5-1-2020

SCREW IT: A COMPUTER VISION APPROACH TO IDENTIFY SCREWS AND FASTENERS

Cristian Garces

SCREW IT: A COMPUTER VISION APPROACH TO IDENTIFY SCREWS AND FASTENERS

by

CRISTIAN CHANNING GARCES

Presented to the Faculty of the Honors College of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

HONORS BACHELOR OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2020

ACKNOWLEDGMENTS

First and foremost, I would like to acknowledge my faculty advisor and professor Dr. Christopher McMurrough for his guidance and assistance these last two semesters.

Secondly, I would like to acknowledge my current and former professors: Dr. Janice Carter Tiernan, Dr. Jaehoon Yu, Dr. Darin Brezeale, and Dr. Bahram Khalili for being positive influences and encouraging me to be a better researcher, student, and leader.

Furthermore, I would like to acknowledge my team members Andrew Ridout, Ruben Martinez, Subodh Dhakal, and Muraj Shrestha. They have been a great asset in assisting me throughout my final year and our overall project. I would like to specifically express my gratitude to Ruben for his assistance in developing a lot of the user interface and Andrew in helping with a lot of the computer vision materials.

Additionally, I would like to thank Mary Koone for offering her knowledge and providing tips to achieve my goals for this project.

Finally, I would like to thank my family who have always been supportive and understanding throughout my student career.

May 07, 2020

ABSTRACT

SCREW IT: A COMPUTER VISION APPROACH

TO IDENTIFY SCREWS AND

FASTENERS

Cristian Garces, B.S. Computer Science

The University of Texas at Arlington, 2020

Faculty Mentor: Christopher McMurrough

Screw/fastener identification has been an issue that most technicians/mechanics have faced in their lifetimes. What kind of screw is it? Metric, Imperial, M5 13 mm, or ¾ inch? Without the proper tools (such as a screw gauge, calipers, rulers, etc.), this process may take a significant amount of time to identify screws. What this project aims to achieve is to develop a cheap and simple application that can quickly identify screw types. This is achieved simply by taking a picture of a screw and wait as the software identifies the necessary information without the hassle of doing anything by hand. Once the screw is identified, it will give the user metrics detailing the screw's specifications (thread pitch, width, metric/imperial, etc.). Hopefully, not only will this improve the quality of work of technicians and workers alike but also inspire further development in this practically untouched field.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Overview

Below is an introduction for the entire project, including my teammates work, so the reader can obtain a better understanding. My work, work that will be considered extra to the main project, consists of the computer vision approach to identify screws.

1.1.1 Vision

In hardware stores all over the world there is an extremely costly and time inefficient process: identifying the screw or bolt a customer brings in. If there were a quick and seamless method for identifying a one-off screw, it could potentially save stores, as well as their customers, a tremendous amount of time. Traditionally, people would use a wire/screw gage to determine the proper screw (Figure ABC is listed as a reference). This app concept may extend in further areas other than by hardware stores such as individual home users, mechanics, artisans, technicians, and more.

1.1.2 Mission

The goal of our project is to identify a screw or bolt simply by taking a photo of it with a smartphone through a mobile app we will build or using a prebuilt system that we will develop. The application will be designed as a user-friendly and easy-to-use method for informing the user about the details of a screw, such as thread pitch, type, and size. For the smartphone-based application, we will be utilizing a cloud service, such as Amazon Web Services, to host our deep learning neural network model which will be able to identify the fastener in the photo and quickly display the information for the user. For our prebuilt system, we will be using computer vision techniques to identify the proper screw/fastener.

1.1.3 Background

How many times per day do customers go into a home center or their local hardware store and try to find a screw to match one that they have lost? It can be very frustrating and can take a considerable amount of time, both on the customer's part and the employee who is trying to help them. Additionally, customers in stores often misplace parts in different locations causing even more confusion for future customers and the employees. An even greater issue is that this can potentially cost the store money if an employee is helping the customer find the correct screw and not attending to other customers in the store. Our app would simplify this process by allowing them to determine the correct screw within seconds.

