.012
17	399.409	0.886	0.867	Non-Conforming	-0.019
18	367.042	0.812	0.788	Non-Conforming	-0.024
19	384.816	0.853	0.837	Non-Conforming	-0.016
20	400.922	0.890	0.867	Non-Conforming	-0.023

E.5 Small Peg Testing Data Test 5

Test 5:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	460.682	1.027	1.000	Conforming	-0.027
2	458.260	1.022	1.002	Conforming	-0.020
3	459.775	1.025	1.001	Conforming	-0.024
4	462.039	1.030	1.002	Conforming	-0.028
5	458.075	1.021	1.000	Conforming	-0.021
6	98.751	0.195	0.563	Non-Conforming	0.368
7	286.942	0.627	0.615	Non-Conforming	-0.012
8	234.134	0.506	0.496	Non-Conforming	-0.010
9	231.561	0.500	0.492	Non-Conforming	-0.008
10	397.016	0.881	0.867	Non-Conforming	-0.014
11	461.373	1.029	1.001	Conforming	-0.028
12	461.343	1.029	1.002	Conforming	-0.027
13	460.435	1.027	1.000	Conforming	-0.027
14	462.919	1.032	0.999	Conforming	-0.033
15	460.975	1.028	1.002	Conforming	-0.026
16	318.219	0.699	0.688	Non-Conforming	-0.011
17	383.738	0.850	0.837	Non-Conforming	-0.013
18	325.672	0.717	0.714	Non-Conforming	-0.003
19	399.859	0.887	0.867	Non-Conforming	-0.020
20	364.897	0.807	0.788	Non-Conforming	-0.019

E.6 Medium Peg Testing Data Test 1

Test 1:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	457.571	1.004	1.005	Conforming	0.001
2	456.586	1.002	1.006	Conforming	0.004
3	457.547	1.004	1.004	Conforming	0.000
4	462.199	1.014	1.004	Conforming	-0.010
5	457.381	1.003	1.005	Conforming	0.002
6	306.382	0.671	0.655	Non-Conforming	-0.016
7	273.605	0.599	0.600	Non-Conforming	0.001
8	220.036	0.481	0.497	Non-Conforming	0.016
9	352.828	0.773	0.773	Non-Conforming	0.000
10	301.865	0.661	0.655	Non-Conforming	-0.006
11	459.723	1.008	1.005	Conforming	-0.003
12	460.281	1.010	1.006	Conforming	-0.004
13	458.439	1.006	0.999	Conforming	-0.007
14	460.205	1.010	1.006	Conforming	-0.004
15	458.394	1.006	1.005	Conforming	-0.001
16	0.000	0.000	0.571	Non-Conforming	0.571
17	247.132	0.541	0.537	Non-Conforming	-0.004
18	414.796	0.910	0.895	Non-Conforming	-0.015
19	347.599	0.762	0.757	Non-Conforming	-0.005
20	398.978	0.875	0.865	Non-Conforming	-0.010

E.7 Medium Peg Testing Data Test 2

Test 2:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	456.420	1.001	1.003	Conforming	0.002
2	461.240	1.012	1.003	Conforming	-0.009
3	460.816	1.011	1.005	Conforming	-0.006
4	457.337	1.003	1.004	Conforming	0.001
5	458.776	1.006	0.999	Conforming	-0.007
6	301.480	0.660	0.655	Non-Conforming	-0.005
7	264.015	0.578	0.571	Non-Conforming	-0.007
8	407.679	0.894	0.895	Non-Conforming	0.001
9	246.292	0.539	0.537	Non-Conforming	-0.002
10	277.529	0.608	0.600	Non-Conforming	-0.008
11	455.040	0.998	1.005	Conforming	0.007
12	457.493	1.004	1.005	Conforming	0.001
13	455.455	0.999	1.003	Conforming	0.004
14	458.670	1.006	1.006	Conforming	0.000
15	458.209	1.005	1.005	Conforming	0.000
16	351.407	0.770	0.757	Non-Conforming	-0.013
17	354.176	0.776	0.773	Non-Conforming	-0.003
18	396.943	0.870	0.865	Non-Conforming	-0.005
19	304.676	0.667	0.655	Non-Conforming	-0.012
20	227.270	0.497	0.497	Non-Conforming	0.000

