BOLT DETECTION AND POSITION ESTIMATION USING DOMAIN RANDOMIZATION

by

EZRA TIMA AMEPEROSA, B.S.

THESIS
Presented to the Graduate Faculty of
The University of Texas at San Antonio
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

COMMITTEE MEMBERS:
Pranav Bhounsule, Ph.D., Chair
Amir Jafari, Ph.D.,
Xiaodu Wang, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Engineering
Department of Mechanical Engineering
August 2018

DEDICATION

This thesis is dedicated to my family and my loving wife Vanessa, who have supported me throughout this journey.

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Dr. Pranav Bhounsule for always looking after my wellbeing. You have always challenged me, and you also helped me land my first engineering job. I would like to also thank Andrea Helbach, Howard Sizek, and Steve Turek from AFRL for guiding me through my project. I would like to express my sincere gratitude to Austin Deric and SwRI for introducing me to this thesis project and extending a helping hand. Lastly, I would like to thank my lab mates in the RAM Lab especially Robert Brothers and Sebastian Sanchez. I have had the best and the most stressful time going through the Master Program together with you guys.

August 2018

BOLT DETECTION AND POSITION ESTIMATION USING DOMAIN RANDOMIZATION

Ezra Ameperosa, M.S.
The University of Texas at San Antonio, 2018

Supervising Professor: Pranav Bhounsule, Ph.D.

Current robot automation efforts of aircraft fuselage layer joining are very inefficient; specifically, in detecting fasteners. Present method of localizing fasteners is manually feeding the locations to the robot which is an expensive task for the number of fasteners needed for joining.

We propose using machine learning to identify and locate the position of bolts on a work piece without prior knowledge of their location. The approach is to generate synthetic data using the technique of domain randomization to use in machine learning to identify bolts and their location. The rationale is that with enough variability in the synthetic images, real-world images will appear as another instance of the synthetic data; this process will allow us to work in varying surrounding conditions. With this method we create synthetic images with several instances of the workspace varying lighting, occlusions, textures, and noise and feed it to a neural network which will return a location of the bolt.

In testing the accuracy of this proof-of-concept, a test bed will consist of workpiece embedded with bolts with known locations. Our developed detector will be given real images of the testbed and will (1) detect the number bolts in the image, and (2) estimate the position of each detected object. On our three test sets we can detect bolts with at least 85% accuracy, while for estimating position our method is able to predict 75% of the bolts under 1.27cm error for the first two test sets and on the third test set predict 75% bolts under 2.54cm.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

In programming robots to perform tasks, it is very common to create a simulation of the robot prior to dispatching the program to the robot. With simulations we can envision a robot's behaviors and test on actual robots. This saves users time and money as we can eliminate bugs and reduce robot down time.

However, there lies an issue in transferring simulation into hardware: the reality gap. While we can make exquisite simulations that represent the real world accurately, there is the underlying fact that nothing is made perfect. Moreover, how similar the simulation and real environment corresponds to how well the simulation will carry over to hardware.

With the advancements in machine learning and deep neural networks, we see work in closing the reality gap using synthetic data. Using synthetic data is attractive as neural networks require a lot of data and creating synthetic data is quick with the advances in computation power. Furthermore, collecting and labeling data is time consuming, however we can quickly and easily label our data if we generate synthetic data.

Our objective is to train a neural network identify and localize bolts on a workpiece that is unaffected by the difference in simulation and the real world (lighting, distractors, textures, noise). To perform this task, we use the method of domain randomization to create training images that have randomized uncertainties between simulation and the real world like lighting, exact camera position, textures, etc. This allows the network to be flexible and capable of working in a range of environments.

1.1 Thesis Contributions

In completing this thesis, we look to demonstrate on a workpiece as a proof of concept for detecting bolts on a fuselage. With this work we'll provide our methods for developing synthetic data and training methods to be used for anyone to develop their own data. We will also be making our training data available, as well as our script for generating synthetic images of our workpiece for others to use in their work. We will also be providing the test sets we create for testing our methods open for other researchers to improve on the work in this thesis. Contributions we make compared to other works is using strictly synthetic images to train a network that identifies objects and makes precise position estimates of multiple objects concurrently.

CHAPTER 2: BACKGROUND AND RELATED WORK

2.1 Convolutional Neural Network

For this work we use convolutional neural networks (CNN) specifically for classification and regression. In short, we provide a CNN an image as an input, and it will go on to evaluating features of the image to correlate the image to a specific characteristic we are interested in. CNNS there are three main components, convolutional layer, pooling layer, and fully connected layer. Here we will introduce these separate parts of the CNN and other special tools in using CNN.

2.1.1 Convolutional Layer

Convolutional layers act as a learnable filter for extracting features that the neural network believes are important. The convolutional layer accepts a data volume of W_1 x H_1 x D_1. We slide K number of filters of size F x F x D_1 across the data volume shifting the filter S units over. The types of the filters are initialized randomly and are readjusted through training the neural network. Padding is also optional to use in filtering which adds P extra layer units around the input image to preserve information of the input volume. Through the filter of the input volume we obtain and output volume of W_2 x H_2 x D_2 where we use equations (11),(2), and (3) and then apply an activation function.

$$W_2 = \left\lfloor \frac{W_1 - F + 2P}{S} + 1 \right\rfloor \tag{1}$$

$$H_2 = \left\lfloor \frac{H_1 - F + 2P}{S} + 1 \right\rfloor \tag{2}$$

$$D_2 = K \tag{3}$$

We illustrate the first two strides of a convolution of an input of size 4 x 4 x 1 using a filter of 3 x 3 x 1 (Figure 1). We add one layer of padding with values of zeros and set one-unit strides to produce an output size of 4 x 4 x 1.

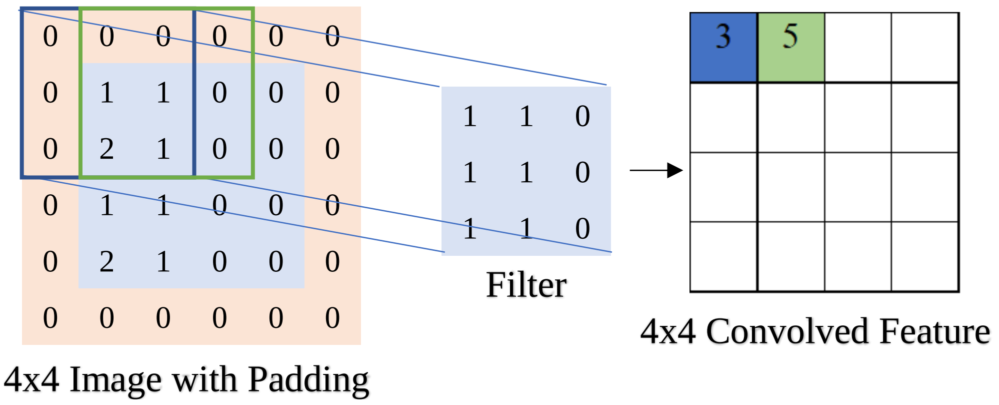

Figure 1: Example Convolution

2.1.2 Activation Function

Typically, after convolutional layers and in computing fully connected layers, we use activation functions to map the values to a certain threshold and introduce nonlinearity to a system. The most common activation function for a simple binary classification (True of False) is to use sigmoid function as it maps the inputs from zero to one. In this work we use Rectified Linear Unit (ReLu) as it is very simple and computationally light on back propagation in training. We show common activation functions in the following table.