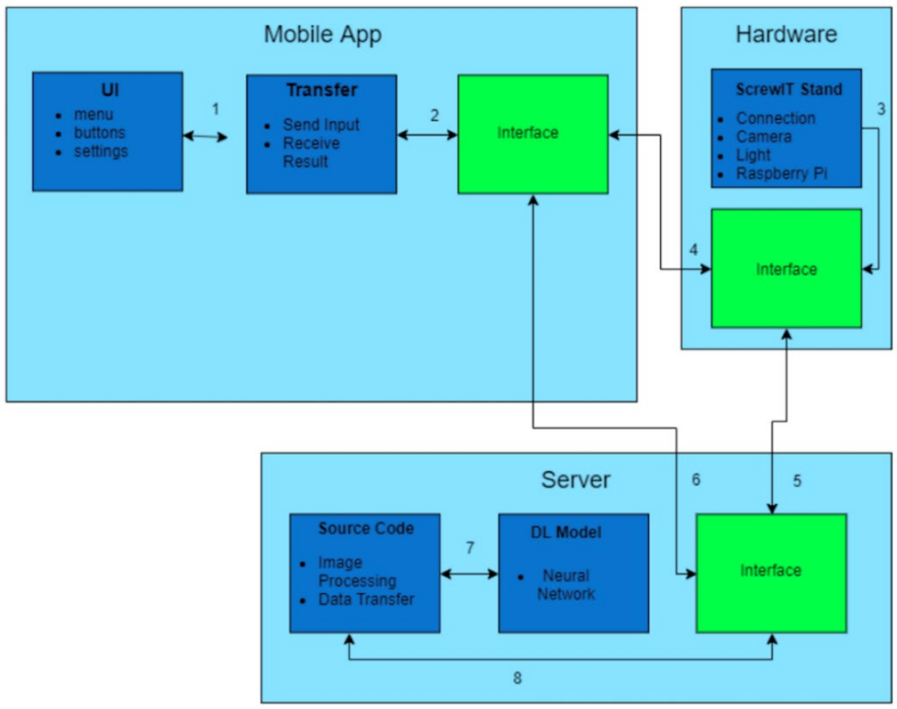


Figure 1.1: Initial Architectural Design

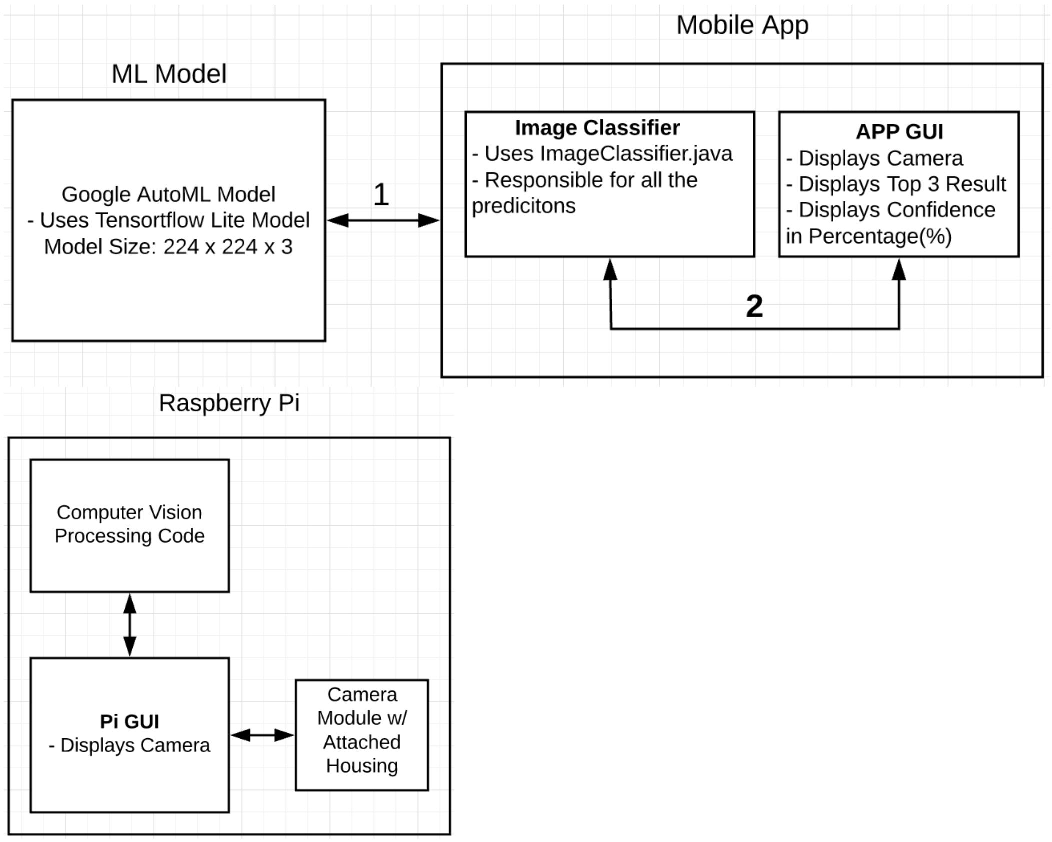

Mobile App
ML Model
Google AutoML Model
- Uses Tensortflow Lite Model
Model Size: 224 x 224 x 3
1
Image Classifier
- Uses ImageClassifier.java
- Responsible for all the predicitons
APP GUI
- Displays Camera
- Displays Top 3 Result
- Displays Confidence in Percentage(%)
2
Raspberry Pi
Computer Vision Processing Code
Pi GUI
- Displays Camera
Camera Module w/ Attached Housing

Figure 1.2: Final Architectural Design

1.1.4 Screw Parameters

The parameters we are interested in include thread pitch, bolt/screw length, thread diameter, and thread angle. These parameters are illustrated in figure 1.3 (apart from the thread angle). For the computer vision method, we will choose to only study the thread pitch.

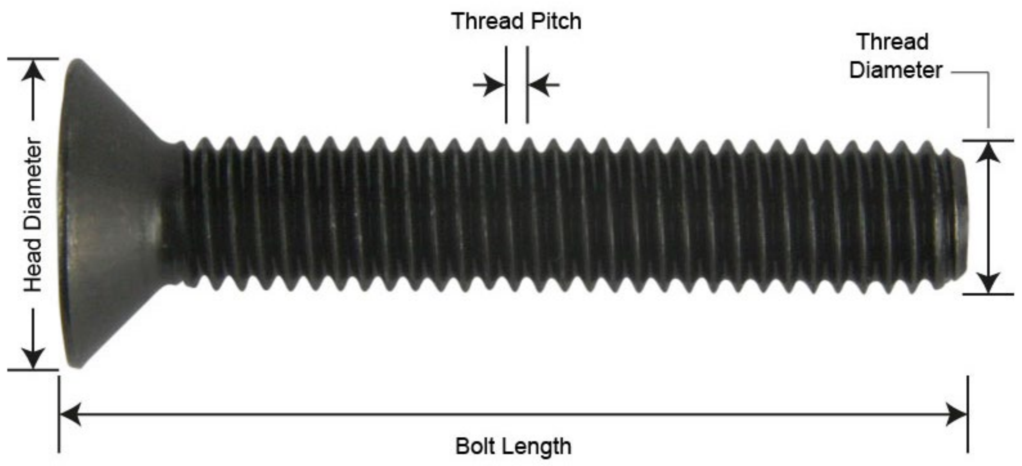


Figure 1.3: Image Detailing the Basic Parameters of a Fastener

1.2 Related Works

The product we are developing is unique and has not been created until now. The goal of our group is to provide efficient, flexible, versatile, simple, and easy to use product that is readily available. Our product mostly focuses and targets the sectors where bolts, screws and/or nuts are used in places such as hardware industries and shops, industrial warehouses, manufacturing and logistics companies, individual workers etc. It is difficult and time consuming for anyone working with bolts and screws to simply identify them by comparing and observing. It may take hours to find the right piece of bolt, and if you are inexperienced in these types of hardware; it may be even worse. On July 2018, Amazon brought a solution to this problem with a feature they added to their existing smart phone application called Part Finder that would allow users to search for screws with pictures from the phone's camera. This piece of software was built and developed by Partpic, a company Amazon acquired in 2016 and claims this feature can identify more than 100 types of fasteners.

However, Amazon's solution will not work for many people for the following reasons:

1. Part Finder is no longer available on the Amazon application
2. Part Finder is not available as its own dedicated application on app stores.
3. If the Amazon app or server goes down, you cannot use this part finder service.
4. It cannot be used on a desktop computer device
5. It had a small data set for a business with tons of resources capable of expanding their data set

Similarly, there are other applications on both Apple's App Store and the Google Play stores such as Fastener Lite, iEngineer, and Nuts and Bolts,who only provide the measurements and a cheat sheet for certain types of screws rather than identifying the screw itself. The main goal of Screw It is to give everyone access to a fast and easy to use identification application. As a side goal, Screw It hopes to offer a dedicated/physical workstation for mechanics and similar demographics to potentially utilize in their shops or other working environments.

