2019

Development of a Neural Network-Based Object Detection for Multirotor Target Tracking

Spencer Harwood
South Dakota State University

DEVELOPMENT OF A NEURAL NETWORK-BASED OBJECT DETECTION FOR MULTIROTOR TARGET TRACKING

BY

SPENCER HARWOOD

A thesis submitted in partial fulfilment of the requirements for the

Master of Science

Major in Mechanical Engineering

South Dakota State University

2019

DEVELOPMENT OF A NEURAL NETWORK-BASED OBJECT DETECTION FOR MULTIROTOR TARGET TRACKING

SPENCER HARWOOD

This thesis is approved as a creditable and independent investigation by a candidate for the Master of Science degree and is acceptable for meeting the thesis requirements for this degree. Acceptance of this thesis doesn't imply that the conclusion reached by the candidate are necessarily the conclusions of the major department.

Marco Ciarcià, PhD Thesis/ Major Advisor	Date
Kurt Bassett, PhD Head, Mechanical Engineering	Date
Dean, Graduate School	Date

ACKNOWELDGEMENTS

I would first like to acknowledge my thesis advisor Dr. Marco Ciarcià. It was through his support and guidance, that this work was made possible. Dr. Ciarcià constantly advised me on my thought process and taught me many new and alternative methods when confronted with new challenges. I was continually given independence on my work with this paper and his support is fully appreciated.

I would also like to thank the numerous members of the ARTLAB who were involved in reviewing my paper and I am gratefully indebted to them for their very valuable comments on this thesis.

Finally, I would like to acknowledge my wife Jade Harwood and my entire family for providing me with the necessary support required to complete the years of study, research, and writing to produce this thesis paper. Without their support this work would not have been possible. Thank you.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

DEVELOPMENT OF A NEURAL NETWORK-BASED OBJECT DETECTION FOR MULTIROTOR TARGET TRACKING

SPENCER HARWOOD

2019

Unmanned aerial vehicles (UAVs) have, for the past few decades, seen an increased popularity in industry and research centres. Despite this intense utilization by both markets there exists an active demand for the development of autonomous guidance, navigation, and control strategies. One need relates to the achievement of a high level of autonomy to identify and track a target object. An elective technique for this set of tasks is neural networks. In the development and study of these networks there is a distinct lack of substantive validation techniques to qualify network performances when implemented in a multirotor UAV. This thesis will first describe the development of a neural network-based object detection subsystem for use in target tracking with an autonomous multirotor UAV. Then, the second part of this thesis will utilize a developed indoor multirotor testbed to externally verify the tracking performance of the multirotor UAV during an object following maneuver.

CHAPTER ONE

INTRODUCTION

Multirotor UAVs have been in development for decades and currently are being utilized in many different applications in the following areas; search and rescue operations, construction planning and surveying, precision agriculture and environmentalism efforts, and package delivery. This widespread utilization is largely due to several characteristics' UAVs are known for; from their cost-effective design in production, their limited impact to its surroundings or workspace, and their ability to gather information without putting active personal in physical harm or danger. Each of these industries have a unique impact on the future designs of UAVs as a result of a culmination of their application requirements. Today there are two primary classes of UAVs. These two classes include fixed wing and rotary wing vehicles. Fixed wing UAVs utilize a rotor for forward thrust and letting air foils generate the necessary lift. While rotary wing UAVs, also called multirotors, have between one and eight rotors to generate vertical lift [1].

Figure 1. Fixed wing UAVs.

Figure 2. Rotary wing UAVs.

Of this subdivision of rotary wing vehicles, vehicles that have four rotors are called quadcopters and will be the focus of research in this paper. Quadrotors have become the standard platform for many commercial and research applications. This is largely attributed to their simplicity, cost, and high manoeuvrability [2]. In this work an AR Drone 2.0, manufactured by Parrot, will be used, pictured in Figure 3. The AR Drone 2.0 is equipped with several additional sensors that make this model suitable for laboratory research.

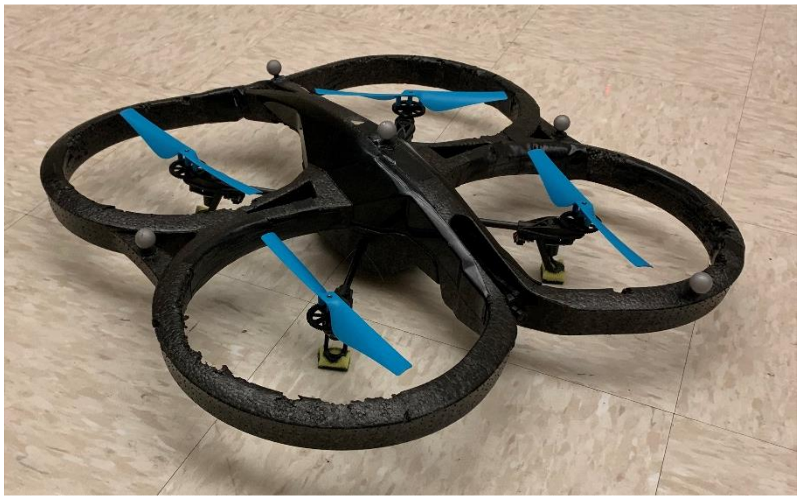


Figure 3. AR. Drone 2.0

The sensors that the AR Drone 2.0 is equipped include three gyroscopes, three accelerometers, three magnetometers, a pressure sensor, an altitude ultrasonic sensor, and a vertical camera. Although all these sensors may be used to develop autonomous navigation the primary focus of this work will utilize the vertical camera. More accurately the real time captured video data will be utilized in an object detection-based object tracking.

The multirotor UAV must locate an object within the camera frame and track the object by providing relevant control information with the end goal of remaining fixated on this object. This task, for a human operator or pilot, may become difficult if the object tends to move erratically or if the object follows an unpredictable pattern resulting in poor tracking performance and therefore wasted motion reducing flight time.

One elective technique autotomizes this task for a multirotor platform is to use a neural network. Neural networks as are a cost-effective solution that can utilize nothing more than the sensors currently available to the AR Drone 2.0 and the use of a PC/mobile device for computation.

Before detailing the construction and implementation of a neural network for object tracking with a multirotor UAV a few regulatory considerations with respect to autonomous flight should be noted. The Federal Aviation Administration, FAA, in 2016 implemented policy 14 CFR § 107 regarding remote pilots and commercial operators. Specifically, all autonomous operations must be overseen by a licensed remote pilot who can override autonomous controls in the event of an unknown situation. Although these provisions pertain to outdoor FAA controlled airspace it shall serve as a safety baseline for indoor laboratory operations.

1.1. Background

In the last decade there have been a variety of research efforts to develop neural networks for use with UAV systems. Some of these works focus primarily on the theoretical development with simulated manoeuvres being used to validate their performances [3]. Whereas numerous other research projects, developed within the last two years, implemented their networks for indoor multirotor UAVs to real world testing [4], [5], [6], [7]. These works focused their attention to shorter manoeuvres that spanned the length of indoor hallways. With many of these models showing promise for sustained flight and therefore greater benchmarks with respect to distance goals. One team has moved its system beyond the lab to outdoor environments in an endurance test spanning a distance greater than 1 km [8].

In all these papers the focus of the neural network was to classify incoming video data into three output situations (left, right, and center), where the position represents the directed path forward for the UAV. The UAV would follow these commands to determine how to best proceed while other system variables like elevation would dynamically be held constant. In each case of the studies presented the benchmarks focus only on a binary result, pass or fail, for a maneuver. To offer an alternative to this line of thought the fitness of a network could also be judged on the accuracy or tracking error taken from an external source. This external validation can provide an in-depth unbiased approach for the comparison of the fitness of a neural network.

1.2. Thesis Goals and Outline

The goal of this thesis is to discuss the development of a neural network-based object detection for position control of a multirotor for experimentation on autonomous target tracking.

Chapter 2 provides an in-depth background into neural networks and contains the necessary steps needed to set up and train a neural network for experimentation in object detection for multirotor vehicles using an AR Drone 2.0 quadrotor. This section will detail the strategies used to collect training data, how to effectively train a neural network using training data, and some of the potential sources of error to understand and to avoid. Then, the chapter will provide a validation study of the presented methods.

Chapter 3 outlines and describes the indoor multirotor testbed in use in the ARTLAB at South Dakota State University. The chapter will go into the details of the existing communication networks utilized by the testbed and how control commands are relayed to the UAV. Then, the chapter will discuss the integration of a trained neural network into the indoor multirotor testbed for object detection and tracking.

Chapter 4 describes the experimentation of object detection and target tracking first with a simulated multirotor model. Then, the chapter will provide a description of an integrated neural network control system with the Indoor Multirotor Testbed.

The last chapter is attributed for a conclusion and recommendation on this thesis.

CHAPTER TWO

NEURAL NETWORK CREATION, TRAINING AND VALIDATION

2.1. Neural Network Theory and Notation

Neural networks, or specifically their base components, were first conceptually constructed in the late 1950's as a hypothetical means of understanding the biological structures of higher organisms with regards to how information is processed, stored, and influences current decision making [9]. A direct representation of a biological neuron was then developed, known as a perceptron, to create a connection to simple computer logic, see Figure 4.

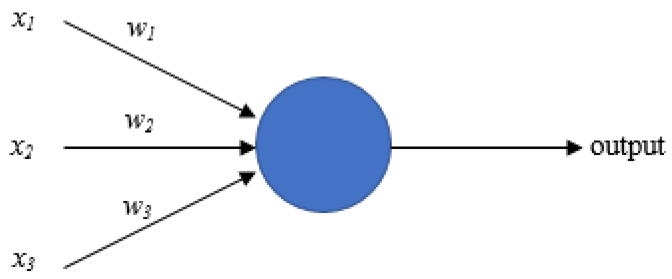


Figure 4. Perceptron representation.

From the single perceptron introduced by Rosenblatt, weights were developed; w_1, w_2, and w_3, which could then be assigned to each connection into a perceptron to represent the importance of that connection from another perceptron. In particular, a large weight could denote that a specific input holds a greater merit to influence a decision over a different input. Therefore, for each of these connections imagine that a weight is assigned. Now present a series of input arguments; x_1, x_2, and x_3, to the connections. The resulting sum of the products of the weights and inputs results in a finite output or outcome for the perceptron. Numerically, any such outcome could be judged by the following relationship:

$$output_i = \begin{Bmatrix} 0, & \sum_j x_{ij} w_{ij} \leq -b_i \\ 1, & \sum_j x_{ij} w_{ij} > -b_i \end{Bmatrix} \tag{1}$$

Where b_i is an associated bias or threshold for a decision to be made. The indices i and j represent the perceptron number within a network and the connection input to the perception respectively. The bias can be thought of, for example, the importance of the decision to be made. Where a large bias could be used to limit the ability of a perceptron to become active.

Continuing from this simple notation Nielson provides an exhaustive derivation of how complex perceptron systems coordinate into simple matrix algebra [10]. The above notation can be rewritten such that all terms are greater than or less than or equal to zero by moving the bias term to the left-hand side and setting the right-hand side equal to zero.

$$output_i = \begin{Bmatrix} 0, & \sum_j x_{ij} \cdot w_{ij} + b_i \leq 0 \\ 1, & \sum_j x_{ij} \cdot w_{ij} + b_i > 0 \end{Bmatrix} \tag{2}$$

This description for perceptrons, primarily its behavior, can be imagined as logic functions such as; AND, OR, and NAND functions. For example, a perceptron may behave as a logic function by first setting an acceptable input of 0 or 1 for each input connection and by assigning a set weight for each connection and a bias for the perception. For the example shown in Figure 5, the weight for each connection is set to a constant -2 and the bias is set to a constant 3. If inputs of 1 and 1 are implemented than the following equation develops:

$$(-2) * 1 + (-2) * 1 + 3 = -1 \tag{3}$$

As this number is negative the actual output of the perceptron is 0. This behavior is an example of a NAND gate, where two active inputs, (1, 1), result in an inactive output 0.