Table 1: Common Activation Functions

Name	Function
Sigmoid	$f(a) = \frac{1}{1 + e^{a}}$
Tanh	$f(z) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$
ReLu	$f(z) = \max(0, a)$

2.1.3 Pooling Layer

The function of the pooling layer (also known as downsampling) is to reduce the spatial dimension. The two main purpose of doing this is to allow for decrease the number of parameters the neural network have to calculate, and to help with reducing overfitting. There are several types of pooling such as max-pooling and average-pooling, but we opt to use max-pooling.
In performing max-pooling, the input to the layer is a data volume of W_1 x H_1 x D_1 and outputs a size W_2 x H_2 x D_2. The output is based on the window of size F, stride, and padding size and uses the same equations in (11), (2), and (3). For max-pooling we slide a window across the input image and outputs the max value within the window. We illustrate this concept in Figure 2 where we have an input size of 4 x 4 x 1, window size of two, no padding, and a stride of 2; our output comes to be a 2 x 2 x 1 data volume.

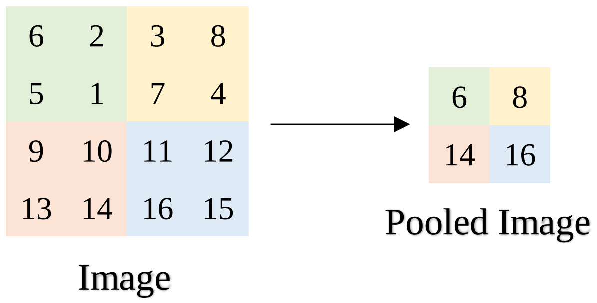

Figure 2: Pooling Example

2.1.4 Fully Connected Layer

Typically, after multiple convolution and pooling layers we transition into fully connected layers. Fully connected (FC) layers (also known as hidden layers) consist of several neurons that execute a weighted sum of the input and applies an activation function (Figure 3).

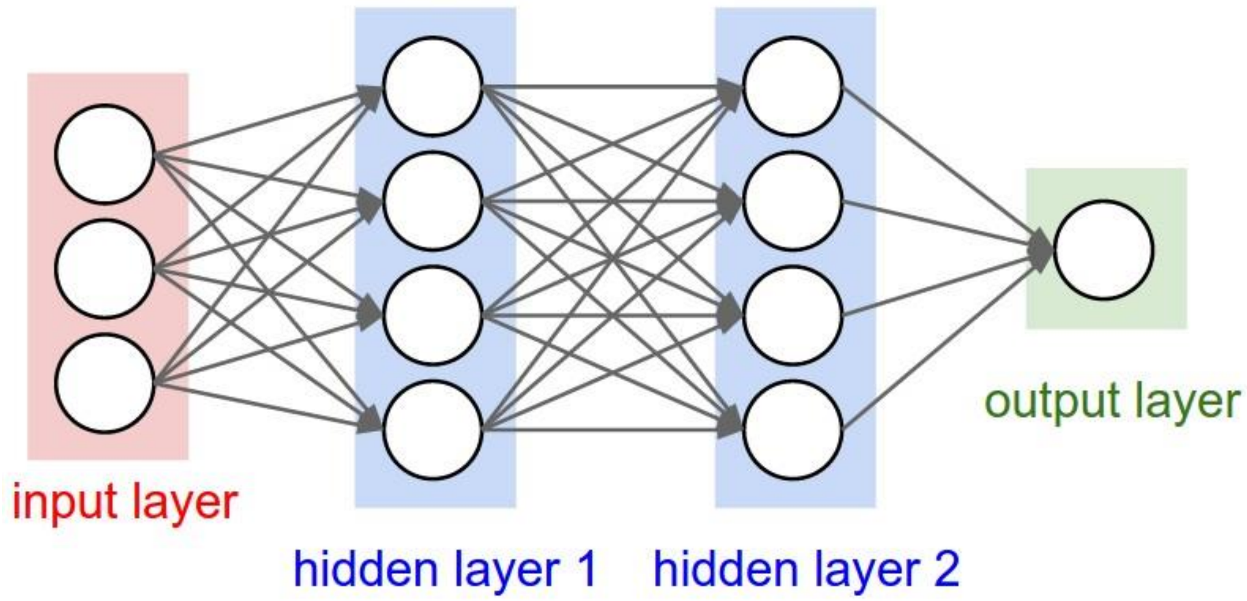

Figure 3: Example of Fully Connected Layers [22]

Each neuron layer in the layer takes the previous layers data as input, the output then connects to each neuron of the next FC layer. The neurons take in the vector x = $(x_0\ x_1\ \ldots\ x_n)^T$ as input into equation (4) where *w* and *b* are learnable parameters, weight and bias. Equation (4) is then used as input for the activation function (Table 1). The output of the activation function is used as the input for the next layer.

$$a = w^T x + b \tag{4}$$

2.1.5 Softmax and Classification Layer

In performing multi-classification softmax activation function is used to determine the multiclass probability. The equation of softmax is shown in equation (5) where P is the probability of the *i*-th neuron out of *k* total neurons.

$$P(a_i) = \frac{e^{a_i}}{\sum_{j=1}^{k} e^{a_j}} \tag{5}$$

The softmax function normalizes the input hidden layer such that $0 \leq P(a_i) \leq 1$ and $\sum_{j=1}^{k} P(a_j)$ =1. The output values each represent a categorial distribution where in the case image classification, each output value represent a label and the highest value declares the input image's label. During training, the neural network's prediction is compared to the true label through cross entropy loss described as:

$$E(y, \hat{y}) = -\sum_{i} y_i log \hat{y}_i \tag{6}$$

Where y and $\hat{y}$ is the target value and the output at neuron i, respectively. With this error function we perform back propagation, described in a further section.

2.1.6 Regression Layer

In our work we use regression layer (specific to MATLAB) to predict the position of bolts. The regression layer is primarily used for back propagation in training a neural network. The final FC layer in the neural network predicts values and has no activation function like the softmax layer. The output value of the FC layer is compared to the true values in our regression layer using mean squared error (equation (7) to be used in training the network.

$$MSE = E(y, \hat{y}) = \sum_{i=1}^{k} \frac{(y_i log \hat{y}_i)^2}{k} \tag{7}$$

2.1.7 Training and Back Propagation

For training the neural network we must compare the neural network's guess to true value in order to create a smarter network. In training the network an image is fed into the network of convolutional, pooling, and fully connected layers. The image is passed through the layers and the network observes the features and takes a guess of what it believes it should be searching for. At this point the network's prediction is compared to the true value using the error functions described in section 1.1.5 and 1.1.6. The error is then used to the update parameters such as the weights and bias in the fully connected layer, as well as the filters in the convolutional layer. This process is performed several times to learn to predict the true values accurately. During training, the network is given true values of what it should be searching for, however when implementing the network, the true values may not be available.