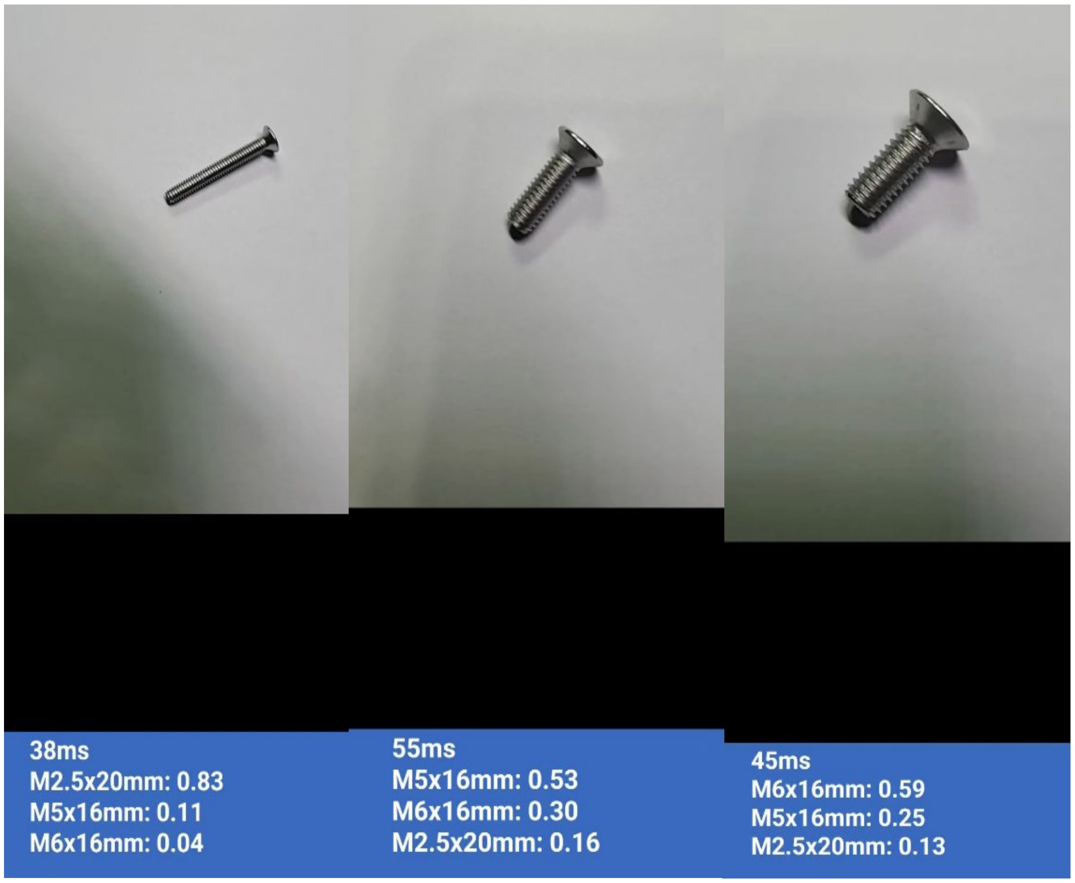
1.3 Approach

As a team, we have decided to keep both deep learning and computer vision as possible options. Therefore, part of the team worked on the deep learning android application and the rest focused on the raspberry pi computer vision application. Even though the team had different goals, we often came together to assist each other when needed.

1.3.1 Deep Learning & Neural Network

Deep learning is a branch of artificial intelligence, specifically a branch of machine learning. “DL is based largely on Artificial Neural Networks (ANNs), a computing paradigm inspired by the functioning of the human brain. Like the human brain, it is composed of many computing cells or ‘neurons’ that each perform a simple operation and interact with each other to make a decision” (O’Mahony, N et al). So, just like how humans use the neurons in their brain to make a decision, computers are able to function similarly but with mathematics applying weight to each choice to allow for the best outcome.

Our model is a 224 x 224 type model, which is the resolution of the images the neural network uses. When we train our model on the Google AutoML Vision, a quantized model is produced which is unsupported by Android. So, we converted the quantized to float model to make it Android compatible. The model accepts 224 x 224 image. The app continuously feeds in the image to the model by converting it to ByteBuffer to be more efficient by avoiding extra copies in computer memory (normal buffers often make copies). The model uses Android GPU, or processing power, to train the model. The image recognition and prediction are done locally as the model is hosted inside the app.



38ms
M2.5x20mm: 0.83
M5x16mm: 0.11
M6x16mm: 0.04

55ms
M5x16mm: 0.53
M6x16mm: 0.30
M2.5x20mm: 0.16

45ms
M6x16mm: 0.59
M5x16mm: 0.25
M2.5x20mm: 0.13

Figure 1.4: Screw It Deep Learning Application

1.3.1.1 Results from Main Project

As shown in figure 1.4, this smartphone application can choose the correct screw types and lengths with relative accuracy. This is after the team had trained the model with a couple of hundred pictures. The only sure way to obtain a better confidence percentage is to feed the model more data.

1.3.2 Computer Vision Success Criteria

The following is the requirements for the computer vision approach for the Screw It version of this application. The intention of this project is to determine which methodology is the best option to pick for future development.

1.3.2.1 The System Shall Identify a Screw at 70% Confidence

Using computer vision techniques (OpenCV), the application will ideally be able to determine the type of screw from M2.5 – M6, excluding M3. In the event of a low percentage (anything lower than 70%), then the goal is to attempt to optimize the developed algorithm as much as possible.

1.3.2.2 There Shall Be a 3D Model to Contain the System Electronics

There is a size limitation to consider as the workstation must be able to print inside the 3D printers on campus. This will not be a main concern due to access to the newer 3D printers and new makerspace in Nedderman Hall located on UTA's main campus.

1.3.2.3 There Shall Be an Interactive Interface for the Customer

Using Tkinter, a graphical user interface (GUI) library, it should output appropriate information to the user controlling the software.

1.4 Project Constraints

It is important to know the constraints of this project. Constraints can help us narrow and confine our area of research but also removes some of our own limitations. The idea is to keep this as simple as possible and avoid any complexities.

1.4.1 Screw Size

We have decided to limit our data set to only include metric screws of size M2.5 - M6 with the option to include M2 and M7 somewhere in the near future. Any larger sizes would force us to create an entirely different workstation. With regards to the larger screws, it is an issue because the workstation must increase in size and buy a lens kit to be able to zoom in/out more.

1.4.2 Screw Length

Although the deep learning version of the project can determine the screw length, the computer vision version is currently unable to. However, ideally, it will be able to determine the proper thread pitch and screw type. This is ideal since the screw name/type is the most important information that a customer or user of this device would require. Any further information would be helpful, but not necessarily needed.

CHAPTER 2

METHODOLOGY

This chapter gives an overview about the experimental setup and the procedure.

2.1 Creating the Workstation

In order to pursue the computer vision route, we must create a workstation to yield the best results. A workstation is necessary so it can remove any parameters from our project that would have been taken into consideration without it. This section will give details regarding the building of the workstation and how its purpose will be used.

2.1.1 The 3D Model

With some assistance from my brother, I modeled a test stand for the purpose of taking optimal pictures. The following parameters must be considered: the lighting, angle, and height position of the camera. The model of the workstation is shown in Figure 2.1.

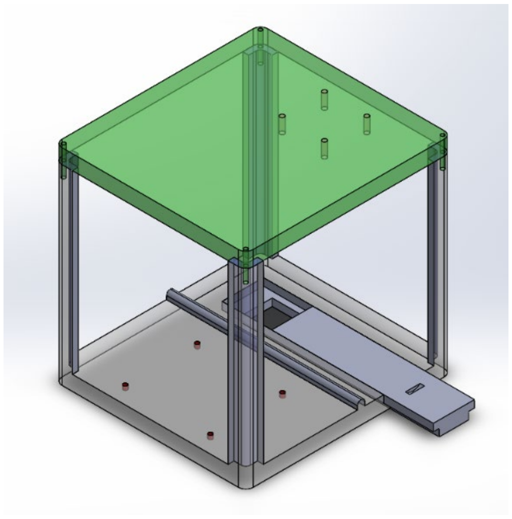


Figure 2.1: Workstation in CAD

Formerly, the model had walls, but a few group members thought it would be best to remove the walls and use available ambient light. However, this would later prove to be a mistake and discussed in a later section. Another aspect of this design is the screw slot; this will assist in holding the screw in place in order to yield consistent results. Additionally, the screw slot is separated to basically be a detachable ‘attachment’ to change out with other attachments. This will allow us to print more slots with some differences (including head size or head type such as hex or flat head – not the screwdriver type), to swap with the existing screw slot to fit different screw heads and sizes without having to reprint a new workstation. However, as mentioned before, we will be constraining ourselves to only studying a specific subset of screws, but it is still a good feature to include. Figure 2.2 gives us a closer view of this part of the model.