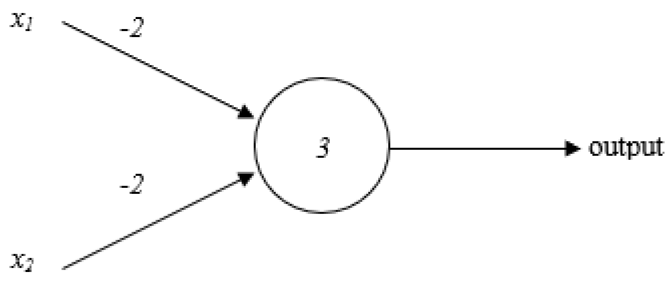


Figure 5. NAND Gate Analogy.

For all perceptrons in a network it may be more useful for the output to be in a continuous output form. This is done to allow for greater flexibility in the output of a perceptron as many situations may require a higher level of fidelity than a binary output system. So, to separate the logic from using a Boolean output a perceptron could use the following format:

$$output_i = \sum_j x_{ij} \cdot w_{ij} + b_i \tag{4}$$

And the *output* is therefore represented by any real number.

It is from this basic structure that a fully constructed architecture may be created to generate what is known as a neural network. Using Figure 6 as an example, we can see a single layered neural network comprised of an input layer, a single hidden layer with four perceptrons, and an output layer. Neural networks can be considerably larger than this representation but quickly become complex to display in the given 2D representational form.

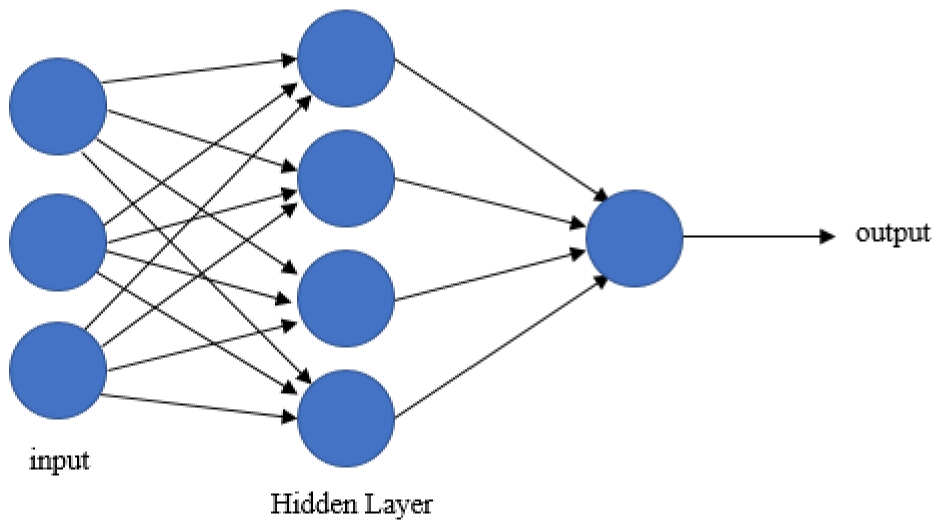


Figure 6. Hidden perceptron layer.

The input layer can be scaled to match a desired input source including; sensor data, known state variables, or even time. On the other hand, the output may be scaled to match a desired list of expected outcomes or targets. As the input and output layer design is set by a situation the design of the hidden layer is left open to more creative interpretation. The hidden layer may consist of several or more layers with any number of perceptrons in each layer. With the general sense being that for more complex and dynamic scenarios, more hidden layers or a larger number of perceptrons may be needed to accurately model a system. Although, it should be noted at this point that larger networks are necessary to solve complex problems but, depending on additional system constraints (like computational processing time), large networks may not be appropriate.

Now that the structure of a neural network has been described the next substantial step is implement a learning algorithm within the network. To address this step, one should first ask to suppose that a large incremental change is made to either a weight or bias value in a perceptron. This can subsequently result in the output of the network of a perceptron suddenly increase by a large value drastically effecting the next perceptron in the network and the overall behavior of the entire network. To correct this behavior a perceptron may be modified to limit the effects that a large weight or bias change can have on each perceptrons output. Thereby constraining a perceptrons output within a set range. This new type of neuron is called a sigmoid neuron.

A sigmoid neuron, is denoted by the following:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{5}$$

Where *z* is the representation of the function from a perceptron's output as in Eq. (4):

$$\sigma(z) = \frac{1}{1 + exp\left(-\sum_j x_{ij} \cdot w_{ij} - b_i\right)} \tag{6}$$

This results in the following plotted function as shown in Figure 7.

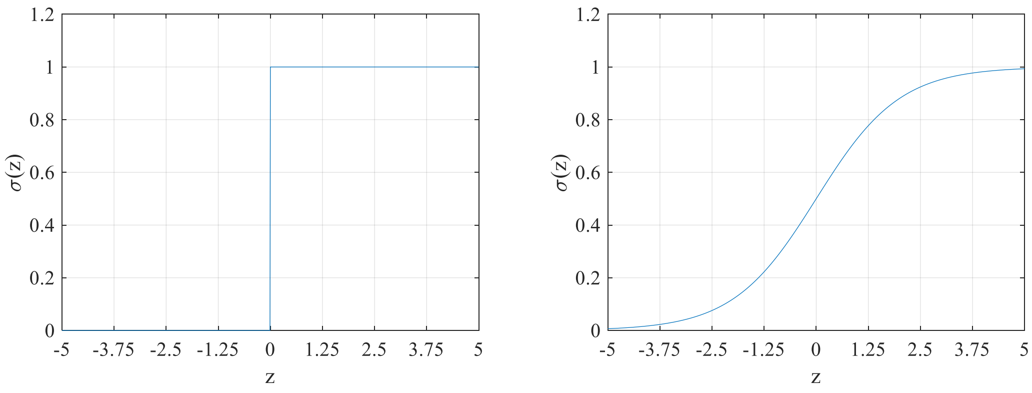


Figure 7. Step Function Eq. (2) (left) and Sigmoid Function Eq. (6) (right).

Where by design, the shape of this function is just a modified version of the Boolean step function mentioned earlier, but retains the benefits of a continuous output bounded between 0 and 1. From this function, finite changes in a weight or bias of a neuron will result in a small finite change in the output of the neuron as the output of the function is now bounded between 0 and 1.

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b_j \tag{7}$$

The use of sigmoid neurons prepares the groundwork for the learning process of a neural network. The next few sections will break down the learning process of the network.

First, we must introduce the concept of training inputs denoted by the value, *x*. Where *x* is a dimensional vector provided as an input into a neural network and such that an output variable, also a dimensional vector, denoted by *y* exists.

$$y = y(x) \tag{8}$$

To generate a learning algorithm, such that *y(x)* accurately approximates a given scenario for a given training input of *x*, a cost function denoted *C*, also known as loss function, may be used.

$$C(w, b) = \frac{1}{2n}\sum_{1}^{n} \|y(x) - a\|^2 \tag{9}$$

Where *n* is the total number of training inputs sets, *y(x)* is the actual vector output given an input vector *x*, and *a* is the actual vector targets given *x*. A target is a known or expected output that the neural network is expected to output given a specific input array. This method is best known as the mean squared error and the goal of using this method is to minimize a cost function through several epochs, or rounds, of training. This is best done using an algorithm known as gradient descent [11], [12].

In the gradient descent method, the gradient vector of the cost function denoted, ∇*C*, is used to determine, for lack of generality, the direction that each of our variables, weights and biases, should change. This total change of the cost function can be constrained within a small step usually called learning rate, denoted by η, such that:

$$\Delta v = v - \eta \nabla C \tag{10}$$

Where *Δv* is a representation of m variables for the cost function, *C*.

$$\Delta v = (\Delta v_1, \dots, \Delta v_m) \tag{11}$$

Removing the generality of the above equations we can represent the weights and biases in the following manner:

$$\Delta w = w - \eta \frac{\partial C}{\partial w} \tag{12}$$

$$\Delta b = b - \eta \frac{\partial C}{\partial b} \tag{13}$$

But this is not a complete notation as described earlier the cost function was stated as an average. Without the use of this average notation one cannot partition out training data into several epochs and one key issue can occur. When processing a large sample of training data to minimize the cost function and therefore the weights and biases, it will be incredibly slow. To mediate this issue all training data may be divided into smaller epochs of number m samples with training inputs of X_m.

$$\Delta w = w - \frac{\eta}{m} \sum_{j}^{m} \frac{\partial C_{X_j}}{\partial w} \tag{14}$$

$$\Delta b = b - \frac{\eta}{m} \sum_{j}^{m} \frac{\partial C_{X_j}}{\partial b} \tag{15}$$

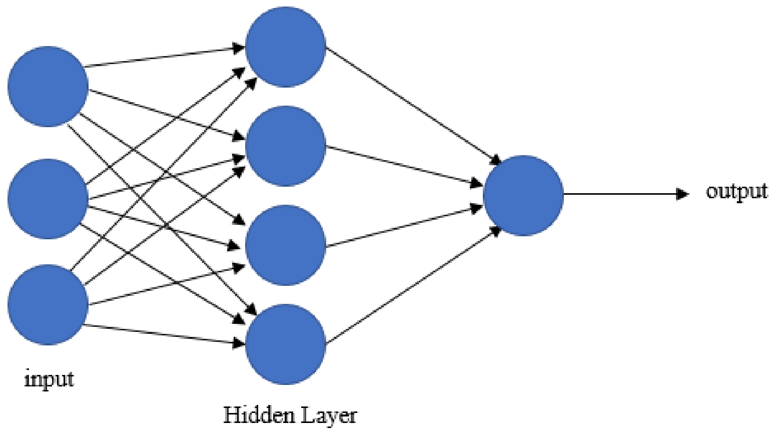
The current structure presented thus far is known as a feedforward neural network, where one layer only feeds into the next layer. Alternative networks, to the feedforward networks, have been previously presented including recurrent neural networks [13]. Where cross layered connections exist between layers creating feedback loops within the network. In other research papers comparing the two models' recurrent networks tend to outperform feedforward networks but at the cost of increased complexity and computational resources, which depending on specific situational requirements may be unfavorable [14].

2.2. Development of a Neural Network-Based Object Detector

2.2.1. Matlab Deep Learning Neural Network Toolbox

The application used for most of this thesis utilizes Matlab and Simulink developed by MathWorks®. Matlab supports a powerful toolbox known as the Deep Learning Neural Network Toolbox. This toolbox can be used for classification, regression, image feature learning, time series, and text data. For the efforts completed in this thesis a shallow neural network will be utilized for situational pattern recognition and target classification.

Although the title of the toolbox claims to use deep learning, this is a misnomer to some of the capabilities within the application. For instance, only a shallow neural network may be used. A shallow neural network is denoted as such due to it characteristic of using a single hidden layer of neurons. Deep learning neural networks are mostly characterized by several hidden layers. The use of a shallow network can introduce several benefits that may be leveraged to improve the efficiency and speed at which neural networks may operate. To quickly detail why a small network boasts greater processing speeds and training rates look to the total number of variables within the cost function for the network. As the number of neurons or hidden layers increase the number of weights and biases exponential increases. For more numerically driven detail some research shows that for the same training data set a shallow neural network does not perform as well as deeper convolution network, but for even moderately complex functions they have great potential [15]. While additional work continues the topic of shallow networks and even compares the use of one to two hidden layer networks [16]. It was shown that two hidden layered networks are more prone to falling into local minima due to a more complex cost function, but otherwise perform similarly to single layered networks. It is from these findings that a shallow neural network is deemed enough in trial testing for this thesis.



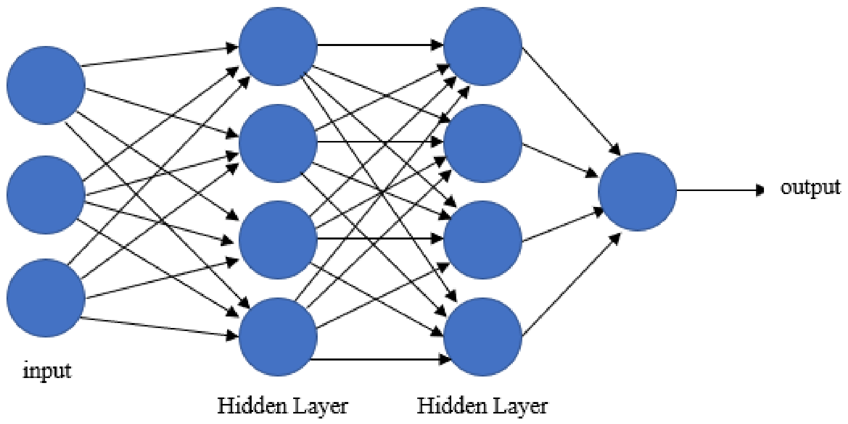


Figure 8. Shallow (top) and deep (bottom) neural network.