2.2 Faster R-CNN

Convolutional neural networks work great for classifying and labeling what an image contains, but what happens when the image contains more instances of the object of interest or contains multiple labels? This is where Faster R-CNN [1] comes into play. Faster R-CNN has two main parts: a convolutional neural network, and a Region Proposal Network (RPN). Apart from its slower predecessors [2], [3], Faster R-CNN uses a light weight network that takes advantage of using the CNN’s convolutional feature map to generate bounding box proposals (Figure 4). With the bounding boxes training can be done to perfect the RPN, the final classifier and a regression for bounding box. [4]

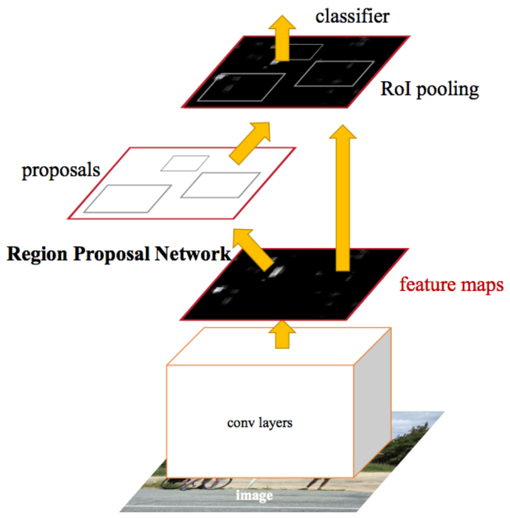

Figure 4: Faster R-CNN visualization [1]

2.3 Overview of Related work

Visual based object detection is a well-studied problem in literature [5] [6] [7], however the authors use an RGB-D camera to detect depth. In our work we use a single monocular camera. Domain randomization is the method where instead of training a model on a single simulated environment, the model is exposed to a randomized environment during training to allow it to be flexible and invariant to differences between the simulated model and the real model [8]. We wish to use this idea to generate synthetic images to train our neural network. A lot of inspiration for this work span from [8] and claim to be the first to successfully transfer a deep neural network trained only on synthetic data, however they use large and distinctive colored objects, and they do not obtain information of multiple objects at once. Many other authors have used domain randomizations [9] [10] but have not transferred to perform in the real world. However the authors of [10] have transferred to the real world for grasping objects by randomizing policies for grasping objects [11] [12]. Authors of [13] [14] [15] have transferred from simulation to the real world but do not perform fine measurements.

A step past domain randomization is to continue training on real the real objects. Bousmalis et. Al. uses Generative Adversarial Networks to make synthetic images look real by comparing it to the real images use the GAN to make the synthetic images seem real [16] The same author uses this work for adapting a synthetic image to appear real to the network for robot pick & place [17] The authors in [18] and [19] use a slight alternate of domain randomization where they instead randomize the physical properties of their robots to perform tasks.

CHAPTER 3: METHODOLOGY

3.1 Data Generation.

In generating the synthetic images, we use the open source software, Blender [20] . We create scenes resembling our physical system, randomizing (1) camera position and field of view (FOV), (2) lighting position and properties, (3) distraction object shape, positions and the number of distraction objects, (4) position of the bolts on a workpiece, and (5) color and texture of table, workpiece, bolts, and distractors. We generate scenes by randomizing the 5 aforementioned features. We fix the workpiece to the center of the scene with the size set to 30.5cm by 30.5cm. A table is set beneath the work piece that is arbitrarily sized to occupy space in the camera frame not covered by the workpiece. The bolt used in the synthetic images is taken from McMaster-Carr’s online 3-D models.

3.1.1 Camera

The randomization of the camera position would allow us to avoid a calibration step that would be performed before testing. Typically, there is an extrinsic calibration to accurately map pixel coordinates to camera coordinates. However, with our method we make our network give us results as if the camera was already calibrated. By randomizing the field of view (FOV), we again skip calibration of finding the FOV of the camera as well as give us the flexibility to use different cameras with different FOV We use a single camera in Blender to capture the synthetic images and randomize the camera’s position with respect to the center of the workpiece. The camera is positioned [-7.6cm,7.6cm] from the center in the x-direction, [-30.5cm, -45.7cm] away in the y-direction, and [30.5cm, 45.7cm] above in the z-direction. After randomizing the position, we aim the camera to the center of the workpiece.

In randomizing the camera FOV in Blender, we estimate the FOV of our physical camera and implement it in the synthetic images. We add a small buffer to allow for any error in estimating the FOV and set the randomization range between [35˚, 40˚].

3.1.2 Lighting

We create two lighting sources where we use the first lighting source as the key light and the second source as backlighting. The key light is given a yellow tint while the backlight is given a light blue tint to mimic lighting in our actual testing environment. The backlighting is also used to soften shadows made by the key lighting. The key lighting's intensity is randomized, and the backlighting is approximately half of the intensity of the key lighting with some added variance.

Positions of the key lighting are placed above the workpiece occupying a semi spherical space with a radius of 2.13 meters from the center of the workpiece. The backlighting is positioned 180˚ about the center of the workpiece from the key lighting. Both light sources point at the center of the workpiece and add up to 10˚ variation.

3.1.3 Distraction objects

The distractors act as a way to prevent the neural network from overfitting on the synthetic data. Without the use of these distractor objects, the neural network would be vulnerable to false positive detections of random objects and any features not captured in training. We use four shapes available in Blender as distractor objects: cube, sphere, cylinder, and cone. The sizes of the distractors are made to be about the same size of the bolt. We vary the number of distractors in each scene from [0, 8] as well as the shape each distractor is. The distractors are placed arbitrarily onto the workpiece.

3.1.4 Bolts

The position of the bolt is placed anywhere on the workpiece excluding the edges. We set the origin to the bottom left corner of the workpiece and record the cartesian coordinates, *x* and *y*, position of the bolt relative to the origin. We also record the bounding boxes of the bolt in the synthetic images to be used for the RPN of our Faster R-CNN.

3.1.5 Color and Texture

We setup three textures for the bolt, distractors, table, and workpiece: solid color texture, marble texture, and a gradient texture. For each of the objects in the scene, we randomly set a texture to the objects and randomize the colors of the texture. We also add the possibility for objects to have [0%, 60%] reflectivity.

Scenes are generated with these randomized features then images are rendered in Blender to a resolution of 448 x 448. We create 75,000 images (Figure 11) to be used for training and validation, with 50,000 images containing only one bolt and 25,000 images containing two bolts on the workpiece.

3.2 Network

For our neural network we use two instances of the VGG16 architecture with pretrained weights from ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [21]. We modify each instance to perform separate tasks. The first instance is to identify all bolts in an image while the second instance is to estimate the position of the bolts on the workpiece. Both networks are also slightly modified to accept 448 x 448 images rather than 224 x 224 to increase the resolution of the tiny size of the bolt. We then combine these two networks at test time to detect all bolts in each image and its position on the work piece. We use MATLAB to train our networks and implement it on our physical system.