E.8 Medium Peg Testing Data Test 3

Test 3:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	454.669	0.997	1.006	Conforming	0.009
2	461.086	1.011	1.004	Conforming	-0.007
3	459.045	1.007	1.004	Conforming	-0.003
4	459.332	1.008	1.004	Conforming	-0.004
5	455.954	1.000	1.005	Conforming	0.005
6	264.801	0.580	0.571	Non-Conforming	-0.009
7	410.957	0.901	0.895	Non-Conforming	-0.006
8	211.085	0.461	0.497	Non-Conforming	0.036
9	354.082	0.776	0.773	Non-Conforming	-0.003
10	300.559	0.658	0.655	Non-Conforming	-0.003
11	458.260	1.005	1.003	Conforming	-0.002
12	459.404	1.008	1.004	Conforming	-0.004
13	460.825	1.011	1.006	Conforming	-0.005
14	458.215	1.005	1.003	Conforming	-0.002
15	458.399	1.006	1.003	Conforming	-0.003
16	348.400	0.764	0.757	Non-Conforming	-0.007
17	275.235	0.603	0.600	Non-Conforming	-0.003
18	245.931	0.538	0.537	Non-Conforming	-0.001
19	394.502	0.865	0.865	Non-Conforming	0.000
20	303.349	0.664	0.655	Non-Conforming	-0.009

E.9 Medium Peg Testing Data Test 4

Test 4:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	456.240	1.001	0.999	Conforming	-0.002
2	460.144	1.009	1.003	Conforming	-0.006
3	455.178	0.998	1.004	Conforming	0.006
4	459.812	1.009	1.003	Conforming	-0.006
5	458.938	1.007	1.003	Conforming	-0.004
6	299.813	0.657	0.655	Non-Conforming	-0.002
7	356.723	0.782	0.773	Non-Conforming	-0.009
8	341.743	0.749	0.757	Non-Conforming	0.008
9	411.415	0.902	0.895	Non-Conforming	-0.007
10	274.291	0.601	0.600	Non-Conforming	-0.001
11	456.644	1.002	1.006	Conforming	0.004
12	457.546	1.004	1.006	Conforming	0.002
13	455.266	0.999	1.004	Conforming	0.005
14	460.349	1.010	1.005	Conforming	-0.005
15	461.088	1.011	1.006	Conforming	-0.005
16	304.044	0.666	0.655	Non-Conforming	-0.011
17	264.421	0.579	0.571	Non-Conforming	-0.008
18	247.386	0.541	0.537	Non-Conforming	-0.004
19	225.904	0.494	0.497	Non-Conforming	0.003
20	396.215	0.869	0.865	Non-Conforming	-0.004

E.10 Medium Peg Testing Data Test 5

Test 5:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	462.903	1.015	1.004	Non-Conforming	-0.011
2	457.688	1.004	1.003	Conforming	-0.001
3	456.433	1.001	1.002	Conforming	0.001
4	458.647	1.006	1.002	Conforming	-0.004
5	460.195	1.010	1.003	Conforming	-0.007
6	265.905	0.582	0.571	Non-Conforming	-0.011
7	396.540	0.869	0.865	Non-Conforming	-0.004
8	410.020	0.899	0.895	Non-Conforming	-0.004
9	Missing	Missing	0.757	Non-Conforming	#VALUE!
10	303.935	0.666	0.655	Non-Conforming	-0.011
11	459.169	1.007	1.006	Conforming	-0.001
12	459.831	1.009	1.006	Conforming	-0.003
13	461.405	1.012	1.003	Conforming	-0.009
14	460.304	1.010	1.003	Conforming	-0.007
15	458.838	1.007	1.000	Conforming	-0.007
16	228.059	0.499	0.497	Non-Conforming	-0.002
17	272.227	0.596	0.600	Non-Conforming	0.004
18	248.821	0.545	0.537	Non-Conforming	-0.008
19	302.181	0.662	0.655	Non-Conforming	-0.007
20	356.261	0.781	0.773	Non-Conforming	-0.008

E.11 Keyed Peg Testing Data Test 1

Test 1:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	519.726	1.151	1.122	Conforming	-0.029
2	514.395	1.138	1.119	Conforming	-0.019
3	509.214	1.126	1.119	Conforming	-0.007
4	513.042	1.135	1.120	Conforming	-0.015
5	520.384	1.152	1.124	Conforming	-0.028
6	455.623	1.003	0.975	Non-Conforming	-0.028
7	427.245	0.938	0.943	Non-Conforming	0.005
8	285.732	0.612	0.625	Non-Conforming	0.013
9	367.832	0.801	0.785	Non-Conforming	-0.016
10	346.915	0.753	0.739	Non-Conforming	-0.014
11	519.754	1.151	1.118	Conforming	-0.033
12	520.313	1.152	1.122	Conforming	-0.030
13	507.249	1.122	1.121	Conforming	-0.001
14	519.602	1.150	1.119	Conforming	-0.031
15	520.312	1.152	1.118	Conforming	-0.034
16	375.230	0.818	0.818	Non-Conforming	0.000
17	337.181	0.731	0.723	Non-Conforming	-0.008
18	419.526	0.920	0.926	Non-Conforming	0.006
19	426.117	0.935	0.906	Non-Conforming	-0.029
20	274.980	0.588	0.598	Non-Conforming	0.010