Lastly, the toolbox utilizes several methods for quantifying training progress including the use of figures, plots, cross entropy, and percent error. The use of figures and plots will be explored later in Section 2.2.4 Neural Network Validation. As for the numeric methods more explanation can be given. For several of the processes utilized within the toolbox to determine training performance one equation is readily used. Cross entropy is a measure of difference between our target value, a, and actual value, y, where larger differences are penalized more heavily and is given by the following equation:

$$Cross\ Entropy = a * \log y \tag{16}$$

2.2.2. Matlab Image Processing Toolbox

Another toolbox that will be utilized is the Image Processing Toolbox. Within this toolbox a series of essential functions to stream, process, and convert usable image data are utilized. Images taken from a data source, like a drone camera, can be resized into different resolution formats. While other properties of the image may be changed to meet user stated system requirements.

In the case of this thesis, the AR Drone 2.0 utilizes a HD camera with a resolution of 720p, 1280×720 pixels, and a frame rate of 30 fps. At the current resolution (720p), if utilized as an input to a neural network, would demand an input layer size of 921,600 neurons. Where each pixel of the image would equate to an input neuron within the neural network. When considering the processing time, to train or calculate real time network outputs, it may be necessary to resize the image stream to a smaller resolution.

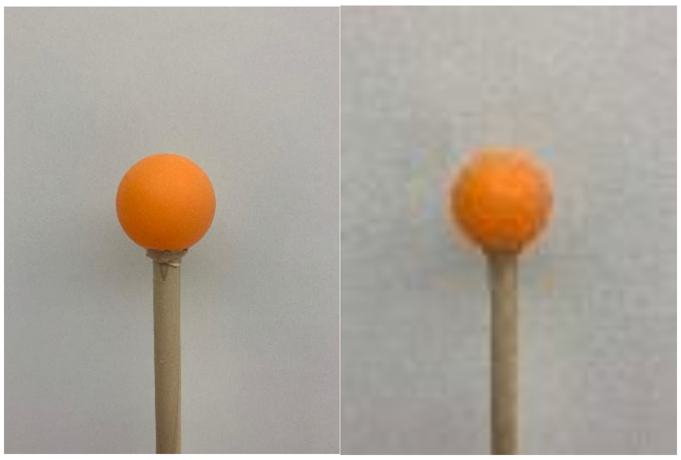


Figure 9. Image Resolution 3024x4032 (left) and 76x101 (right).

When scaling an image, like the example in Figure 9, the general concept is relatively unchanged to human eyes, but for a neural network the intension of the object may greatly change. To counteract this effect good engineering judgment should be made to determine if the new scaled resolution adversely effects the scenario presented to the neural network.

The images taken from the UAV may then be processed further for use in a neural network to meet user set system requirements. With the use of a Matlab created neural network, its first layer, or input layer, must consist of a singular numerical value to represent each associated input neuron. Two possible solutions may be used as the images taken from a drone's camera are a 24-bit TrueColor RGB, consisting of 8-bit red, 8-bit green, and 8-bit blue. The first option is to implement each color value for each pixel as an input to the neural network, but this will generate an input layer three times the size of our second option. The second option is to use the command rgb2gray() within Matlab. This command will convert each image to an 8-bit image without color information like chrominance. The new image will now consist of pixels having a value base of 256 combination. Where a value of 0 represents black pixel and a value of 255 represents a white pixel. The function uses the following equation to convert each RGB pixel.

$$I = 0.2989 * R + 0.5870 * G + 0.1140 * B \tag{17}$$

For the work within this paper the second option will be utilized.

Where I is an outputted 'gray' pixel between 0-255. Values R, G, and B are the representative 8-bit color value, between 0-255, for red, green, and blue respectively. Figure 10 shows an example of this conversion process.

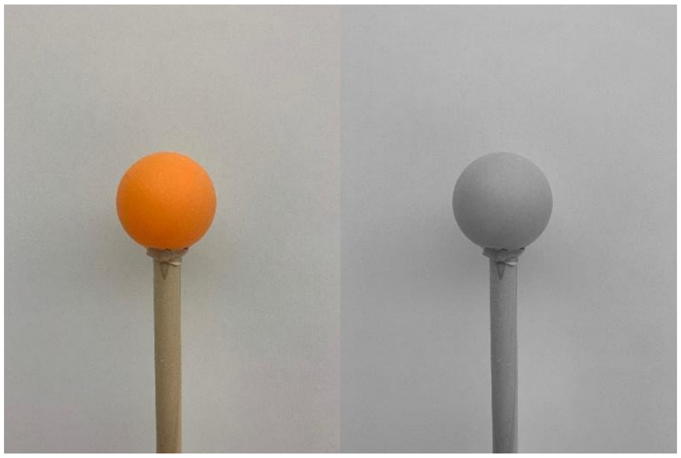


Figure 10. RGB image (left) and gray image (right).

As mentioned previously when rescaling an image, by converting an image to grayscale care must be taken to change the intention of the training data.

2.2.3. Neural Network Training

To train a network for a scenario, training data must first be gathered. Depending on the complexity of the needed network, several thousand samples of training data should be collected, while recording their expected output, to be paired with each sample. For convenience, the same camera that the system will be utilized on, will be used to collect these samples.

For the experiments later discussed in CHAPTER F all training data will be collected using the AR Drone 2.0 front facing camera, as pictured in Figure 11.

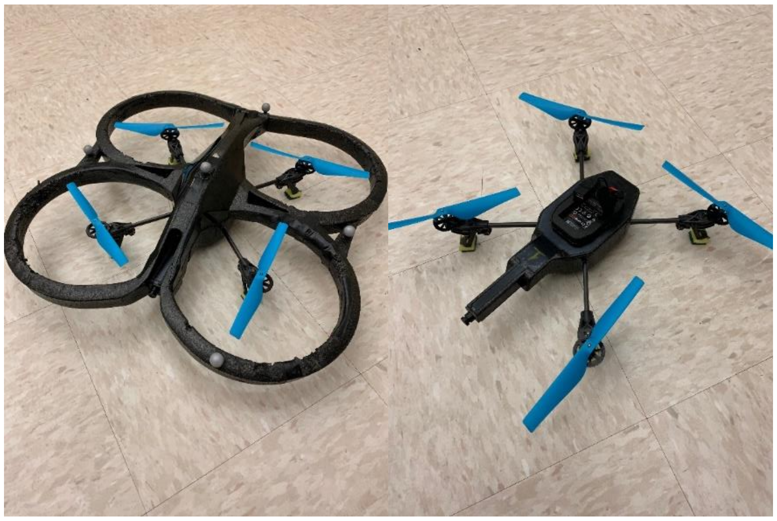


Figure 11. AR Drone 2.0 Covered (left) and uncovered (right).

For the collection of training data, video data from the UAV should be captured. A USB storage device may be connected to the USB port located within the AR Drone’s battery compartment. Then, utilizing the AR.FreeFlight mobile application, available in both Android and IOS app stores, the camera recording features may be controlled. Figure 12 shows a breakdown of the application’s window view.

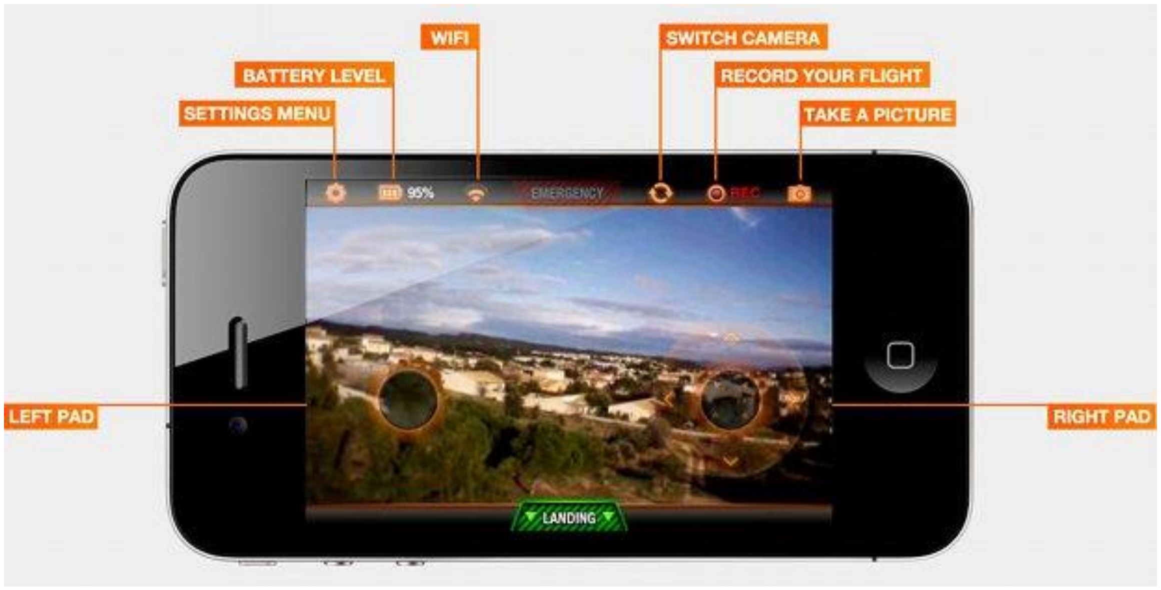


Figure 12. AR.FreeFlight mobile application.

Located in the upper right of the window, a recording button can be used to record videos of the flight. For a recording all frames within the recording should be limited to an intended output scenario. For example, if the intention for an object detection is to determine whether an object is on the right or left side of the video frame. Then, at minimum, two separate video files will be required where the object is mutually exclusive to the left or the right side of the video frame. When all videos are taken they may be converted to image data using a Matlab script available in APPENDIX I; A.

Once converted, the images for each case may then be converted to a singular file format known as a comma separated variable, .csv. Each individual image is converted to a single row within a master matrix. Where the first column of each row is the target or expected value of the corresponding image to the row, also known as the label. The number of rows within the matrix is equal to the number of image samples. An illustrative example of the .csv file type is shown in Figure 13.

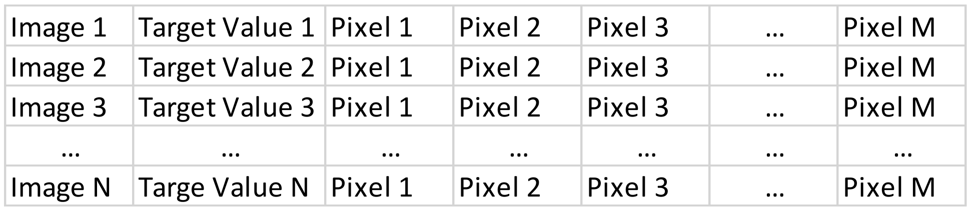


Image 1	Target Value 1	Pixel 1	Pixel 2	Pixel 3	…	Pixel M
Image 2	Target Value 2	Pixel 1	Pixel 2	Pixel 3	…	Pixel M
Image 3	Target Value 3	Pixel 1	Pixel 2	Pixel 3	…	Pixel M
…	…	…	…	…	…	…
Image N	Targe Value N	Pixel 1	Pixel 2	Pixel 3	…	Pixel M

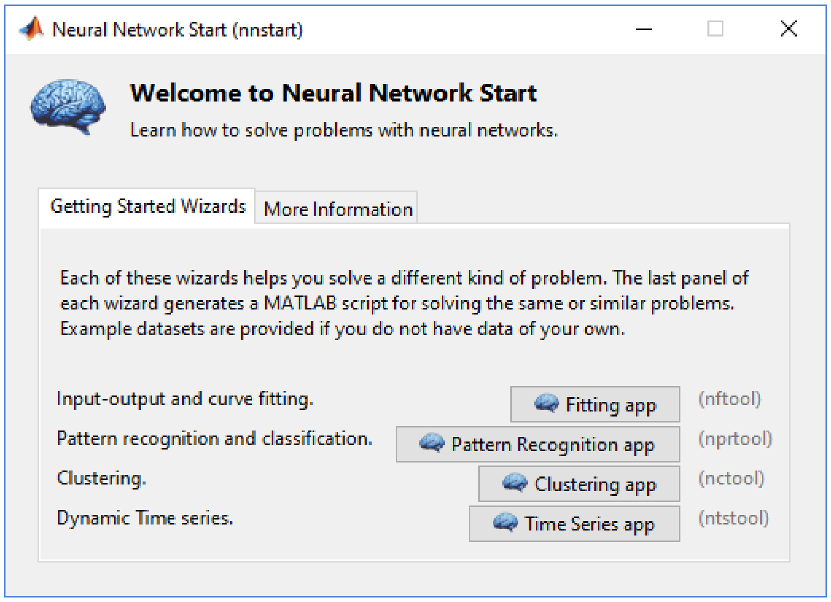
Figure 13. Example layout of a (.csv) file.