3.2.1 Training the network

For the first network we use VGG16 with pretrained weights from ILSVRC. We modify the network to accept 448 x 448 images then add two convolutional layers with ReLu activations following each layer. The convolutional layers have 32 filters with a filter size of 3 x 3. We add padding size of one and use a stride of one. We add a max pooling layer after the second convolution layer with a pool size of two, zero padding and a stride of two. We add these new layers to the beginning of the VGG16 architecture. We change the last fully connected layer to a two-neuron layer and then change the softmax and classification layer to have an output label of either a bolt or background.

For training the first neural network, we use Faster R-CNN to detect multiple bolts in an image. We split the 75,000 synthetic images into 67,500 training images, 7,500 into validation images. We use the training set and the bounding boxes taken from Blender as input to the network. Stochastic gradient descent with momentum is used because MATLAB does not allow Adam optimization for Faster R-CNN. We use 0.9 for the momentum coefficient. We add L2 regularization with a coefficient of 1e-4. A mini batch size of 256 is used and trained for three epochs. We also increase the bounding box sizes of the bolts in the training images as MATLAB cannot process objects that are too small. Because we increase the bounding box sizes, we modify how we evaluate our detector's ability. We discuss this in a further section.

For the second network we use the first network that has been trained on classifying bolts and modify the last fully convolutional layer to a new two neuron layer followed by a regression layer that outputs two values, x and y position.

Training the second neural network we use the 49,000 images that only contain a single bolt and use the other 1,000 for validation. We don't use the images with two bolts as the neural network can only estimate the position of one bolt at a time. We use Adam optimization with an initial learning rate of 1e-4, and a drop rate factor of 0.1 every 4 epochs. We train the network for 9 epochs with no L2 Regularization and a mini batch size of 32.

3.3 Full Algorithm

After training the networks we create an algorithm to take the positive detections from the first network and send them into the second network to estimate a position. First, we run the image of our physical system through the first neural network and detect *k* positive detection of bolts. Since the second network is only capable of estimating the position of one of *k* bolts in an image at a time, we devise the method of running *k* copies of the image through the second network *k* times.

We make *k* copies of the input image to use as input for the second network to estimate the positions. We take a copy of the image and 'blur' out the all positive detections in the image except for a single positive detection. This is repeated for all *k* copies such that each copy contains a different detection that is not blurred. For blurring the separate positive detections, we take its bounding box and find the average color of the pixels inside the bounds and change all the pixels in the bounds into the average color. We take the *k* images and run them through the second network and retrieve estimated positions for all *k* positive detections. We then take all *k* estimated positions and link them to their corresponding positive detection (Figure 12).

CHAPTER 4: EXPERIMENTS

To test the effectiveness of the algorithm, we create three test sets that exhibit the algorithm to different scenarios. We record test images in a room where we do not control the lighting condition and stay constant throughout the testing. We use a 30.5 x 30.5 wooden pegboard as our workpiece and place it on an optical breadboard table. We randomly place two bolts in the workpiece. In the first test set we use the same type of stainless steel bolts for both bolts. We then replace one stainless steel bolt for a black oxide bolt for the second test set. For the last test set we mix between the stainless steel, black oxide and a cadmium-plated bolt. We also mix in another color workpiece in the third set. For the first and second test set, we place our off the shelf camera at an arbitrary distance and height, straight ahead from the workpiece and for the third set we place the camera 7.6cm to the left-- no kind of camera calibration was performed for testing. In the first two sets, when changing the position of the bolts, we place the workpiece on the table in a fixed *Y* position on the table but leave the *X* position unfixed. The third test set the workpiece is placed in a fixed position on the table. For each set we take 20 images with the two bolts placed in various positions.

For evaluating the effectiveness of our algorithm, we measure precision and recall for the first detection network. We define recall and precision as:

$$Recall = \frac{True\ Positives}{True\ Postives + False\ Negatives} \tag{8}$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{9}$$

Recall it is the ratio of correctly identified instances to all correct instances. In other words, did the detector find all instances of the bolts? As for precision, it is the ratio of the correctly identified instances to total number of positive detections. What this measure is—of all the instances where the detector thought it saw a bolt, how many detections are actually bolts? With precision and recall we can better measure how well our detector performed in contrast to measuring just how many bolts it detected out of the total bolts. We use the precision and recall in finding the F1 Score at varying thresholds When we measure recall and precision, we do not consider Intersection over Union (IoU) for measuring how well the true and estimated bounding boxes compare, rather, we ignore a minimum IoU. This is a repercussion of increasing the bounding box size during training. The F1 score (equation (10) is the harmonic mean of both recall and precision to give a performance score. We use this metric to find the score for different confidence thresholds in detecting the bolt. We use the threshold with the highest score for each test set and report how well the detector performs.

$$F1\ Score = \frac{2 \times Recall\ \times Precision}{Recall + Precision} \tag{10}$$

We measure the accuracy of the algorithm to predict the bolts' position on the workpiece with respect to the origin. We run our test set images through our algorithm and retrieve estimated positions. Using equation (11) we find the distance between true position and estimated position to determine the error where x and y are the true positions and $\hat{x}$ and $\hat{y}$ are the estimated positions.

$$Bolt\ Position\ error = \sqrt{(x - \hat{x})^2 + (y - \hat{y})^2} \tag{11}$$

We collect the mean of the errors for each test set as well as the standard deviation. We look at how many bolts have an error of less than 1.22cm and discuss how well the algorithm estimated position.

4.1 Detection Evaluation

Set one we evaluate the precision and recall at confidence thresholds between [50, 70]. The precision versus recall plot is shown on Figure 5a.