E.12 Keyed Peg Testing Data Test 2

Test 2:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	512.523	1.134	1.121	Conforming	-0.013
2	505.384	1.118	1.119	Conforming	0.001
3	512.000	1.133	1.120	Conforming	-0.013
4	516.470	1.143	1.118	Conforming	-0.025
5	511.761	1.132	1.120	Conforming	-0.012
6	346.282	0.752	0.739	Non-Conforming	-0.013
7	431.914	0.949	0.926	Non-Conforming	-0.023
8	455.268	1.002	0.975	Non-Conforming	-0.027
9	433.572	0.953	0.943	Non-Conforming	-0.010
10	279.489	0.598	0.625	Non-Conforming	0.027
11	505.014	1.117	1.120	Conforming	0.003
12	522.861	1.158	1.121	Conforming	-0.037
13	513.576	1.137	1.119	Conforming	-0.018
14	511.584	1.132	1.120	Conforming	-0.012
15	510.077	1.128	1.115	Conforming	-0.013
16	454.967	1.002	0.975	Non-Conforming	-0.027
17	280.297	0.600	0.598	Non-Conforming	-0.002
18	356.620	0.776	0.785	Non-Conforming	0.009
19	286.687	0.615	0.625	Non-Conforming	0.010
20	430.400	0.945	0.926	Non-Conforming	-0.019

E.13 Keyed Peg Testing Data Test 3

Test 3:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	522.308	1.157	1.119	Conforming	-0.038
2	509.847	1.128	1.118	Conforming	-0.010
3	519.711	1.151	1.120	Conforming	-0.031
4	521.422	1.155	1.118	Conforming	-0.037
5	521.110	1.154	1.120	Conforming	-0.034
6	376.702	0.822	0.818	Non-Conforming	-0.004
7	425.264	0.933	0.906	Non-Conforming	-0.027
8	449.846	0.990	0.975	Non-Conforming	-0.015
9	427.609	0.939	0.926	Non-Conforming	-0.013
10	431.760	0.948	0.943	Non-Conforming	-0.005
11	519.781	1.151	1.119	Conforming	-0.032
12	521.422	1.155	1.121	Conforming	-0.034
13	513.101	1.135	1.121	Conforming	-0.014
14	513.198	1.136	1.120	Conforming	-0.016
15	512.980	1.135	1.119	Conforming	-0.016
16	354.854	0.771	0.785	Non-Conforming	0.014
17	279.545	0.598	0.598	Non-Conforming	0.000
18	347.066	0.754	0.739	Non-Conforming	-0.015
19	339.679	0.737	0.723	Non-Conforming	-0.014
20	293.280	0.630	0.625	Non-Conforming	-0.005

E.14 Keyed Peg Testing Data Test 4

Test 4:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	513.273	1.136	1.119	Conforming	-0.017
2	521.897	1.156	1.118	Conforming	-0.038
3	521.898	1.156	1.122	Conforming	-0.034
4	520.389	1.152	1.121	Conforming	-0.031
5	513.126	1.135	1.119	Conforming	-0.016
6	276.347	0.591	0.598	Non-Conforming	0.007
7	427.917	0.940	0.926	Non-Conforming	-0.014
8	324.479	0.702	0.723	Non-Conforming	0.021
9	288.805	0.620	0.625	Non-Conforming	0.005
10	519.346	1.150	1.119	Conforming	-0.031
11	512.348	1.134	1.120	Conforming	-0.014
12	421.204	0.924	0.906	Non-Conforming	-0.018
13	521.000	1.154	1.120	Conforming	-0.034
14	511.173	1.131	1.119	Conforming	-0.012
15	520.604	1.153	1.121	Conforming	-0.032
16	369.595	0.805	0.785	Non-Conforming	-0.020
17	456.478	1.005	0.975	Non-Conforming	-0.030
18	347.467	0.754	0.739	Non-Conforming	-0.015
19	381.752	0.833	0.818	Non-Conforming	-0.015
20	439.894	0.967	0.943	Non-Conforming	-0.024