Where N is the total number of images and M is the total number of pixels within an image. The Matlab script for this is reported in APPENDIX I; B.

With all the training videos converted to comma segmented variables one final script will be needed to combine the files for use in the Matlab Deep Learning Neural Network Toolbox. While combining the rows of data, the data should be randomly rearranged to not mislead the training application due to grouping of consistent similar data to improve training efficiency. This is done using the random permutation function while maintaining the integrity of each row unchanged. The Matlab script for this is represented in APPENDIX I; C.

Finally, this training data may be implemented with the Matlab Deep Learning Neural Network Toolbox to train a shallow neural network. To begin the training process, an initiating Matlab script is needed to segment training data and to define variable names. This script outlining this process is in APPENDIX I; D. The data can be further segmented using a holdout subset. This holdout data is separated and stored for later use in a post training test of the neural networks performance and provides new unseen data for verification.

With the training data prepared and segmented, the training process within the Matlab Deep Learning Neural Network Toolbox may be started through the use a procedurally generated Matlab function known as nnstart. Within this toolbox prompt the Pattern Recognition app will be utilized for classification tasks, as shown in Figure 14. This will classify an image by outputting an array of probabilities for each output type. The value with the highest probability is the classification output of the neural network.


Neural Network Start (nnstart)
Welcome to Neural Network Start
Learn how to solve problems with neural networks.
Getting Started Wizards
More Information

Each of these wizards helps you solve a different kind of problem. The last panel of each wizard generates a MATLAB script for solving the same or similar problems. Example datasets are provided if you do not have data of your own.

Input-output and curve fitting.	Fitting app	(nftool)
Pattern recognition and classification.	Pattern Recognition app	(nprtool)
Clustering.	Clustering app	(nctool)
Dynamic Time series.	Time Series app	(ntstool)

Figure 14. Neural Network Start.

2.2.4. Neural Network Validation

To validate that the methods previously presented are correct, and to further outline the Deep Learning Neural Network Toolboxes capabilities and features, a trial experimentation can be conducted for a simplified scenario. For a descriptive example, imagine that a user requires a system to determine whether a yellow dot, or pin, is located on the left or right side of an images frame. To accomplish this a neural network may be used to perform an image classification task. As shown in Figure 15, a pin is stuck into the center of a piece of regular white paper.

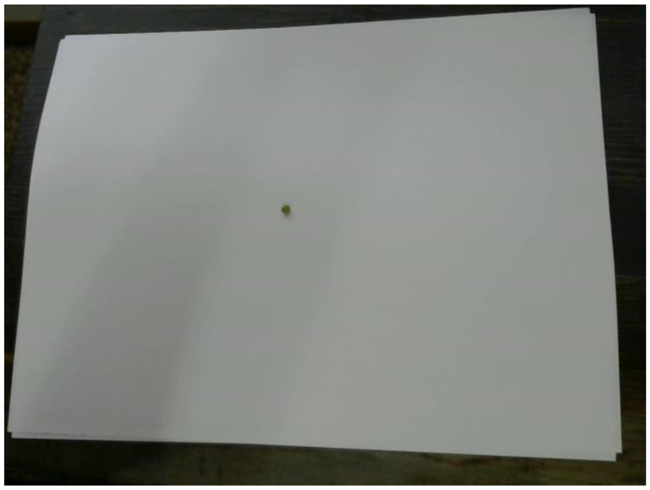


Figure 15. Validation object follow.

The white paper is used to create a neutral backdrop limiting background noises obstructions that simplify the scenario. To train such a network a large series of sample images are created and assigned target values corresponding to whether the pin is on the left or right side of the camera frame, see Figure 16.

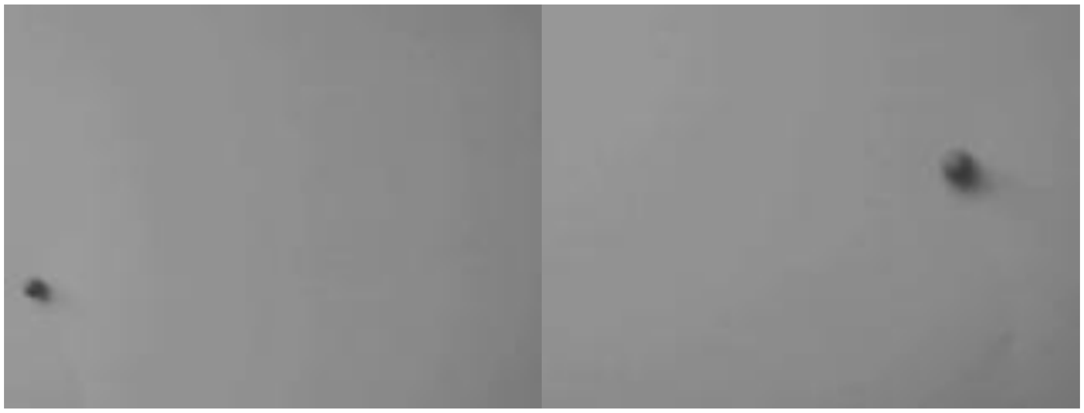


Figure 16. Target value of 1 (left), target value of 2 (right).

Note that the sample images are taken at a series of different heights or distances to the paper, therefore pin size will be independent of the neural network and not considered. To limit the size of the input layer of the neural network all images are scaled such that the current resolution is 80x60 pixels. This will leave the input layer size of 4800 neurons, where each neuron is associated to a pixel from the provided image data. As stated in the previous section all sample images are converted to greyscale formatting and converted to csv file formatting. The total number of samples collected are outlined in Table 1.

Table 1. Validation Samples

	Number of Samples
Test Images (80x60)	19662
Left	9831
Right	9831

An equal number of training samples were created for either of the two scenarios; left side of camera frame and right side of camera frame. Of these sample taken at random, one-third is reserved for later testing and accuracy confirmation. Leaving 13,108 samples for the network training process. Within the Matlab launcher the remaining samples are divided into the three necessary data groups, in a 70%, 15%, and 15% ratio, as shown in Table 2.

Table 2. Validation and Test Data.

	Number of Samples
Training	9176
Validation	1966
Testing	1966

These three categories are utilized by Matlab for various subtasks within the training process. The training category is used for computing the gradient and updating the weights and biases for the next epoch. The validation set is used in monitoring the training process and its progress. If the validation set, over several epochs, notices a trend of stagnant or increasing cross-entropy for this data set, the training process will stop. If training stops the weights and biases at the minimum validation set will be used in the final neural network. The final category is the testing subset and is used to provide an unbiased feedback of training effectiveness during the training process. If for example the cross-entropy error reaches a minimum at a significantly different epoch than the validation data set it might indicate an issue with the training process.

With the training samples divided, the construction of the network architecture must be completed by selecting the total number of neurons that will be utilized within the hidden layer. In this instance let us use 50 neurons and now train the network. The post training results are tabulated in Table 3.

Table 3. Training Results.

	Cross-Entropy	Percent Error
Training	6.11	2.73
Validation	1.63	4.02
Testing	1.64	3.81

From these results we can see that the final percent error for all three groups was below 4%. To better understand the training performance of the system several figures and plots may be utilized. At the final training epoch, the training confusion matrix, shown in Figure 17, can be utilized. A confusion matrix provides visual matrix representation of the total number of correctly and misclassified data. For each of the three subdivided training categories; training, validation, and testing, a separate confusion matrix is given, while a total confusion matrix combining the classes is given. For each matrix, plotted along the y-axis, the two actual output cases of the neural network are placed while along the x-axis the target output classes are denoted. Within each matrix the green squares denote the number of samples correctly classified to the target class and the red squares denote incorrect classifications. The bottom right square denotes the overall success of classifying the output classes.

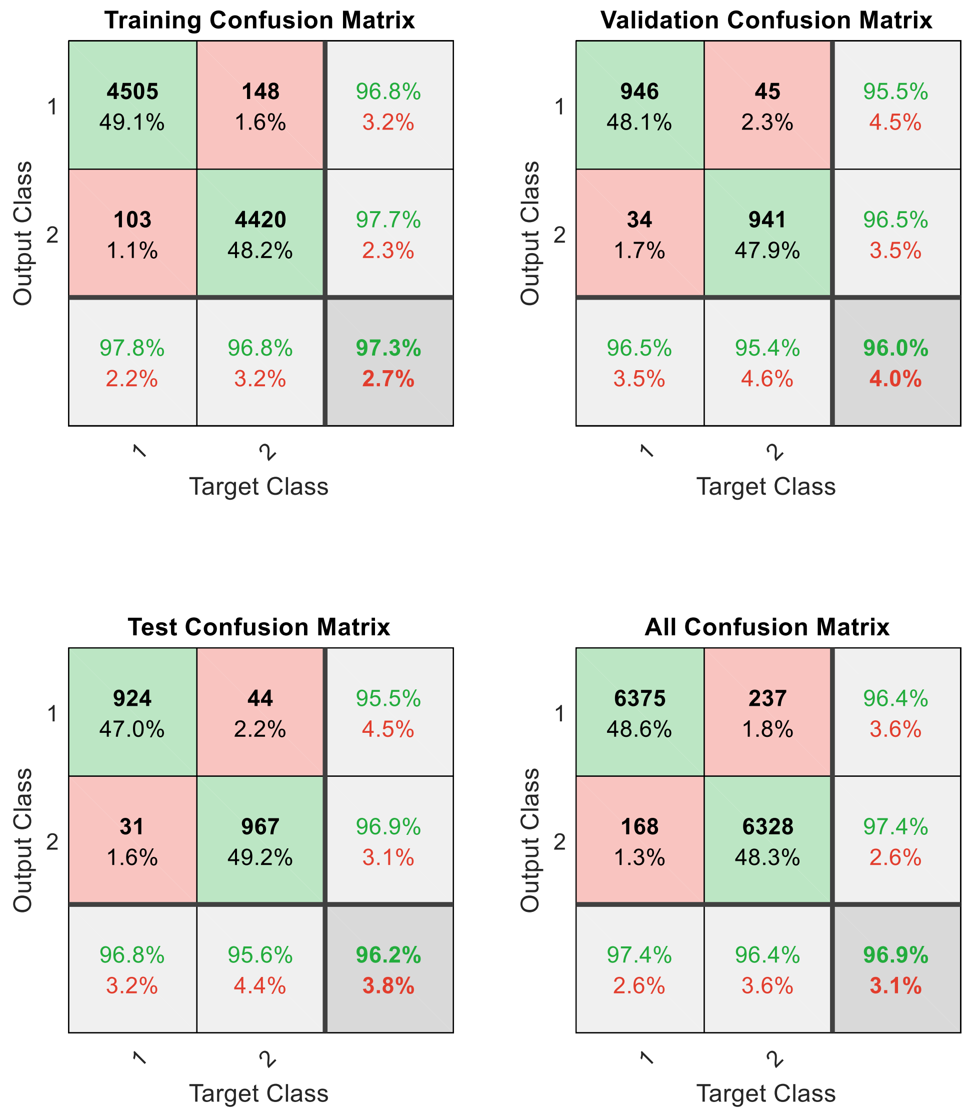
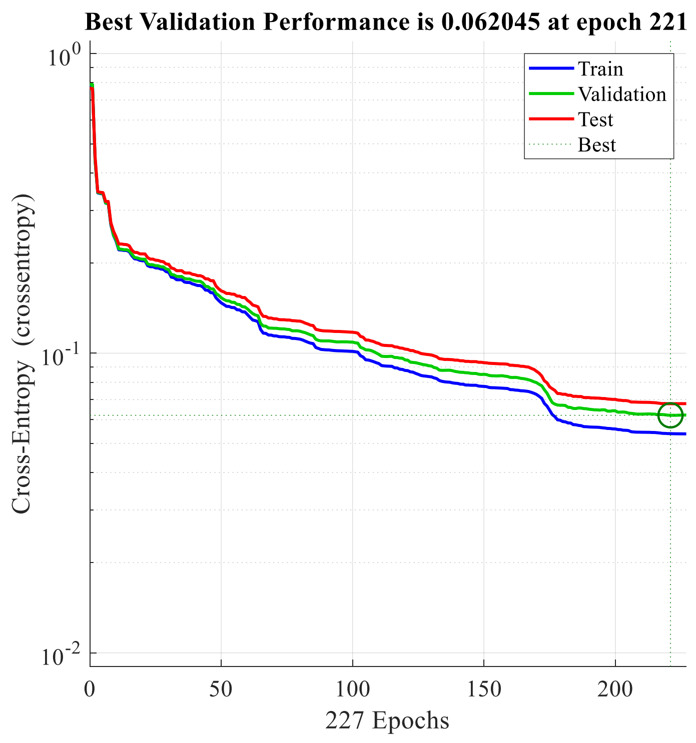


Figure 17. 50 Neuron Confusion Matrix

From here it was clear, that of the sample images used, a total of 405 samples or 3.1% failed placement across all three categories. If this is the case, then from the performance plot we can determine whether the training was stopped prematurely, or the current model is an accurate representation of a trained neural network for the given data. The performance plot is shown in Figure 18 and uses cross entropy as a measure of success.