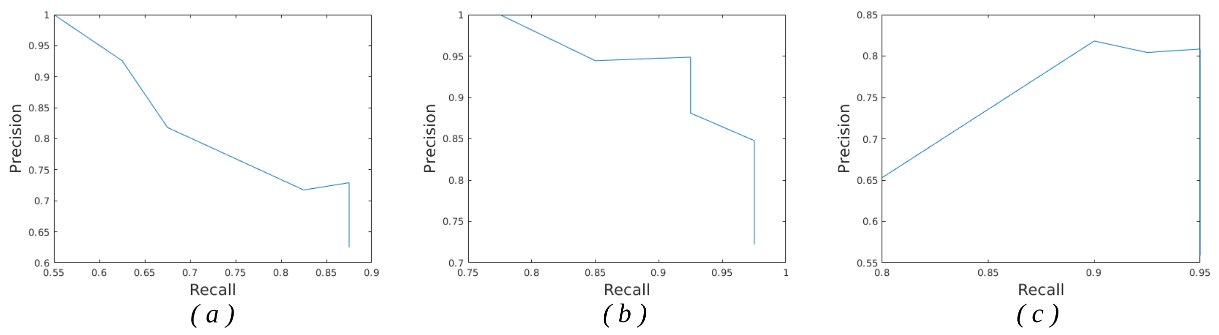

Figure 5: Precision vs Recall for (a) Test Set 1 (b) Test Set 2 (c) Test Set 3

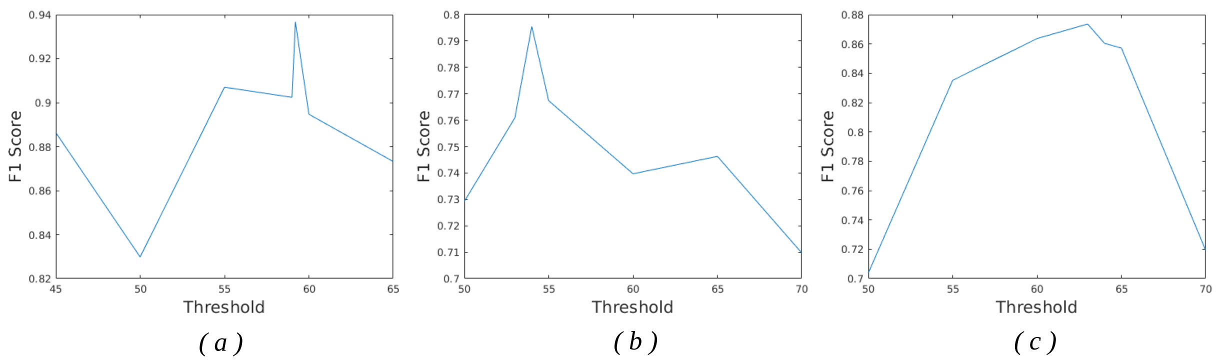

Figure 6: F1 Score vs Threshold for (a) Test Set 1 (b) Test Set 2 (c) Test Set 3

We find the F1 score for each threshold and find the maximum F1 score of 0.73 at a threshold of 54% (Figure 6a). At a threshold of 54% we have recall of 35 of 40 bolts correctly identified and precision of 35 bolts out of 48 total positive detections (Table 2).

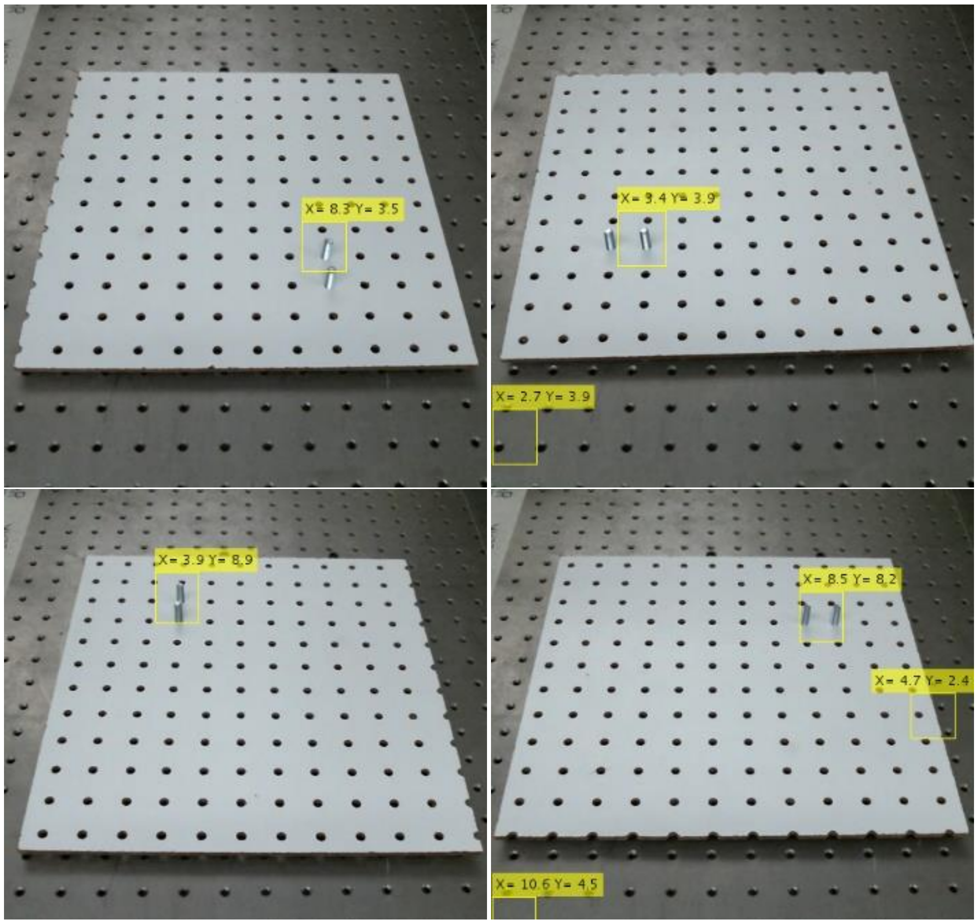

Figure 7: Incorrect Bolt Detection

In this test set we have four images in specific that have bolts placed 2.54cm away from each other on the workpiece-- we find that in our detection step, it has challenging time identifying the bolts separately (Figure 7). When detecting images with bolts that are positioned within 2.54cm proximity, we see that the bounding boxes are combined into a single bounding box. This is a consequence of increasing the bounding boxes during training as MATLAB limits what region of interests it processes based on the network down sampling.

Table 2: Test Set Performance

	Test Set 1	Test Set 2	Test Set 3
Bolts correctly identified	35 out of 40	37 out of 40	38 out of 40
Average Error (Inches)	0.48	0.42	1.16
Standard Deviation (Inches)	0.53	0.29	1.73
Bolts with < 0.5 inch error	29 out of 35	30 out of 37	15 out of 38

For set two the precision and recall are evaluated at thresholds between [45, 65]. Figure 6b show that at a threshold 59.2% we achieve our highest F1 score of 0.94. At this threshold our recall is measured to correctly identify 37 out all 40 bolts in the set, while our precision is measured to have 37 correctly identified bolts out 39 positive detections. Compared to the first test set, our F1 score increased to near a perfect F1 score of 1. The different color bolts were the only change between set one and two.