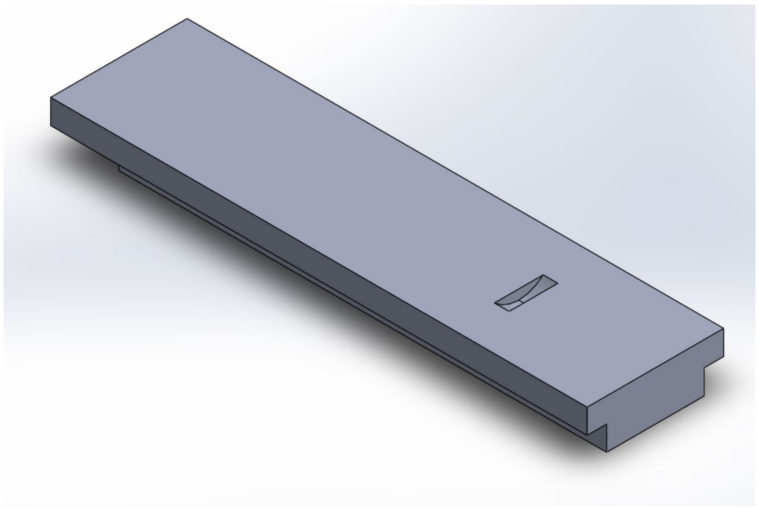


Figure 2.2: Screw Slot

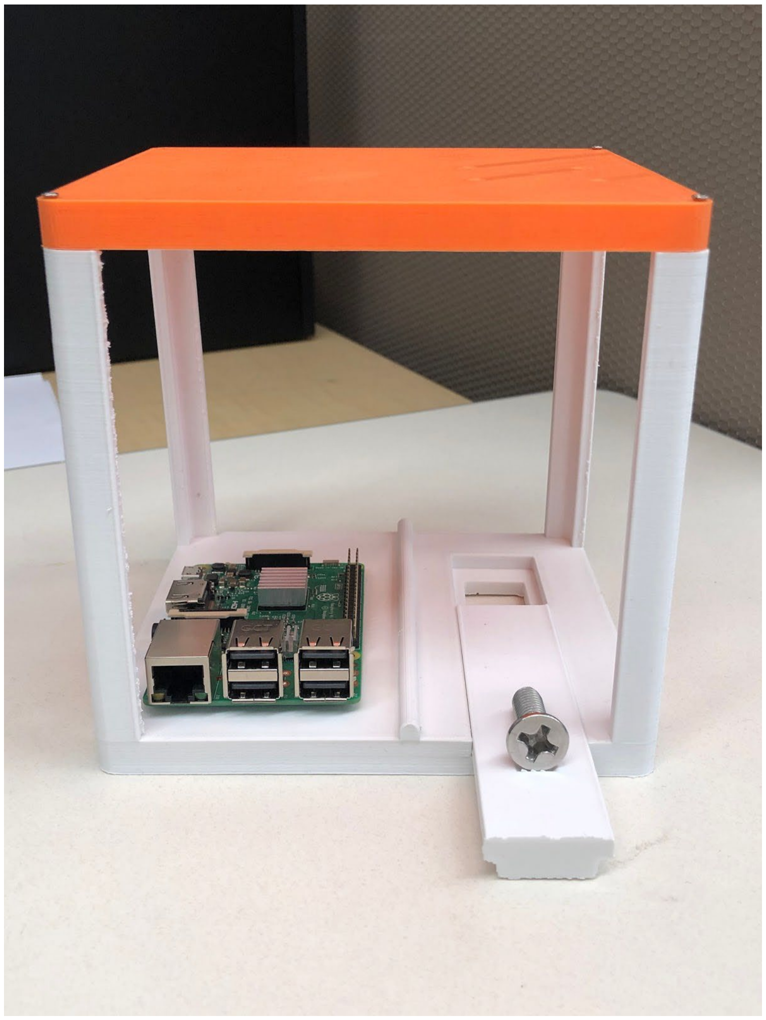


Figure 2.3: Complete 3D Printed Model

2.1.2 Misfortune

Disaster struck soon after the university closed campus due to efforts to slow the spread of COVID-19. The workstation broke into pieces after an unfortunate accident (seen in Figure 2.4). This caused the team to lose the ability to take consistent pictures. To continue, because of this COVID-19 pandemic, the team did not have the proper ability to print a new workstation.

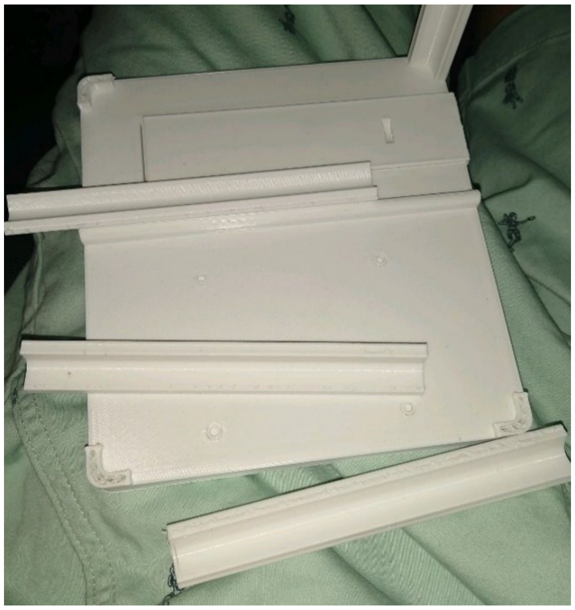


Figure 2.4: Remains of Workstation Stand

The team spent quite some time trying to come up with a solution to this problem. Though, recently, it took an outsider's point of view on the situation to give us an answer. Simply create a makeshift stand using a some left over Styrofoam cups and cut out parts to make the legs, allowing light to come in. To allow the camera to take pictures, cut out a whole for the lens to be inserted into the cup from the top (which is the bottom of the cup). Figure 2.5 shows the temporary/make-shift workstation. The toothpicks serve as 'screws' to lock the camera in position.