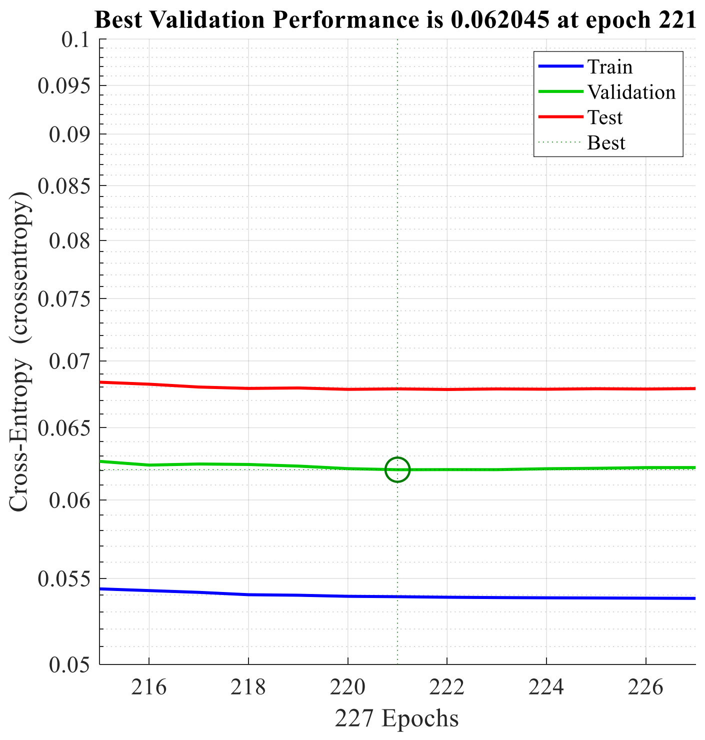


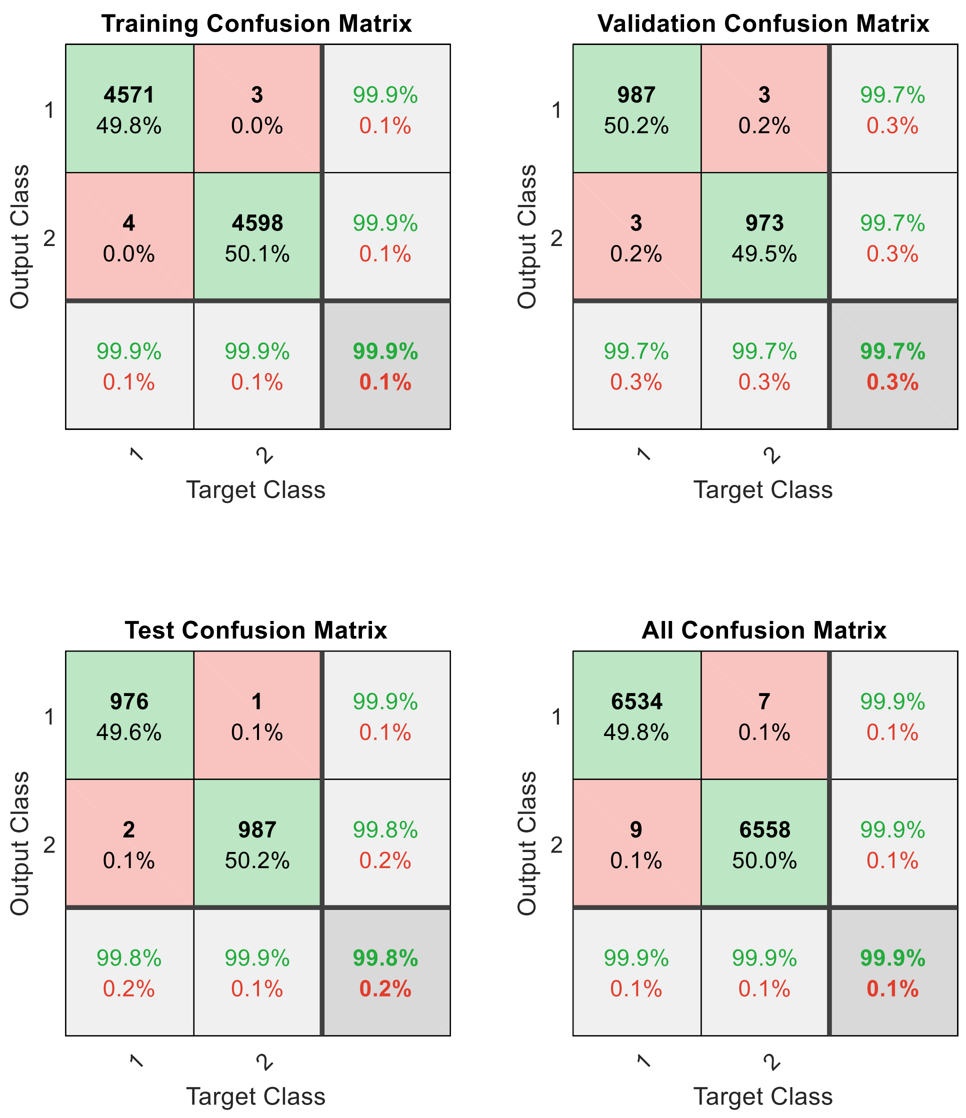
Figure 18. 50 Neuron Performance Plot (top) detail zoom (bottom).

After 227 training epochs the network had reached a converged solution when no significant changes occurred and that the cost function must have reached a local minimum. Therefore, the training reached a reasonable conclusion for the training data provided. If this is the case, then perhaps by changing the number of hidden layer neurons a better performing network may emerge. To test this theory, four trials were run and trained by varying the number of hidden neurons within the hidden layer between 10 and 200 while maintain the same training data. The results are shown in Table 4.

Table 4. Varying Hidden Neurons.

10 Hidden Neurons		50 Hidden Neurons	
Cross-Entropy	Percent Error	Cross-Entropy	Percent Error
1.22	0.07	6.11	2.73
3.48	0.31	1.63	4.02
3.48	0.16	1.64	3.81
100 Hidden Neurons		200 Hidden Neurons	
Cross-Entropy	Percent Error	Cross-Entropy	Percent Error
0.63	2.27	0.67	2.43
1.72	3.00	1.84	3.20
1.72	2.70	1.84	3.76

From these results a stark contrast is found between the four trials. In particular, the 10 hidden neuron network outperforms the others. The confusion matrix, shown in Figure 19, highlights this fact by showing that only 16 samples were misclassified across the three categories.



Training Confusion Matrix

Output Class \ Target Class	1	2	
1	4571 49.8%	3 0.0%	99.9% 0.1%
2	4 0.0%	4598 50.1%	99.9% 0.1%
	99.9% 0.1%	99.9% 0.1%	99.9% 0.1%

Validation Confusion Matrix

Output Class \ Target Class	1	2	
1	987 50.2%	3 0.2%	99.7% 0.3%
2	3 0.2%	973 49.5%	99.7% 0.3%
	99.7% 0.3%	99.7% 0.3%	99.7% 0.3%

Test Confusion Matrix

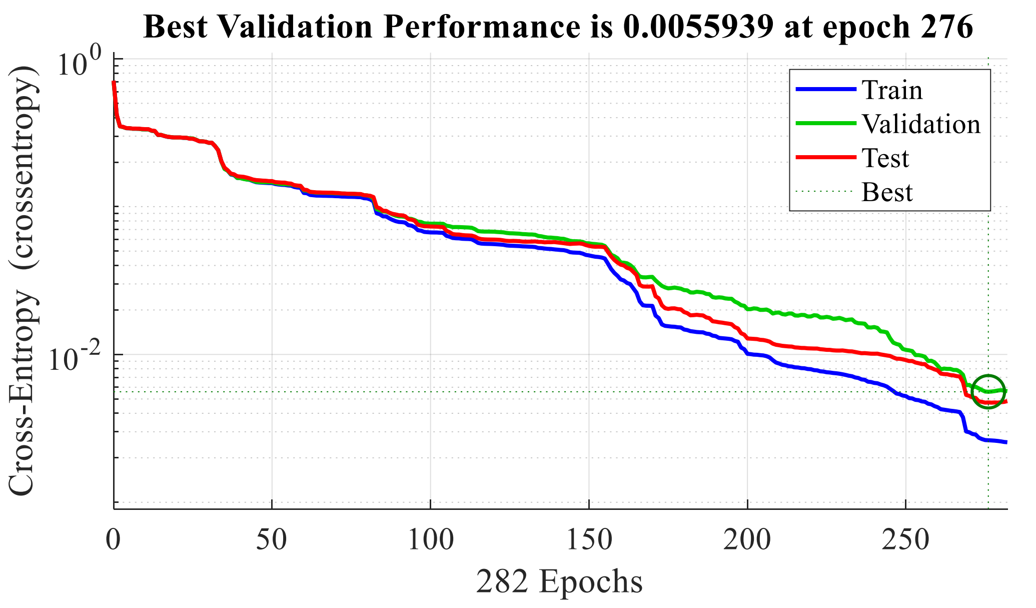
Output Class \ Target Class	1	2	
1	976 49.6%	1 0.1%	99.9% 0.1%
2	2 0.1%	987 50.2%	99.8% 0.2%
	99.8% 0.2%	99.9% 0.1%	99.8% 0.2%

All Confusion Matrix

Output Class \ Target Class	1	2	
1	6534 49.8%	7 0.1%	99.9% 0.1%
2	9 0.1%	6558 50.0%	99.9% 0.1%
	99.9% 0.1%	99.9% 0.1%	99.9% 0.1%

Figure 19. 10 Neuron Confusion Matrix.

To visualize how the training performance got to this solution we may check with the performance plot in Figure 20.



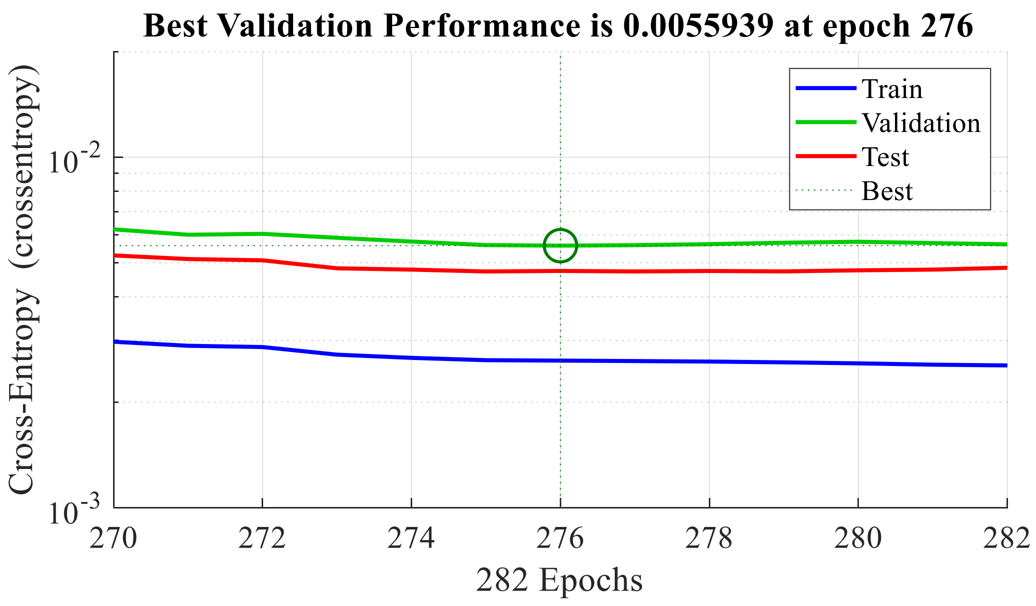


Figure 20. 10 Neuron Performance Plot (top) detail zoom (bottom).