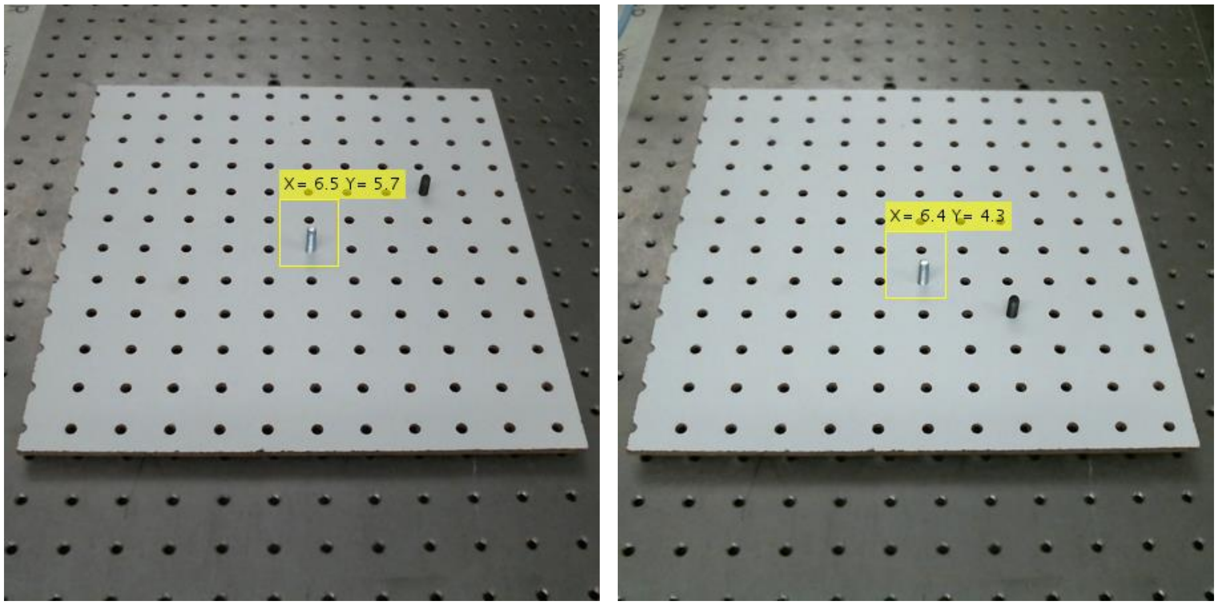

Figure 8: Missed Bolt detection in Test Set 2

Still, when we look at images that failed to identify the bolts (Figure 8), we see that the detector didn't perform better because we switched colored bolts; two of the three failed identifications were from the black bolt. We do find that the test set has bolts spaced farther than 2.54cm apart in contrast to set one. We have not tested distinct color bolts adjacent of each other, but we believe that the detector would be able to detect the bolts separately-- we wish to add this to future works.

Test set three, we have the most variations out of the other sets. We evaluate between thresholds between [50, 70] and find at a 63% threshold is where our F1 score at a maximum of 0.87 (Figure 6c). We have a recall of 38 out of 40 bolts identified and a precision of 38 bolts out 47 positive detections. Comparing to the other test sets, our detector has the highest recall in set three. Again, in this test set we have only two images that have bolts in close proximity of each other. We also see the camera automatically white balancing when changing the workpiece colors. The detector is invariant to the workpiece change as well as our camera performing automatic calibration—we have no control over the camera performing white balancing.

For all the test sets we see high recall values with set one having the lowest recall of ~0.88. As for precision, set two performs the best. Set one and three both have about the same amount false positives, and they tend to appear in the same area throughout the sets: the bottom left part of the images, and the mid-right side of the images. What seems to be causing the false positive is unknown, but they are very consistent. We also observed that the detector has only failed in detecting bolts that are on the right side of the workpiece. These failures need further observations to better our detector

4.2 Position Estimation Evaluation

The histogram shown in Figure 9a show the distance between true and estimated position on the abscissa and the frequency of the error on the ordinate for test set 1. From our algorithm the mean error is found to be 1.22cm with a standard deviation of 1.35cm (Figure 9a & Table 2). For test set 1, we estimated the position of 29 bolts within 1.27cm successfully.

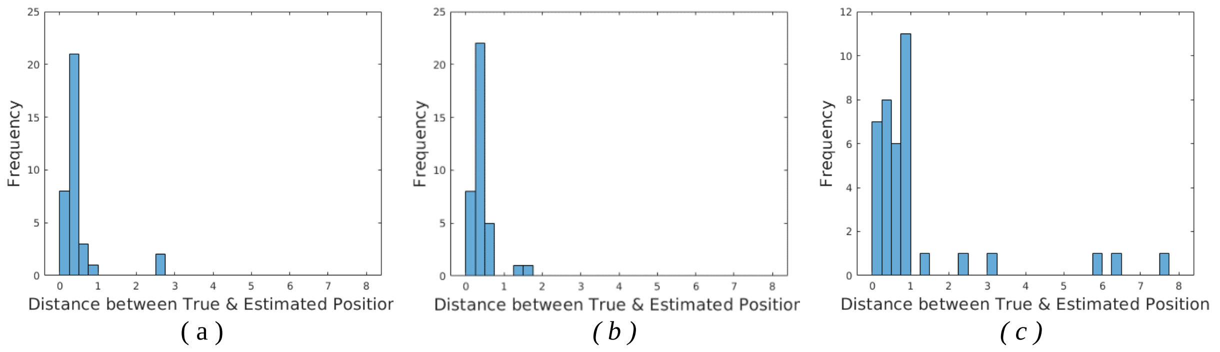

Figure 9: Histogram of (a) Test Set 1 (b) Test Set 2, (c) Test Set 3

The position estimation error in set two decreased in set two compared to set one. Our average error lowers to 1.07cm and our standard deviation lowers to 0.74cm (Figure 9b & Table 2). In comparison to set one, the decrease in error isn't dramatic. In this set we can estimate one more total of bolts correctly than test one but have less correctly estimated out of correctly identified bolts.

For set three we see a huge increase in error with the average error of 2.95cm and a standard deviation of 4.4cm. (Figure 9c & Table 2). In test three, we were only able to estimate 15 bolts under 1.27cm error, however we are able to estimate 32 bolts of the 38 identified correctly under 2.54cm.

When generating synthetic images for training, we set the camera up to a maximum of 7.6cm in either x-direction. We hypothesize that if we placed the camera closer to the middle, it would perform better than if it were placed on the extreme allowable positions. These randomizations in image generation allow us work in varying environment, but they also service as extra leeway for error in transferring from simulation to hardware. We also did not add extra variance in aiming the camera at the workpiece in the synthetic data. This leads to us having to always aim our real camera at the center of the workpiece, which we can't repeat 100% of the time. This could possibly lend into inaccuracy in position estimation.

CHAPTER 5: CONCLUSION

5.1 Summary

In this thesis we seek to produce a proof of concept for automating the process of localizing bolts on a fuselage. Although automation efforts have been partially successful, the process of passing the fastener locations to the robot is a manual process, which is a time-consuming process. We have set out to produce a proof of concept of a method that will automatically detect bolts and predict the fastener location without human intervention.

We identify our testbed and develop synthetic images that don’t necessarily match our real system. The synthetic images are used for training a convolutional neural network and contain several variations such as lighting, camera position, and textures. These variation span from the idea that if there is enough variation in training, the neural network will be invariant to poor lighting conditions, uncalibrated cameras, and other undesirable conditions. We develop an algorithm that consist of two neural network that perform the separate tasks of identifying the bolts and prediction the location of the bolt on a workpiece and are able to identify and predict bolts precisely.

5.2 Limitations

In this section we discuss limitations and short comings of our proposed method.