E.15 Keyed Peg Testing Data Test 5

Test 5:					
Peg #	Pixel	Predicted Length (Inches)	Actual Length (Inches)	Decision	Error (Inches)
1	508.545	1.125	1.120	Conforming	-0.005
2	510.739	1.130	1.116	Conforming	-0.014
3	512.203	1.133	1.124	Conforming	-0.009
4	520.247	1.152	1.121	Conforming	-0.031
5	520.294	1.152	1.120	Conforming	-0.032
6	370.127	0.807	0.785	Non-Conforming	-0.022
7	326.212	0.706	0.723	Non-Conforming	0.017
8	283.209	0.607	0.598	Non-Conforming	-0.009
9	420.274	0.922	0.906	Non-Conforming	-0.016
10	426.292	0.936	0.926	Non-Conforming	-0.010
11	515.313	1.141	1.120	Conforming	-0.021
12	515.100	1.140	1.120	Conforming	-0.020
13	509.000	1.126	1.121	Conforming	-0.005
14	515.756	1.142	1.122	Conforming	-0.020
15	522.000	1.156	1.122	Conforming	-0.034
16	378.187	0.825	0.818	Non-Conforming	-0.007
17	293.818	0.631	0.625	Non-Conforming	-0.006
18	343.209	0.745	0.739	Non-Conforming	-0.006
19	456.525	1.005	0.975	Non-Conforming	-0.030
20	440.767	0.969	0.943	Non-Conforming	-0.026

LIST OF REFERENCES

Alper, G. (2011, December 8). Basics of Lens Selection for Machine Vision Cameras. Retrieved March 9, 2018, from http://info.adimec.com/blogposts/bid/73803/basics-of-lens-selection-for-machine-vision-cameras

Basler AG. (2018). Software Download | Basler. Retrieved May 23, 2018, from https://www.baslerweb.com/en/sales-support/downloads/software-downloads/#type=pylonsoftware;version=5.1.0;os=linuxarm;series=baslerpulse;model=all

Collins-Thompson, K. (2018). Regression Evaluation - Module 3: Evaluation. Retrieved November 12, 2018, from https://www.coursera.org/lecture/python-machine-learning/regression-evaluation-iKS4j

Du, H. (2015). Research on carrot automatic sorting machine. Applied Mechanics and Materials, 741, 66-69. doi:https://www.scientific.net/AMM.741.66

Echard, T. (2018). [Verbal Communication].

ElMasry, G., Subero, S., Molto, E., & Blasco, J. (2012). In-line sorting of irregular potatoes by using automated computer-based. Journal of Food Engineering, 112(1-2), 60-68. Retrieved October 30, 2017, from http://www.sciencedirect.com/science/article/pii/S0260877412001690

Fernandez-Robles, L., Azzopardi, G., Alegre, E., & Petkov, N. (2017). Machine-vision-based identification of broken inserts in edge profile milling heads. Robotics and Computer-Integrated Manufacturing, 44, 276-283. Retrieved October 30, 2017, from http://www.sciencedirect.com.ezproxy.lib.purdue.edu/science/article/pii/S0736584515300806

Golnabi, A., & Asadpour, A. (n.d.). Design and application of industrial machine vision systems. Robotics and Computer-Integrated Manufacturing, 23(6), 630-637. Retrieved October 30, 2017, from http://www.sciencedirect.com.ezproxy.lib.purdue.edu/science/article/pii/S0736584507000233

Gravetter, F. J., & Wallnau, L. B. (1991). Essentials of Statistics for the Behavioral Sciences. St. Paul, MN: West Publishing Company.

Haider, S. H., Anton, S. P., & Abdullah, N. H. (2011). Metal Parts Visual Inspection Based on Production Rules. Applied Mechanics and Materials, 110-116, 4091-4095. Retrieved October 30, 2017, from https://search-proquest-com.ezproxy.lib.purdue.edu/docview/1443265506?rfr_id=info%3Axri%2Fsid%3Aprimo.