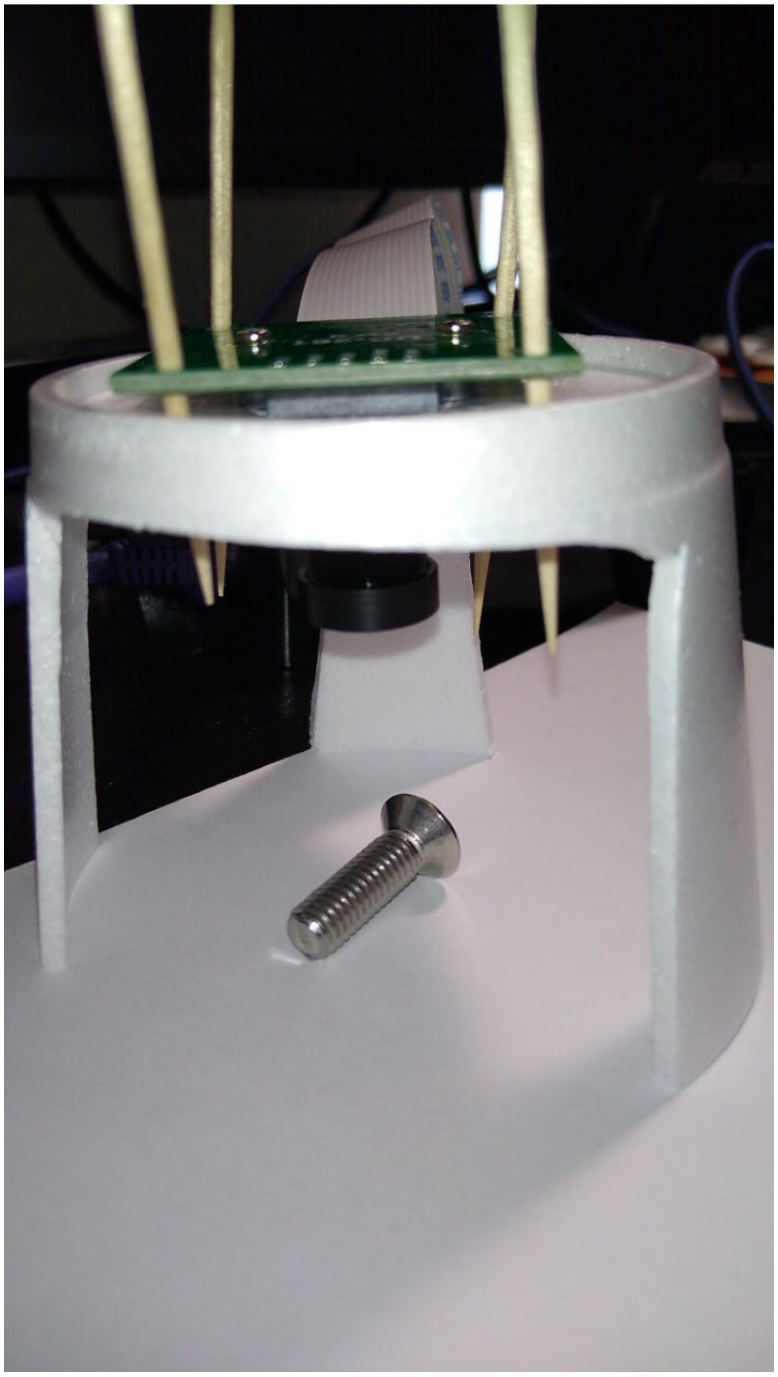


Figure 2.5: Makeshift Workstation

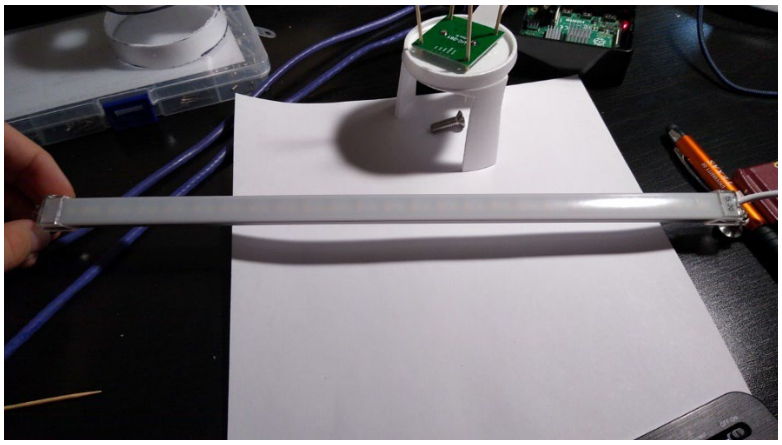


Figure 2.6: Makeshift Workstation with Dedicated Light Source

2.2 Software Development

This section will discuss how the software application on the raspberry pi will work and provide further explanation on commonly used algorithms. As briefly mentioned, the initial design intended for the raspberry pi to communicate directly to a smart phone via Bluetooth within the main mobile application. It was decided this was not the best decision and concluded it was best to isolate the project into two.

2.2.1 Data Processor

The idea behind finding the thread pitch is explained in this section. The Canny edge algorithm determines all possible edges which then goes to the Hough transform to obtain possible lines. We can limit which lines are drawn after indicating the specific angle range of the line. The method of how this is performed is in Appendix A: Source Code Sample. The idea is to first get the degree of the end and start points in radians (a, b), then we can obtain the pair (x0, y0) by multiplying the radians values by the rho values. This is followed by getting the coordinate pair for each point (x1,y1) and (x2,y2) by subtracting by again multiplying by the respective radian values (a, b). In conclusion, we can perform simple trigonometry to get the angle of the line and limit which are drawn on the final image.

Afterwards, we must select the theta value that has the most occurrences by using a histogram-like approach (view figures 2.16 and 2.17). This is shown in Appendix B: More Sample Source Code. Once we obtain the most occurring theta, we must go through the theta's respective rho pair (as originally the Hough line transform yielded [rho, theta]). Then, we can sort the rhos list and thus calculate the running difference between these values. The minimum value of this list/array would be the ideal thread pitch, with a calculated threshold, and is compared to an existing 'database' of known thread pitch values to output the potential to the user.

2.2.1.1 Canny Edge Detection

This algorithm was developed by John F. Canny in 1986 to identify edges/boundaries of objects from images. "The Canny edge detector classifies a pixel as an edge if the gradient magnitude of the pixel is larger than those of pixels at both its sides in the direction of maximum intensity change." (Ding, L., & Goshtasby, A). After the calculations are complete, it outputs all edges found in grayscale (edges are typically in white). It is then passed to the Hough transformation to give us more data by selecting all possible lines. Figure 2.7 illustrates the output of this algorithm.

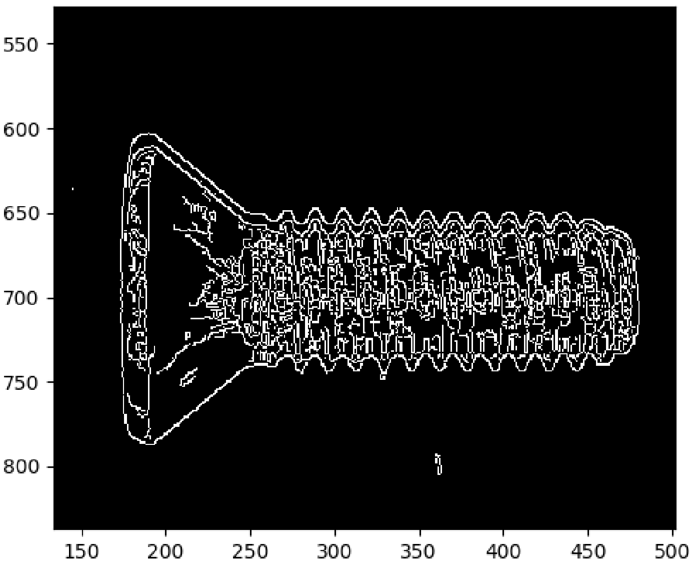


Figure 2.7: Output of Canny Edge from a Screw Image

2.2.1.2 Hough Line Transformation

The Hough transformation is a technique used to detect lines and shapes by using the output from the Canny edge detection algorithm. The following description of Hough transformation is obtained from the official OpenCV documentation: "Any line can be represented in these two terms, (ρ, θ)… create a 2D array or accumulator to hold values of two parameters and it is set to zero initially." Similarly, to a histogram, depending on how accurate you want the data to be, you would need to represent 180 degrees as 180 columns in this array (90 columns for degrees by 2: 0-2, 2-4, etc.). In our project, we have selected 180 for optimal results. For rho values, "the maximum distance possible is the diagonal length of the image." (Bradski, G.). So the rho size is the hypotenuse of 1920 x 1080 where each number represents one pixel (1920 x 1080 is the resolution of the camera). Figure 2.8 illustrates the how a line looks in parametric form of (ρ, θ)

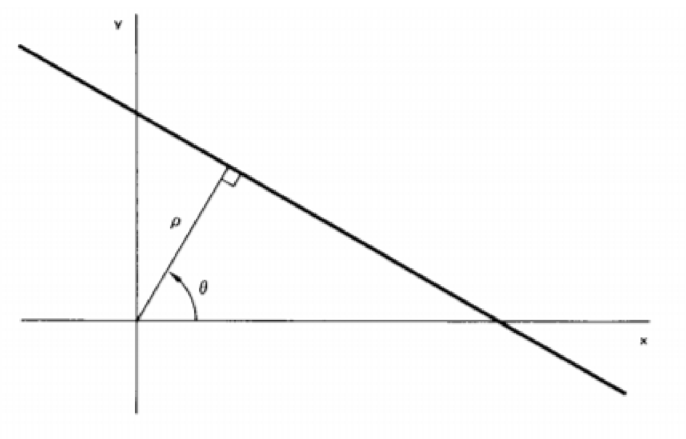


Figure 2.8: A Line in Parametric Form: Rho and Theta

2.2.2 Libraries

Below are the main libraries that are being used to implement the application previously specified.