5.2.1 Network dependence

Because we split images with several bolts into separate images to feed into our position estimating network, we rely on our detection network on accurately identifying all the bolts. In

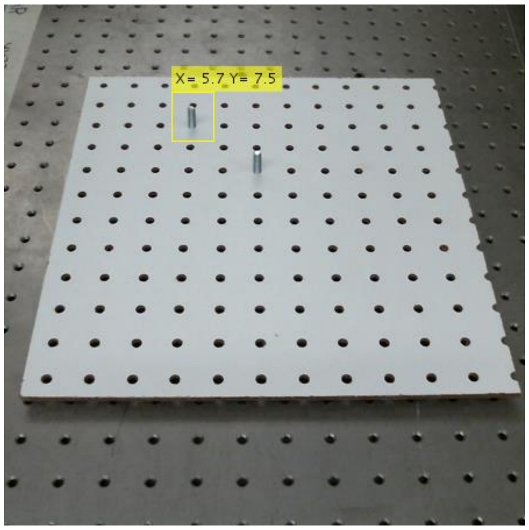

Figure 10: Incorrect Identification of bolt

Figure 10 we show that the detector only identified one bolt. The detected bolt's estimated position is (5.7, 7.5), however its true position is (3.5, 9.5). The true position of the undetected bolt is (5.5, 7.5). With our method the algorithm would make two images, blurring out one bolt in one image then blurring out the other bolt in the second image, then send it through the position estimating network. However, in this instance, because the detector only identified one bolt, it does not blur any bolts and sends the unedited images through the second network. Through this we are not able to control which bolt the second network looks at to estimate its position. In this case we get the position of the undetected bolt and is unintentionally associated to the detected bolt. To remedy this is to create a single network that allows for multiple regressions akin to regional proposal networks rather than separate networks.

5.2.2 Duplicating Images for Position Estimation

Continuing our discussion on our procedure of duplicating images, its not very efficient for images with a considerable number of bolts. Presumably, if we are presented a setting where a workpiece has 20 bolts we must find the position for, our algorithm will make 20 separate images for each bolt and send each one through our second network one at a time. Again we suggest creating a single network that uses a RPN that is lightweight on computation and capable of measure multiple objects simultaneously.

5.2.3 Overlapping bounding boxes

Previously discussed, when the detector's bolts that are too close to each other our detector tends to combine the bounding boxes into a single bounding box for both. Again, this roots from needing to expand the bounding boxes to compensate for MATLAB having a minimum region of interest size that it will compute. When the images go through the RPN, it goes through a non-max suppression layer where the RPN eliminates copies of similar bounding boxes that have an IoU less than a certain threshold. Two corrective actions we could take would be: (1) somehow bypass MATLAB's limitation and use the correct, unedited bounding boxes during training or (2) increase the IoU threshold to place a stricter policy for discarding similar bounding boxes.

With MATLAB we lose a lot of flexibility in how we make our neural network—in hindsight a stronger framework such as TensorFlow, Caffe, etc. should have been used.

5.3 Future Works

Extending from the limitations of our algorithm, we wish to perform these tasks to improve the algorithm's performance. Our highest priority for improving the algorithm would be to switch to a stronger neural network framework as MATLAB serves as an incredible tool for learning but not so much for development.

Aside from eliminating our limitations we would like to extend our algorithm to more complex workpieces where we add more dimensions than just x and y position of a bolt on our workpiece, e.g. roll, pitch, yaw, and z position. This would give us more knowledge of our environment and the opportunity to perform more complex tasks such as finding the normal vector of the bolt to let us place a nut on the bolt.

In the near future we would like to add more test sets to be available to the public. Specifically, we would like to add more images that challenge our method and give others the opportunity to improve from our results. In addition to our current sets, we want to add test sets that add more adversarial objects in the image than just holes. Add different lighting environments and use cameras that are vulnerable to lens distortion and image noise.