In contrast to the performance plot for the 50-neuron case, training progressed after reaching several near convergence states, but eventually took another step to an even lower minimum. This demonstrates one of the effects presented earlier where smaller neural networks do not suffer from complex cost functions that are prone to getting caught in local minima.

From these networks we can now test their performances with never before seen held-out data from earlier. For the four cases this performance data can be seen in Table 5.

Table 5. Unseen data performance.

Hidden Neurons	Percent Correctly Classified
10	99.71%
50	96.35%
100	97.12%
200	96.57%

From these tests, the best performing network still occurs with the 10 hidden neurons confirming the training results.

As presented, the procedures for training neural networks for object detection have been demonstrated. These methods may now be applied to more complex object detection for use of object tracking with a multirotor UAV.

CHAPTER THREE

INTEGRATION OF NEURAL NETWORKS TO THE INDOOR MULTIROTOR TESTBED LAB

3.1. Introduction to the Multirotor Testbed

Within the last two years the development of a multirotor testbed was implemented in the Aerospace Robotics Testbed Laboratory (ARTLAB) [17]. ARTLAB is an experimental facility in the department of Mechanical Engineering at South Dakota State University. The multirotor testbed utilizes eight “Prime13” Optitrack Motion Capture System (MCS) cameras for the purpose of real-time tracking of position and attitude of a rigid body, for example the AR Drone 2.0, as shown in Figure 21.

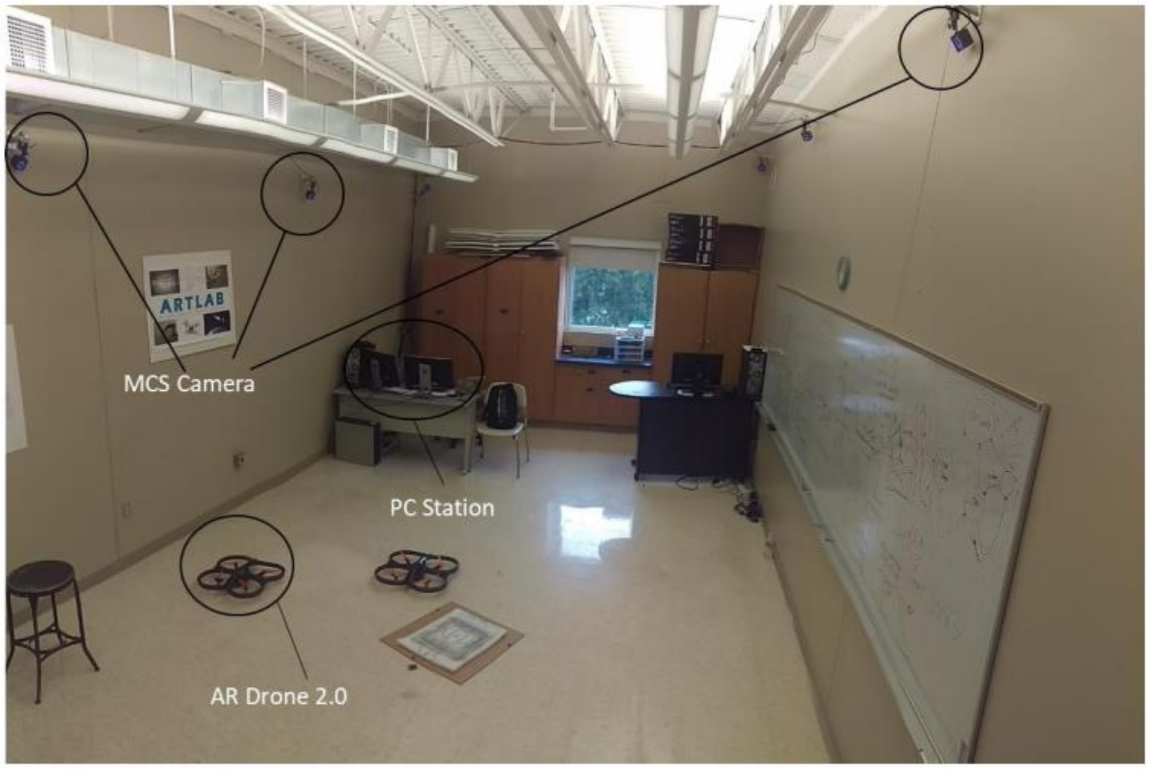


Figure 21. Aerospace Robotics Testbed Laboratory (ARTLAB).

The Prime13 cameras, shown in Figure 22, determine rigid bodies by tracking a set of retro reflective passive markers attached to a rigid body of interest. Figure 23 shows an example of the passive markers utilized by the multirotor testbed.

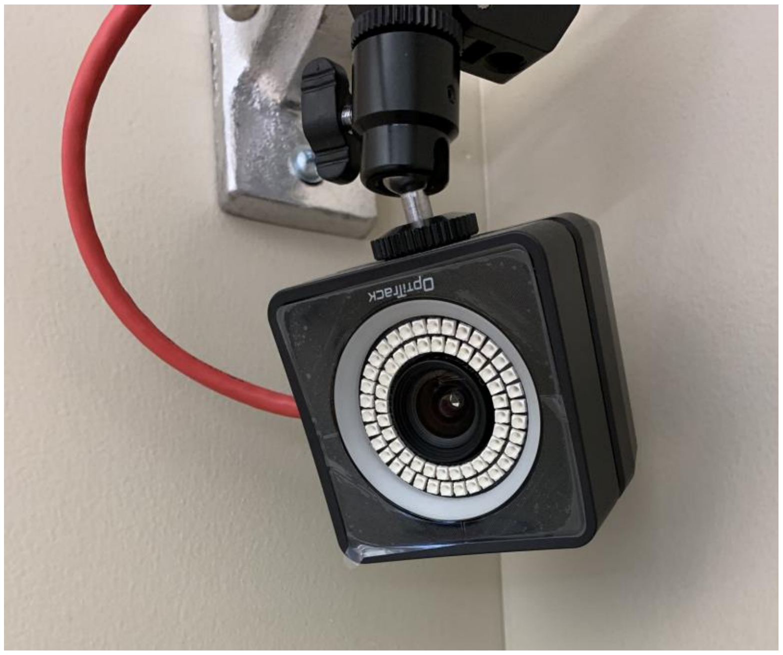


Figure 22. Prime13 Optitrack Motion Capture System.

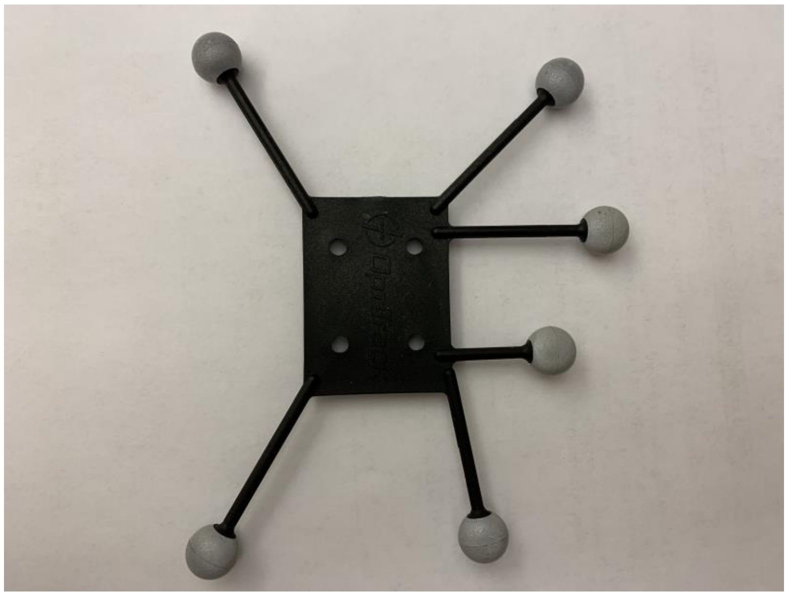


Figure 23. Passive Markers.

This relayed current state, in terms of position and attitude, of the AR Drone 2.0 is then streamed to a PC station running Motive: Tracker, an Optitrack proprietary application.

Within this application a group of markers may be associated to a single rigid body. Depending on the flight scenario, many rigid bodies can also be tracked simultaneously. Figure 24 displays the three-dimensional representation of the ARTLAB within the Motive software.

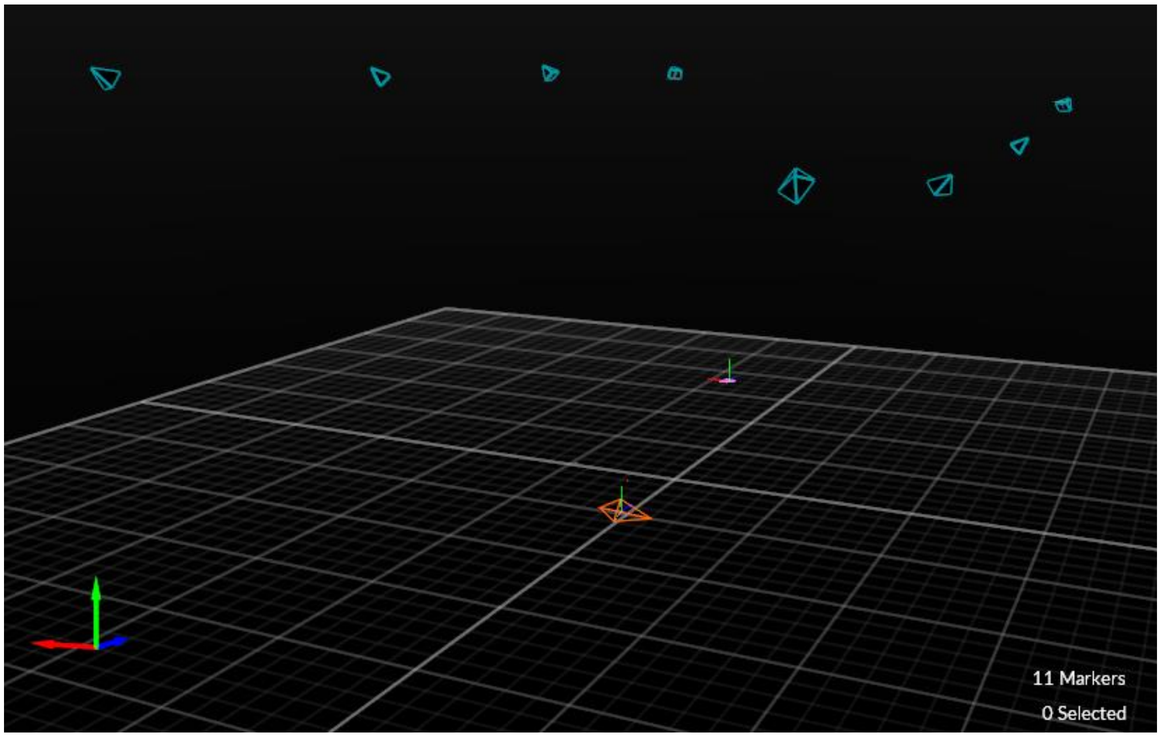


Figure 24. Motive: Tracker.

From this representation the position of the eight cameras, in blue, are shown along with two rigid bodies, shown in red. The two rigid bodies within this case include the AR Drone 2.0 and an object of interest.