2.2.2.1 OpenCV

A real time computer vision library that is free for academic and commercial purposes. All the important algorithms are from this library (Canny edge detector, Hough line transformation).

2.2.2.2 Tkinter

This is used to create the Pi based local application interface so the user can interact with and take pictures of screws.

2.2.2.3 Matplotlib

This library is used to generate plots (histograms for this project) mainly to serve as a reference for the developer only. This is to help diagnose bugs and issues with the software.

2.3 Results

2.3.1 Preliminary Results

The first attempts to detect the lines of a screw were insufficient. This could be due to the fact that we had no light source other than ambient light or camera flash. Additionally, our team lacked the understanding on how the Hough transformation worked. The reason being is that computer vision is highly complex field in computer science and none of group members have formerly took the actual class. In fact, most computer vision courses are only taught to graduate students. So, these images were generated with only a basic understanding back in late February.

2.3.1.1 Initial Test Images

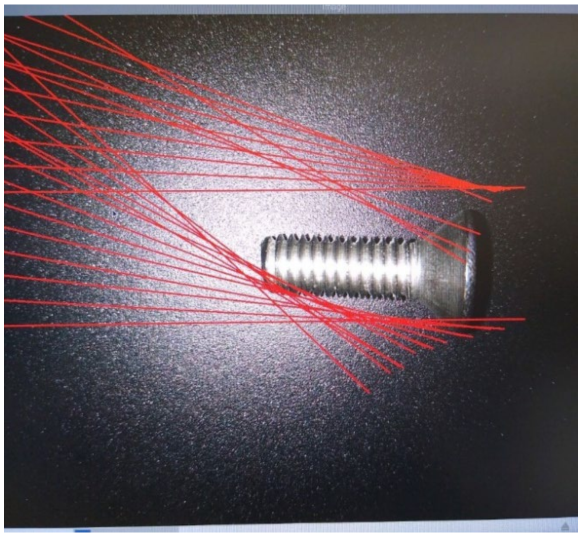


Figure 2.9: Test Image 1

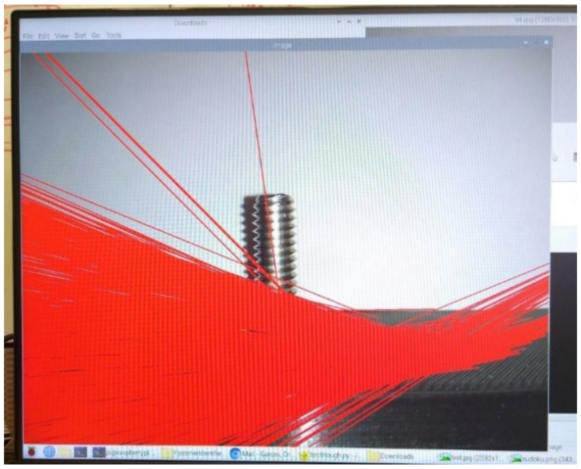


Figure 2.10: Test Image 2

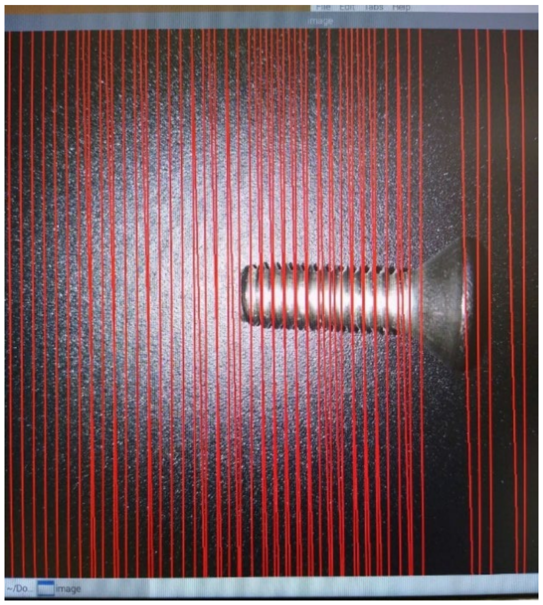


Figure 2.11: Test Image 3

2.3.1.2 Problems

The main and persistent problem is the lighting source. Picture after picture, the lines clearly did not match and were unable to obtain just a few threads of the fastener. Lastly, a minor obstacle was the background of which the screw lied upon caused the line detection algorithms to behave strange. Figure 2.9 and 2.10 illustrates this phenomenon the best. Figure 2.11 was the best out of the initial test images; however, it obtained non-existent lines for almost the entire image.

2.3.2 Final Results

2.3.2.1 New Test Images

With these new test images, one can see how drastically different the results are from using a proper and sufficient light source. Previously the team used ambient room light or a desk lamp. However, as one would have noticed, is that the old lighting choice proved to generate poor images and thus minimized the amount of data points. With a proper and better light source, we are able to compute better results. To compare the old and new light source data, Figures 2.12 and 2.13 only yielded ~300-500 data points Figures 2.14 and 2.15 have yielded ~2000 data points which greatly improves our accuracy