Hou, T. (n.d.). International Journal of Production Research. Automated vision system for IC lead inspection, 39(15), 3353-3367. Retrieved October 30, 2017, from http://web.b.ebscohost.com.ezproxy.lib.purdue.edu/ehost/detail/detail?vid=0&sid=4f4094d6-4f92-4c96-8b53-840789ff2e61%40sessionmgr120&bdata=JnNpdGU9ZWhvc3QtbGl2ZQ%3d%3d#AN=6532461&db=bth

Jarimopas, B., & Jaisin, N. (2008). An experimental machine vision system for sorting sweet tamarind. Journal of Food Engineering, 89(3), 291-297. Retrieved October 30, 2017, from https://www.sciencedirect.com/science/article/pii/S0260877408002173

JetsonHacks. (2015, September 17). Jetsonhacks/jetsonGPIO. Retrieved from https://github.com/jetsonhacks/jetsonGPIO

Kahn Academy. (n.d.). The idea of spread and standard deviation. Retrieved November 12, 2018, from https://www.khanacademy.org/math/probability/data-distributions-a1/summarizing-spread-distributions/a/introduction-to-standard-deviation

Li-na, T., & Li-jun, Z. (2012, August 28). Research of ceramic substrate automatic sorting system. Retrieved October 20, 2017, from http://ieeexplore.ieee.org/document/6324555/

Machine Vision Store. (2018). Machine Vision Store. Retrieved March 3, 2018, from https://machinevisionstore.com/Design/ExEntocentricCalculator

Marques, O. (2011). Practical image and video processing using MATLAB. Hoboken: Wiley-IEEE Press. (Marques, 2011)

Metal Extrusion. (n.d.). Retrieved February 12, 2018, from http://thelibraryofmanufacturing.com/extrusion.html

Neff Automation. (n.d.). NEFF | Industrial Automation Distributor. Retrieved February 12, 2018, from https://neffautomation.com/

NVIDIA Corporation. (2018). JetPack 3.2.1 Release Notes. Retrieved April 15, 2018, from https://developer.nvidia.com/embedded/jetpack-3_2_1

Office 365. Excel [Computer software]. (2016). Retrieved from https://products.office.com/en-us/excel

OpenCV and Pylon [E-mail to the author]. (2018, July 18).

OpenCV Dev Team. (n.d.). Imgproc. Image Processing¶. Retrieved from https://docs.opencv.org/2.4/modules/imgproc/doc/imgproc.html

Packages. (n.d.). Retrieved February 12, 2018, from http://www.solidworks.com/sw/products/3d-cad/packages.htm?scid=hp_tab_products_3d

Pearson, T. (2009, July 18). Hardware-based image processing for high-speed inspection of grains. Retrieved October 22, 2017, from http://www.sciencedirect.com/science/article/pii/S0168169909001021 (Pearson, 2009)

Pop, C., Grigorescu, S., & Davidescu, A. (2012). Robot Vision Application for Bearings Identification and Sorting. Applied Mechanics and Materials, 162, 523-530. Retrieved October 30, 2017, from https://www.scientific.net/AMM.162.523.pdf.

Prabuwono, A., Sulaiman, R., Hamdan, A., & Hasniaty, A. (2006, November 17). Development of Intelligent Visual Inspection System (IVIS) for Bottling Machine. Retrieved October 30, 2017, from http://ieeexplore.ieee.org/document/4142152/authors?ctx=authors

Razavi, S., Bostan, A., & Rezaie, M. (2010). IMAGE PROCESSING AND PHYSICO-MECHANICAL PROPERTIES OF BASIL SEED (OCIMUM BASILICUM). . Journal of Food Process Engineering, 33(1), 51-64. Retrieved October 30, 2017, from http://web.b.ebscohost.com/ehost/detail/detail?vid=0&sid=f1dbf262-f403-404c-85a2-c3384194e293%40sessionmgr102&bdata=JnNpdGU9ZWhvc3QtbGl2ZQ%3d%3d#AN=47716052&db=bth

Sofu, M. M., Kayacan, M. C., & Cetisli, B. (2016). Design of an automatic apple sorting system using machine vision. Computers and Electronics in Agriculture, 127, 395-405. Retrieved October 31, 2017, from http://www.sciencedirect.com/science/article/pii/S0168169916304513?via%3Dihub

Tran T., Nguyen T., Nguyen M., Pham T. (2017) A Computer Vision Based Machine for Walnuts Sorting Using Robot Operating System. In: Akagi M., Nguyen TT., Vu DT., Phung TN., Huynh VN. (eds) Advances in Information and Communication Technology. ICTA 2016. Advances in Intelligent Systems and Computing, vol 538. Springer, Cham

Translations for computer vision. (n.d.). Retrieved February 12, 2018, from https://www.definitions.net/definition/computer%20vision

What is Computer-Aided Design (CAD)? - Definition from Techopedia. (n.d.). Retrieved February 12, 2018, from https://www.techopedia.com/definition/2063/computer-aided-design-cad

You, F., & Zhang, Y. (2008). A Mechanical Part Sorting System Based on Computer Vision. Computer Science and Software Engineering. Retrieved October 30, 2017, from http://ieeexplore.ieee.org/abstract/document/4721885/