From the Motive software the state conditions of the AR Drone 2.0 are streamed to Simulink using User Datagram Protocol (UDP), communication ports. The Simulink model will then process the information on current and desired drone position, for example using a Proportional-Integral-Derivative (PID) controller, to calculate and stream, via Wi-Fi connection, the required control signals to execute the maneuver. This process is pictured in Figure 25.

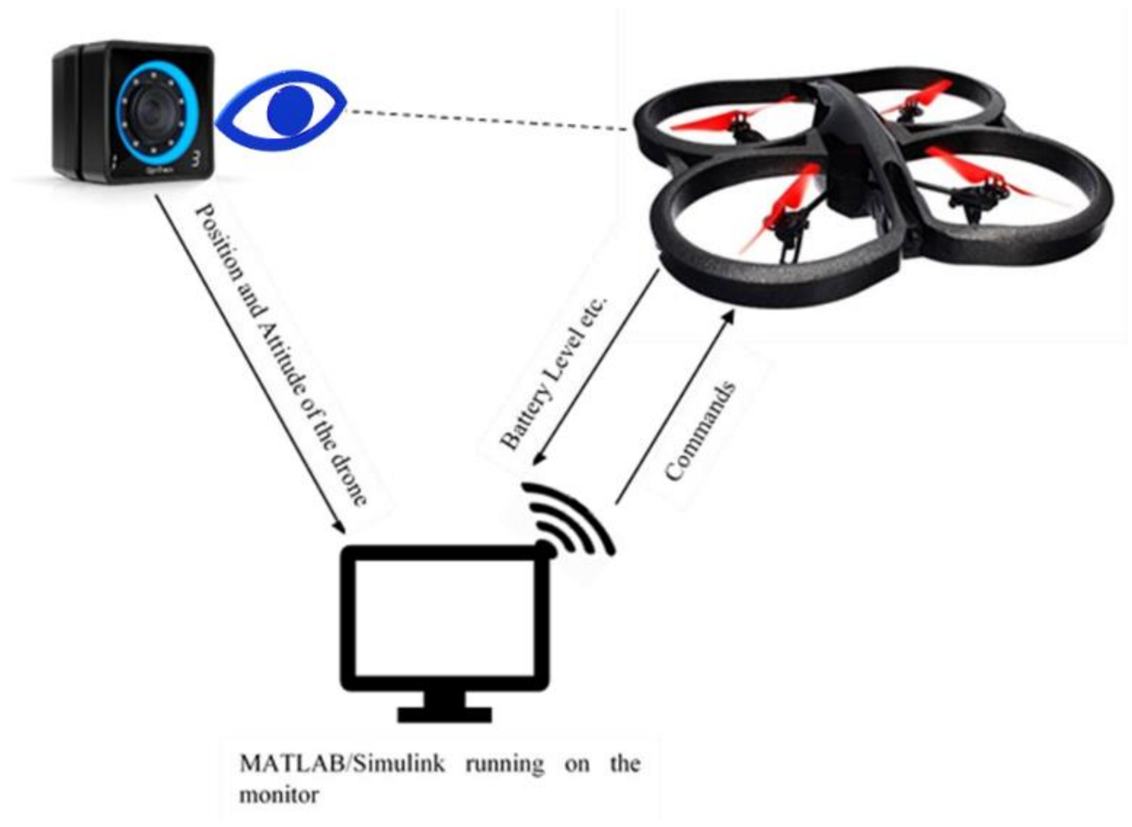

Position and Attitude of the drone
Battery Level etc.
Commands
MATLAB/Simulink running on the monitor

Figure 25. Multirotor testbed structure.

In reference to this testbed structure, a neural network may be implemented to both accurately track and command a UAV indoors. This is done by modifying the current multirotor model and implementing a neural network subsystem within the control framework. The neural network subsystem may then utilize image data from the multirotor and output relevant command data to the PI controller.

3.2. Integration of Neural Networks to the Multirotor Testbed

With an understanding of the multirotor testbed’s functionality and the corresponding communication protocols outlining the connections between the model components a neural network may now be inserted. Since the neural networks are developed and trained within Matlab/Simulink, they must also be implemented in the same environment.

3.2.1. Obtaining the Image Data Stream

The first step in obtaining image data from the AR Drone 2.0, is to connect the PC workstation to the Wi-Fi signal transmitted by the UAV, when powered on. When the AR Drone 2.0 is turned on, an active video stream is actively uploaded to an IP address and port. The video stream utilizes a video compression format known as H.264 or MPEG-4 and is set by the UAV manufacturer. Because this video format is currently not a supported file type within Matlab/Simulink, to obtain this video stream, a separate programing environment should be utilized to connect to the video stream and then export image frames to be streamed to Simulink for processing or use. For this purpose, Python will be used, see APPENDIX I; E for the script. From this Python script, an image stream is captured, processed to grayscale, and sent to Simulink using Transmission Control Protocol (TCP) communication through the local IP address and an open port. Further image processing will need to occur before this image stream may be utilized by a trained neural network.

3.2.2. Processing Image Data in MATLAB-Simulink

Through the process of converting and streaming images to Simulink, the image becomes mirrored about the vertical direction and the image is rotated by 90 degrees, see Figure 26 for reference.

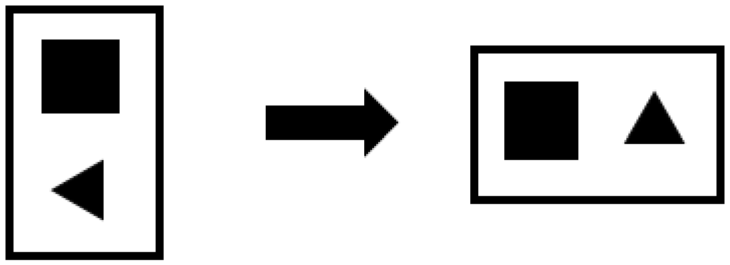


Figure 26. Image Rotation.

To execute this operation the image may be passed through a Matlab Function block to correct the image orientation by taking the inverse of the image matrix. Once the image is properly oriented, it image must be converted to a single row array using a reshape function like that used in the training image conversion process. From this array data type, image data may be passed through a developed and trained neural network to generate useful command data for a multirotor UAV.

3.2.3. Neural Network Model Integration

Within Simulink, a neural network block may be imported in the control model as shown in Figure 27. Where the AR Drone 2.0 wirelessly streams image data to Matlab using the previously presented Python script. The image is then processed and sent through the neural network. The output of the network can then be used, within the multirotor controller, to determine a new commanded state for the multirotor to move to. The Optitrack system may be used to provide a third-party observer for comparison of neural network performance.

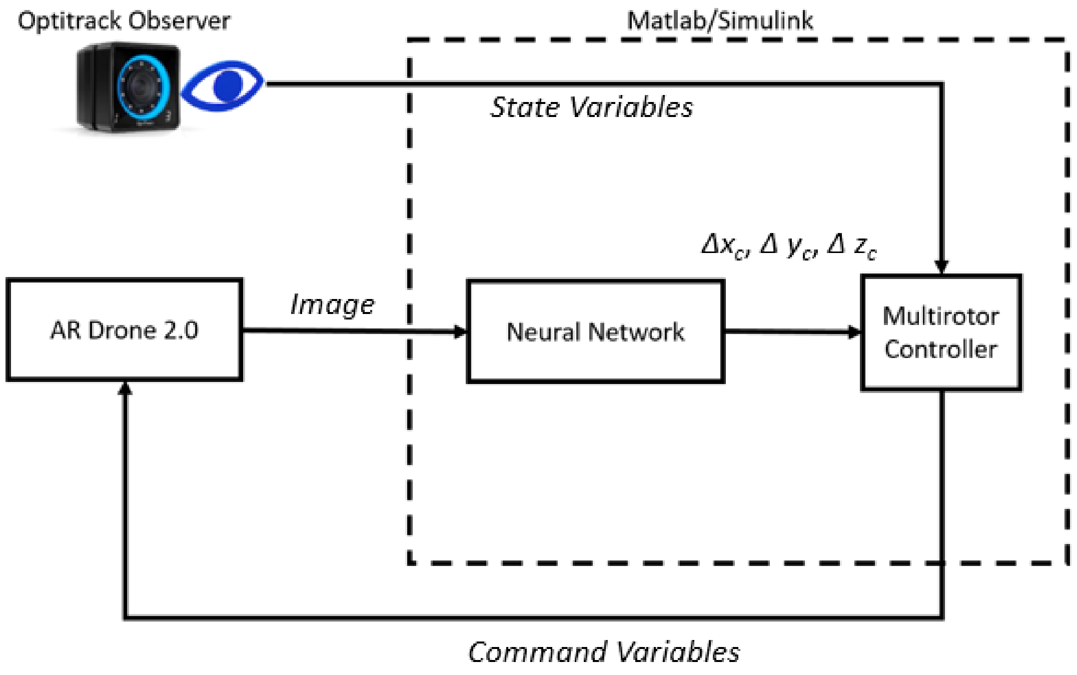


Figure 27. Schematic of the Neural Network-Control Integration.

CHAPTER FOUR

TESTBED EXPERIMENTATION FOR OBJECT DETECTION AND TARGET TRACKING

In this chapter a multirotor UAV in conjunction with the multirotor indoor testbed will be used to locate and follow an object. For this purpose, a tennis ball will be used as an object for the multirotor to detect. The tennis ball has been chosen for its generic size and uniform edges. Although, for all scenarios explored in this work, the transmitted images used in the neural networks are in a grayscale format the color of the tennis ball provides a sufficient color differentiation between the object and the background wall colors. The ball is free hanging from the ceiling as shown in Figure 28 to form a free-floating object. Above the object a six-marker object is fastened. This marker object is used so that the multirotor testbed may locate and create a rigid body to represent the object for a variety of testing purposes using the markers as a control scenario. When initiating an experimental maneuver, the marker object will allow the Indoor Multirotor Testbed to autonomously center the multirotor for experimentation of the neural network controller. Once the UAV has reached a set location within reference and the tennis ball within the field of view of the on-board camera, all commanded variables will then be controlled by the neural network only.

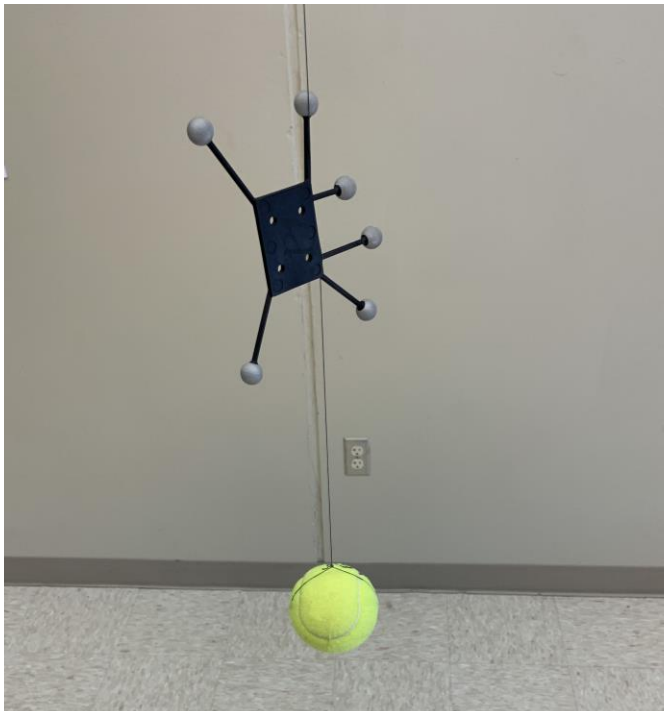


Figure 28. Object Follow Tennis Ball.

For the following sections of this chapter, three separate neural networks will be developed for three cases. To determine if the object is; on the left or right side of the image frame, on the upper or lower side of the image frame, and if the object is near or further away.

4.1. Neural Network Development and Training

Using the procedures outlined in section 2.2, three neural networks may be trained to accurately determine the location of an object within the drone's camera frame. For each of the three cases the general structure of the neural network will remain constant. The input layer will be comprised of 9,216 pixels or the equivalent of an image resolution of 128x72. A single hidden layer will be used comprised of 100 neurons within the layer. Contrary to the validation case, which showed that 10 neurons within the hidden layer had the best performance, 100 neurons will be used to accommodate the increased complexity of the scenario. And finally, for each case only two outcomes will be available such that the output size of the network is two. Figure 29 shows the Matlab representation of this structure.