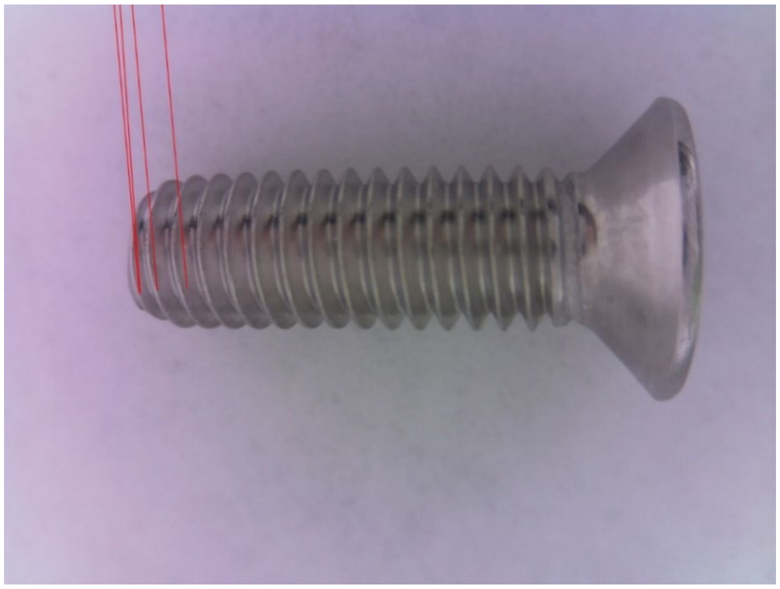


Figure 2.12: Hough Line Transformation with Ambient Light: A

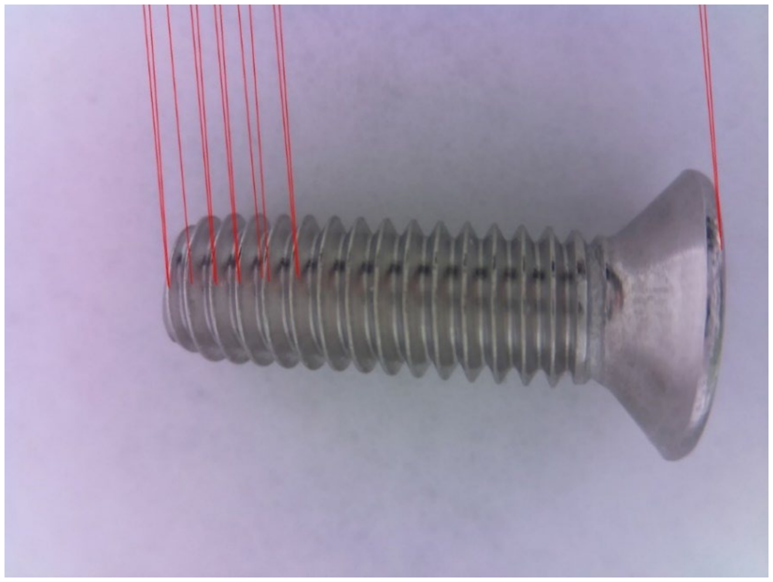


Figure 2.13: Hough Line Transformation with Ambient Light: B

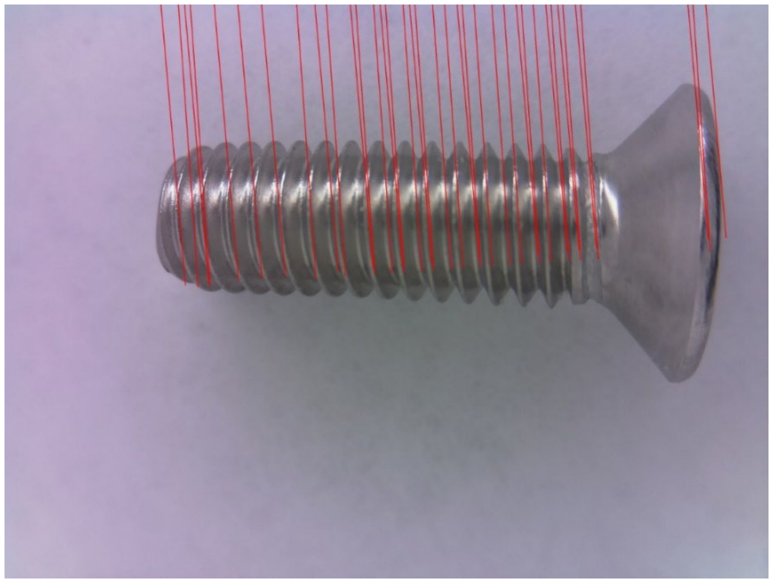


Figure 2.14: Hough Line Transformation with Light Source: A

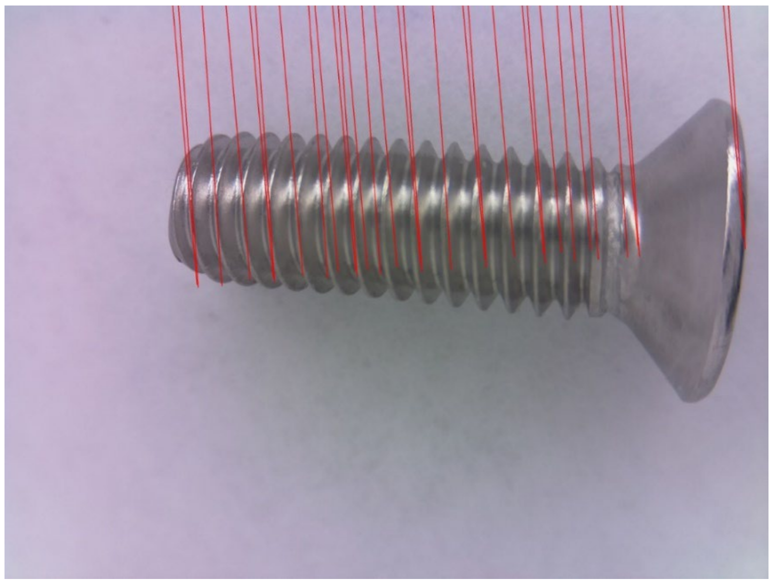


Figure 2.15: Hough Line Transformation with Light Source: B

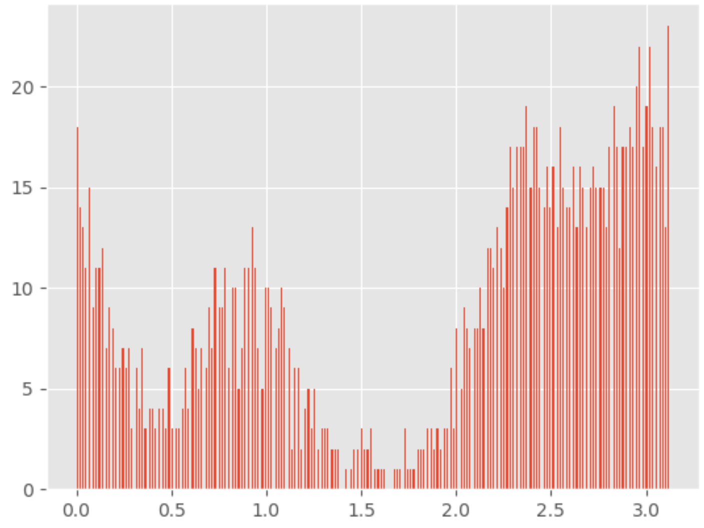


Figure 2.16: Histogram of Theta Values

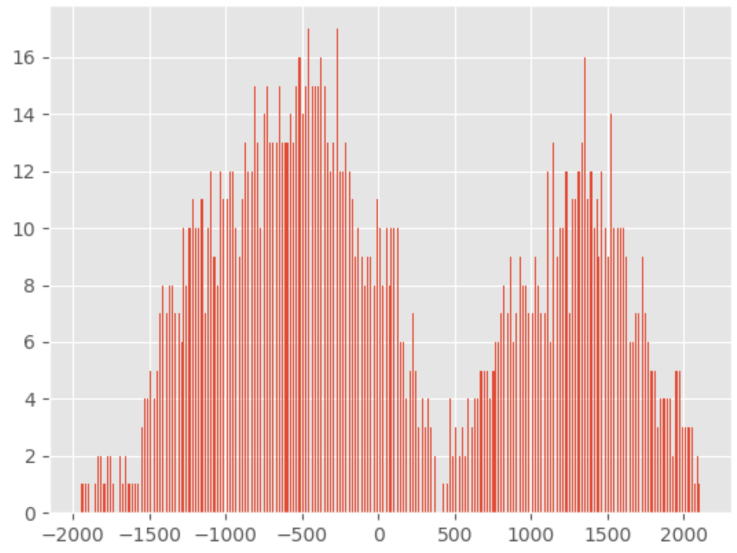
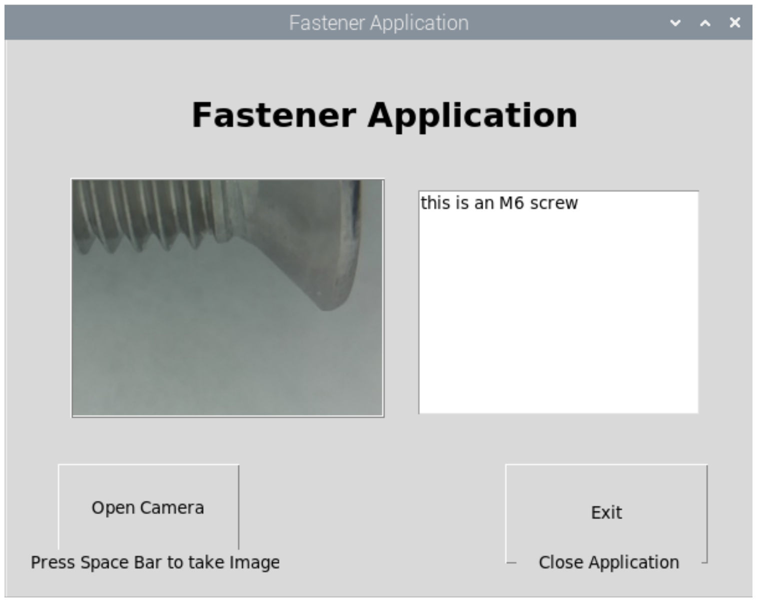


Figure 2.17: Histogram of Rho Values



this is an M6 screw

Figure 2.18: Fastener Application User Interface

2.3.2.2 Problems

A minor issue is the image on the interface is not to scale. Unfortunately, there was not a simple way to resolve this. The position of the screw is also a major concern as the data fluctuates based on the angle of screw.