APPENDIX

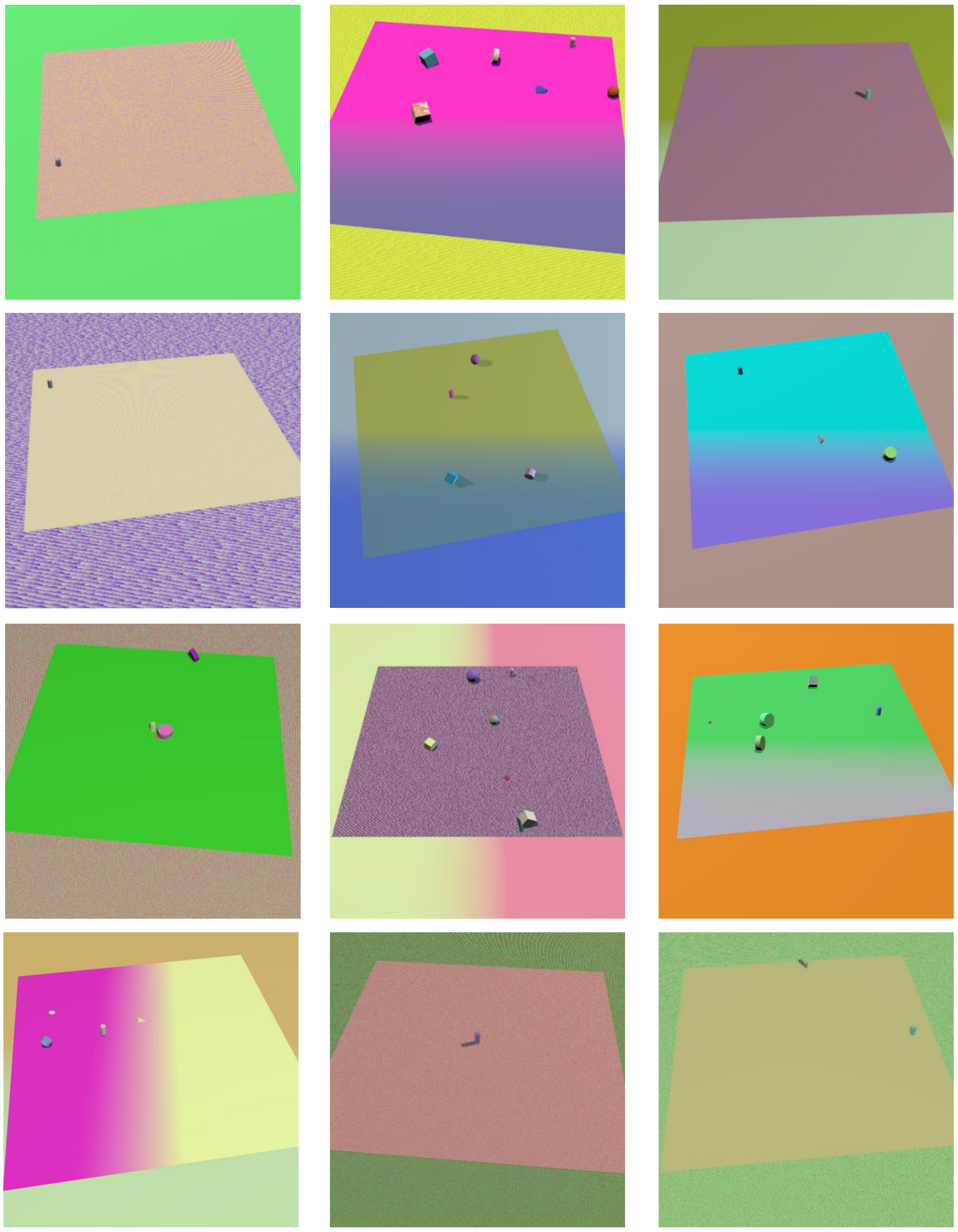

Figure 11: Select Training Images

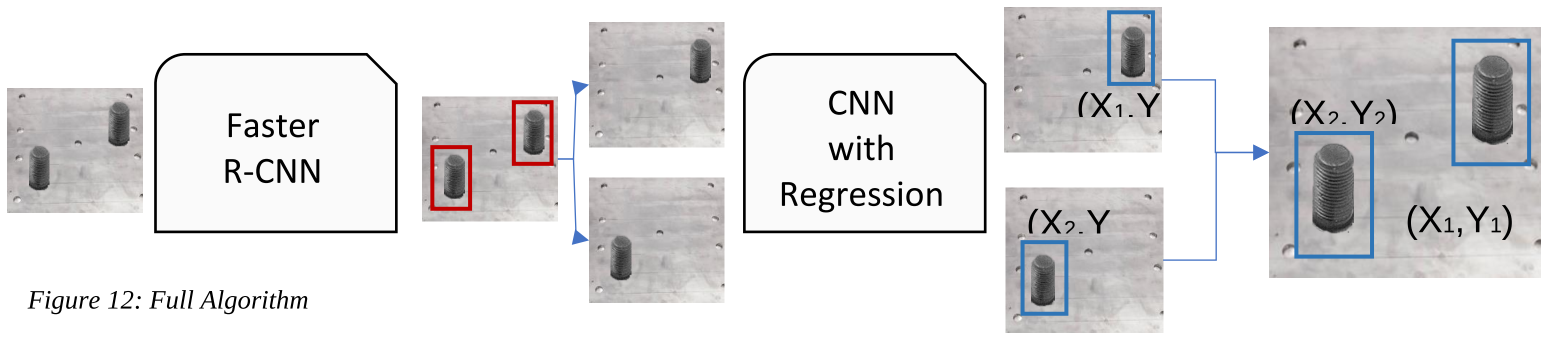

Figure 12: Full Algorithm

WORKS CITED

[1] S. . Ren, K. . He, R. B. Girshick and J. . Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, no. 6, pp. 1137-1149, 2017.

[2] R. B. Girshick, "Fast R-CNN," *arXiv: Computer Vision and Pattern Recognition,* vol. , no. , pp. 1440-1448, 2015.

[3] R. B. Girshick, J. . Donahue, T. . Darrell and J. . Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," *arXiv: Computer Vision and Pattern Recognition,* vol. , no. , pp. 580-587, 2014.

[4] Y. Choe, H.-C. Lee, Y.-J. Kim, D.-H. Hong, S.-S. Park and M.-T. Lim, "Vision-Based Estimation of Bolt-Hole Location using Circular Hough Transform," in *ICROS-SICE International Joint Conference*, Fukuoka, 2009.

[5] R. . He, J. . Rojas and Y. . Guan, "A 3D object detection and pose estimation pipeline using RGB-D images," *arXiv: Robotics*, vol. , no. , pp. 1527-1532, 2017.

[6] Y. . Xiang and D. . Fox, "DA-RNN: Semantic Mapping with Data Associated Recurrent Neural Networks," *arXiv: Computer Vision and Pattern Recognition,* vol. 13, no. , p. , 2017.

[7] A. . Zeng, K.-T. . Yu, S. . Song, D. . Suo, E. W. Jr., A. . Rodriguez and J. . Xiao, "Multi-view self-supervised deep learning for 6D pose estimation in the Amazon Picking Challenge," *arXiv: Computer Vision and Pattern Recognition,* vol. , no. , p. , 2017.

[8] J. . Tobin, R. . Fong, A. . Ray, J. . Schneider, W. . Zaremba and P. . Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *arXiv: Robotics,* vol. , no. , pp. 23-30, 2017.

[9] X. . Peng, B. . Sun, K. . Ali and K. . Saenko, "Learning Deep Object Detectors from 3D Models," *arXiv: Computer Vision and Pattern Recognition,* vol. , no. , pp. 1278-1286, 2015.

[10] J. . Mahler, F. T. Pokorny, B. . Hou, M. . Roderick, M. . Laskey, M. . Aubry, K. J. Kohlhoff, T. . Kröger, J. J. Kuffner and K. . Goldberg, "Dex-Net 1.0: A cloud-based network of 3D objects for robust grasp planning using a Multi-Armed Bandit model with correlated rewards," , 2016. [Online]. Available: http://dblp.uni-trier.de/db/conf/icra/icra2016.html. [Accessed 3 8 2018].

[11] J. . Mahler, J. . Liang, S. . Niyaz, M. . Laskey, R. . Doan, X. . Liu, J. . Aparicio and K. . Goldberg, "Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics," *arXiv: Robotics,* vol. 13, no. , p. , 2017.

[12] J. . Mahler, M. . Matl, X. . Liu, A. . Li, D. V. Gealy and K. . Goldberg, "Dex-Net 3.0: Computing Robust Robot Suction Grasp Targets in Point Clouds using a New Analytic Model and Deep Learning.," *arXiv: Robotics,* vol. , no. , p. , 2017.

[13] L. . Pinto, M. . Andrychowicz, P. . Welinder, W. . Zaremba and P. . Abbeel, "Asymmetric Actor Critic for Image-Based Robot Learning," *arXiv: Robotics,* vol. 14, no. , p. , 2017.

[14] F. . Sadeghi and S. . Levine, "CAD2RL: Real Single-Image Flight Without a Single Real Image," *arXiv: Learning,* vol. 13, no. , p. , 2017.

[15] A. . Siravuru, A. . Wang, Q. . Nguyen and K. . Sreenath, "Deep visual perception for dynamic walking on discrete terrain," , 2017. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1712.html. [Accessed 3 8 2018].

[16] K. . Bousmalis, N. . Silberman, D. . Dohan, D. . Erhan and D. . Krishnan, "Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks," *arXiv: Computer Vision and Pattern Recognition,* vol. , no. , pp. 95-104, 2017.

[17] K. . Bousmalis, A. . Irpan, P. . Wohlhart, Y. . Bai, M. . Kelcey, M. . Kalakrishnan, L. . Downs, J. . Ibarz, P. P. Sampedro, K. . Konolige, S. . Levine and V. . Vanhoucke, "Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping," *arXiv: Learning,* vol. , no. , p. , 2018.

[18] R. . Antonova, S. . Cruciani, C. . Smith and D. . Kragic, "Reinforcement Learning for Pivoting Task.," *arXiv: Robotics,* vol. , no. , p. , 2017.

[19] X. B. Peng, M. Andrychowixz, W. Zaremba and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," 2017.

[20] "Blender Foundation - blender.org," , . [Online]. Available: http://www.blender.org/blenderorg/blender-foundation/. [Accessed 3 8 2018].

[21] O. . Russakovsky, J. . Deng, H. . Su, J. . Krause, S. . Satheesh, S. . Ma, Z. . Huang, A. . Karpathy, A. . Khosla, M. S. Bernstein, A. C. Berg and L. . Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision,* vol. 115, no. 3, pp. 211-252, 2015.

[22] A. Karpathy, "CS231n Convolutional Neural Networks for Visual Recognition," [Online]. Available: http://cs231n.github.io/neural-networks-1/. [Accessed 8 June 2018].

VITA

Ezra Ameperosa is from Killeen, TX. He studied mechanical engineering and earned his Bachelor’s in Mechanical Engineering from The University of Texas at San Antonio will earn his Master’s in Mechanical Engineering from there as well. He plans to automate his entire life, so he can enjoy being lazy.