2.3.2.3 Confidence Rate and Discussion.

As one can see, the overall computer vision project failed to meet its requirement to detect the proper screw with 70% confidence. The highest efficiency was only at 44% as compared to the deep learning model at 53%. Though, not exactly failure, this project still has a lot of promise as there are several aspects that could be optimized, such as using a distribution plot to increase the possibility of choosing a 'more correct' theta value.

The completion of this project was inhibited by the closing of the school, the accidental destruction of the workstation, and the time it took finding suitable lighting for the camera. However, there is still a lot of potential this project has, especially in areas where I could better optimize the code as I now have a much better understanding of the algorithms I am working with. To add onto this, I already have a number of ideas to implement into the project in the future (which will not be discussed at this moment in time).

Table 3.1: Efficiency of Computer Vision vs Deep Learning

	M6	M5	M2.5
Deep Learning Model	59%	53%	83%
Computer Vision	44%	40%	32%

CHAPTER 3

CONCLUSION

To conclude, I do not consider this project to be a failure despite not completing the main requirement. The main goal was to learn which method would be the best option for further development. I believe the best approach to the problem all depends on the time scale (both in terms of identifying a piece and time to develop the software). If the goal were to obtain fast results in a short time frame, then deep learning would be the best option. However, the results may vary as it is all dependent on the size of the data set. For a long-term project, computer vision would probably be the best option as it would yield more consistent results and perhaps become more manageable software-wise. As previously mentioned, deep learning will eventually become the better image processor somewhere along the future.

3.1 Possible Explanation on Low Confidence Rate

One possible explanation of a low confidence rate is due to artificial issues. To explain, I drastically underestimated the intricacies of how the computer vision algorithms worked on a fundamental level, and even OpenCV in general.

An additional reason for the low confidence rate is that our team lost a lot of time due to the pandemic. For upwards to three weeks, the raspberry pi computer (which had all the software) was stuck on campus. Furthermore, my own computers were unable to set up the programming environment which did not allow me to work on the project from home, so development was all dependent if I had physical access to the machine.

3.2 Computer Vision vs Deep Learning

Deep learning has seen more rapid growth and development as compared to computer vision in the last few years. Computing power, accuracy and even cost-effectiveness have all seen improvements. In fact, deep learning has shown in some cases to be more accurate than the traditional computer vision (O'Mahony, N et al). This could also explain the fact that my confidence rate is significantly different as compared to our main projects confidence rate using deep learning. Though it is important note that deep learning methods cannot solve everything a computer vision application can. There are some aspects where it would be better to use computer vision rather than deep learning.

3.3 Lessons Learned

One major lesson I learned is to prototype as early as possible as to avoid rushing at the end of the project. Secondly, I learned it is important to consider all options more carefully, such as: what light source to use, how high should the camera be, or what kind of camera lens should we use. Since the team chose to forego the light source, we wasted time on getting bad results. If we had chosen a proper light source to begin with, we could have gotten much better results at the beginning of this project.

APPENDIX A

SOURCE CODE SAMPLE

```python
img = cv2.imread(fileName)

  #imS = cv2.resize(img, (960, 540))

  gray = cv2.cvtColor(img, 0)#cv2.COLOR_BGR2GRAY
  edges = cv2.Canny(gray, 50, 150, apertureSize=3)

  #edges = cv2.Canny(img, 50, 150, apertureSize=3)
  lines = cv2.HoughLines(edges, 20, np.pi / 180, 200)

  rhos = []
  thetas = []
  #print(lines)
  for line in lines :

      #print(k)
      rho,theta = line[0]
      rhos.append(rho)
      thetas.append(theta)
      a = np.cos(theta)
      b = np.sin(theta)
      x0 = a * rho
      y0 = b * rho
      x1 = int(x0 + 1000 * (-b))
      y1 = int(y0 + 1000 * (a))
      x2 = int(x0 - 1000 * (-b))
      y2 = int(y0 - 1000 * (a))

      deltaX = x2 - x1
      deltaY = y2 - y1

      if (deltaX != 0):
          angle = (np.arctan(deltaY / deltaX) * (180 / np.pi))
      else:
          angle = 90

      # Conditional to include only the vertical (or nearly vertical) lines
      if (angle > 84 and angle < 86):
          cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)
```

APPENDIX B

MORE SAMPLE SOURCE CODE

```python
def selectMostBin(rho_theta, typeFlag):
    # size of the list
    n = len(rho_theta)

    # minimum and maximum values of the list
    min_ = min(rho_theta)
    max_ = max(rho_theta)
    #calculating the range
    range_ = max_ - min_

    #getting the number of required
    numOfIntervals = int(round(math.sqrt(n))) * 10#bins
    widthOfIntervals = (range_)/(numOfIntervals)

    #buckets are essentially the bins in list form
    buckets = np.arange(min_,max_ + widthOfIntervals, widthOfIntervals)

    #this is essentially the frequency (y-axis of histogram) in list form
    bck = np.zeros(numOfIntervals+1)

    # for optimization, use a binary search tree approach
    for x in rho_theta:
        for i in range(len(buckets)-1):
            if x >= buckets[i] and x <= buckets[i+1]:
                bck[i] = bck[i] + 1
                break

    #gets the index of the bin with the highest frequency
    index_max = max(xrange(len(bck)), key=bck.__getitem__)
    #gets the acual bin value of the highest frequency
    min_Bin = buckets[index_max]
    #this is the upper boundary of the bin (min_bin)
    max_Bin = buckets[index_max+1]

    #for all intents and purposes, the actual bin that will be filled is min_Bin

    #if range is 40, add an extra bucket for comparison purposes
    #buckets [ 0, 10, 20, 30, 40, 50]
    #bck     [ 0,  0,  0,  0,  0, 0]
    #input: 0, 1, 15, 3, 22, 20, 14, 10, 39, 40
    #bck =   [ 3, 3, 2, 1, 1, 0]
    #the input value, 15, will fill the 10 bin as it falls in between 10 and 20

    #bin [10-20] or [min_bin - max_bin]
```

REFERENCES

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Ding, L., & Goshtasby, A. (2001). On the Canny edge detector. *Pattern Recognition*, *34*(3), 721–725. doi: 10.1016/s0031-3203(00)00023-6

O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G. V., Krpalkova, L., … Walsh, J. (2019). Deep Learning vs. Traditional Computer Vision. *Advances in Intelligent Systems and Computing Advances in Computer Vision*, 128–144. doi: 10.1007/978-3-030-17795-9_10

BIOGRAPHICAL INFORMATION

Cristian Garces joined The University of Texas at Arlington in the Fall of 2016 majoring in Computer Science and will be graduating with highest honors in Spring 2020. His research interests include particle physics, theoretical computer science, and cybersecurity. He has participated in undergraduate research for two and half years during his time UTA in High Energy Physics under the guidance of Dr. Jaehoon Yu. Cristian has attended recognized conferences to discuss his research. Furthermore, he has made contributions to the international physics project known as ICARUS in both web development optimizations and online monitoring systems for some of their essential hardware components. He plans to continue his education by pursuing a PhD studying Cybersecurity/Information Security at UTA.