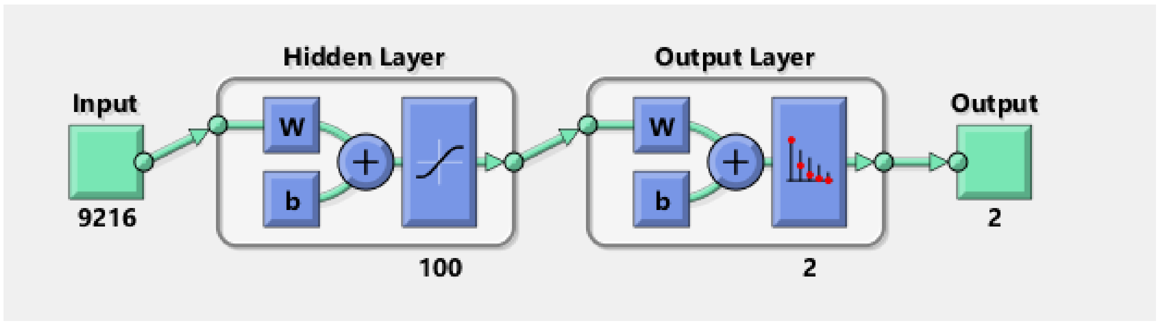


Figure 29. Tennis Ball Object Follow Neural Network Structure.

When training each of the three neural networks within Matlab the training data will segmented into the three training classes as shown in Table 6.

Table 6. Training Data Segmentation.

	Number of Samples
Training	70%
Validation	15%
Testing	15%

4.1.1. Neural Network Left Verses Right

For this network, the output should determine whether the tennis ball is located on the left or right side of the image frame. This output may then be utilized to develop commanded values to the multirotor UAV, with the intension of keeping the object in the center of the image frame, therefore centered with the UAV, as shown in Figure 30.

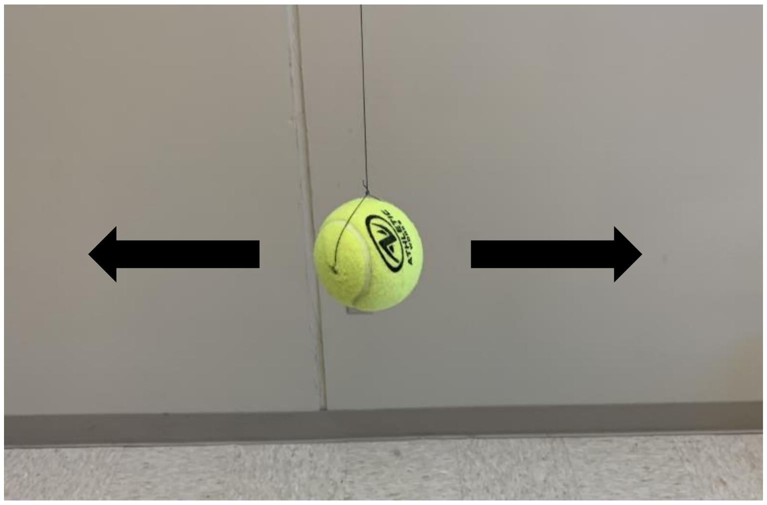


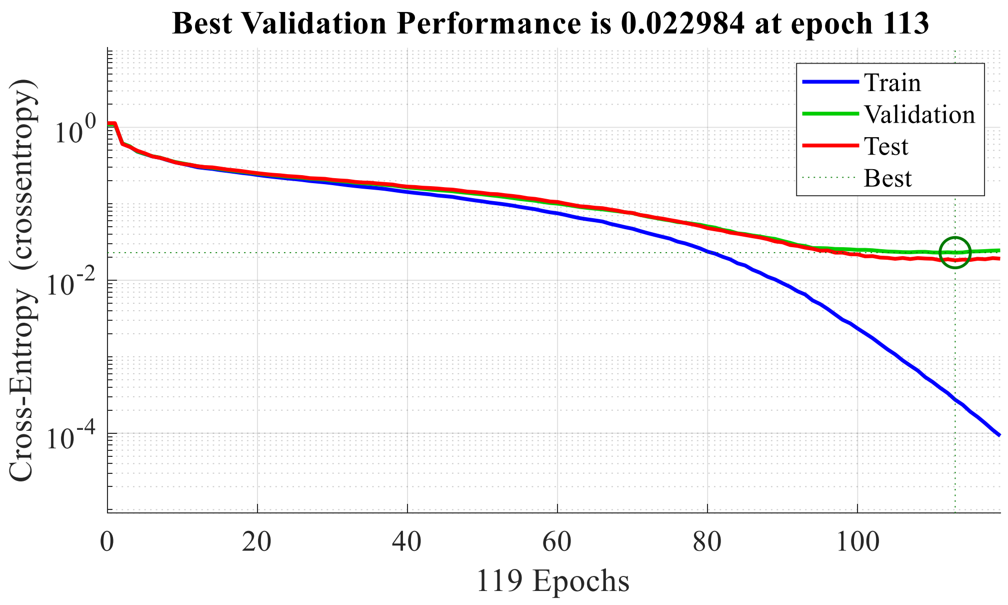
Figure 30. Left Versus Right, Tennis Ball.

To train this network 24,106 images were collected with 50% of the images containing the tennis ball of the left side of the image and 50% on the right side. To increase the total number of training samples an additional experiment will be explored. This entails mirroring all the images about the vertical plane, thereby doubling the total number of images. For the mirrored images the target value associated with them will change from left to right and right to left. The training performance using this method will be compared to that of the other two networks to ensure the viability of this practice. The total number of samples is tabulated in Table 7. Where one-fourth of the data will be held out for post training performance testing.

Table 7. Left Versus Right Training Data.

	Number of Samples
Total Images	48212
Hold Out	12053
Training Remaining	36159

The performance plot for the training session is shown in Figure 31. While the confusion matrix is shown in Figure 32.



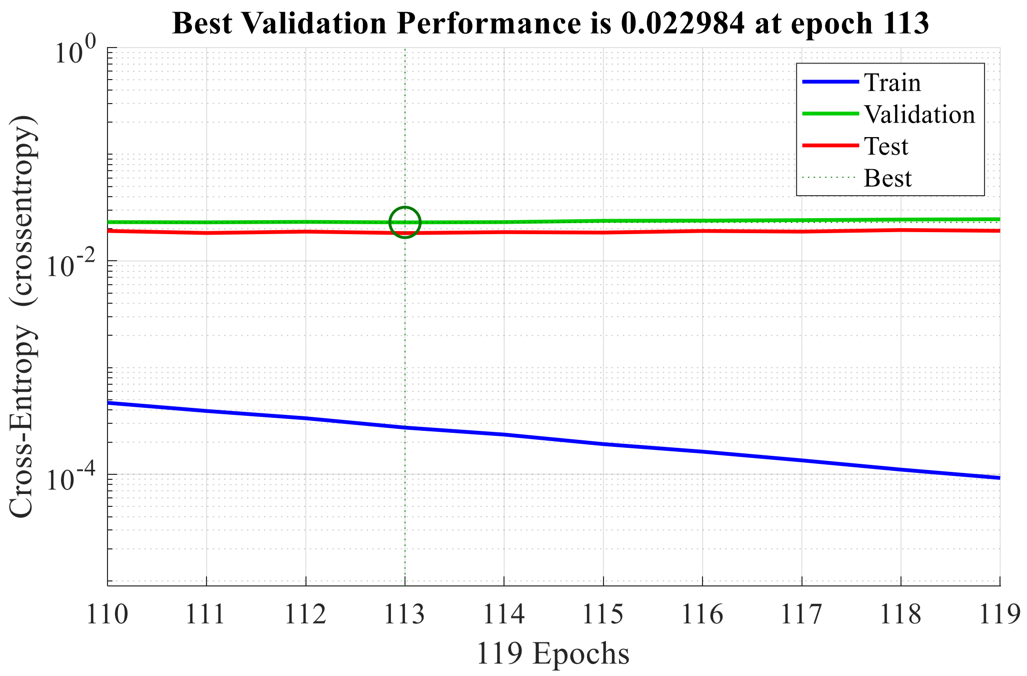


Figure 31. Left Versus Right Performance (top) detail zoom (bottom).

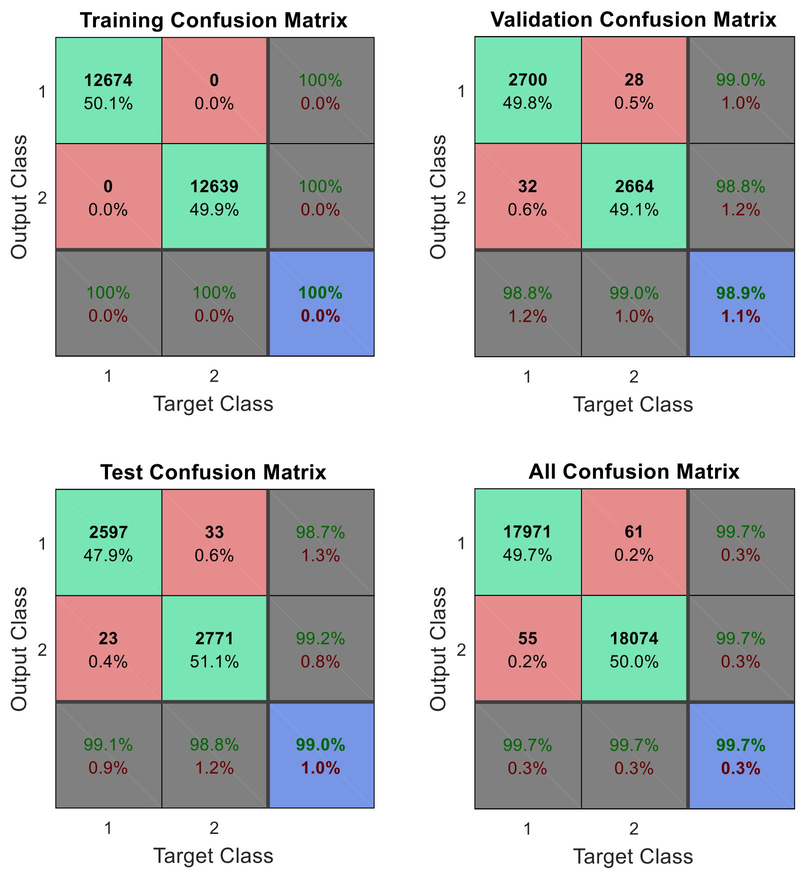


Figure 32. Left Versus Right Confusion Matrix.

The performance plot validation and testing categories reached a convergent solution after 113 epochs. While the confusion matrix only failed to classify 116 images or 0.3% of the training data. As for the post training performance the neural network accurately classified 98.73% of the never before seen held-out training data.

4.1.2. Neural Network Up Versus Down

For this network, the output should determine whether the tennis ball is located on the upper half or lower half of the image frame. This output may then be utilized to develop commanded values to the multirotor UAV. With the intension of keeping the object in the center of the image frame, as shown in Figure 33.

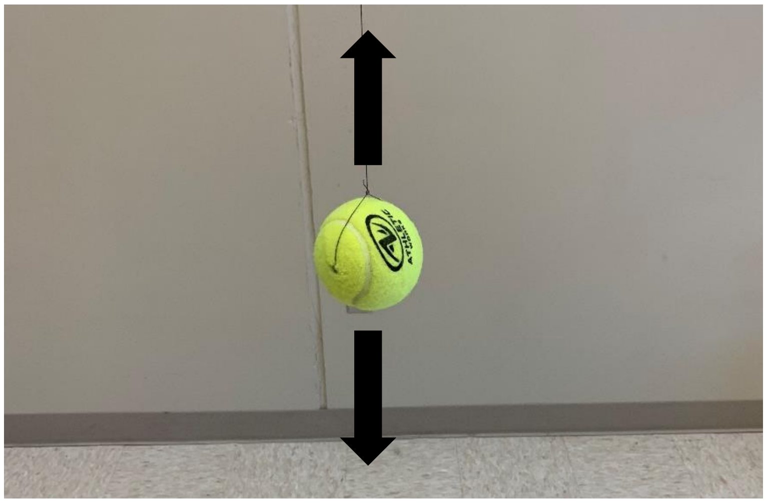


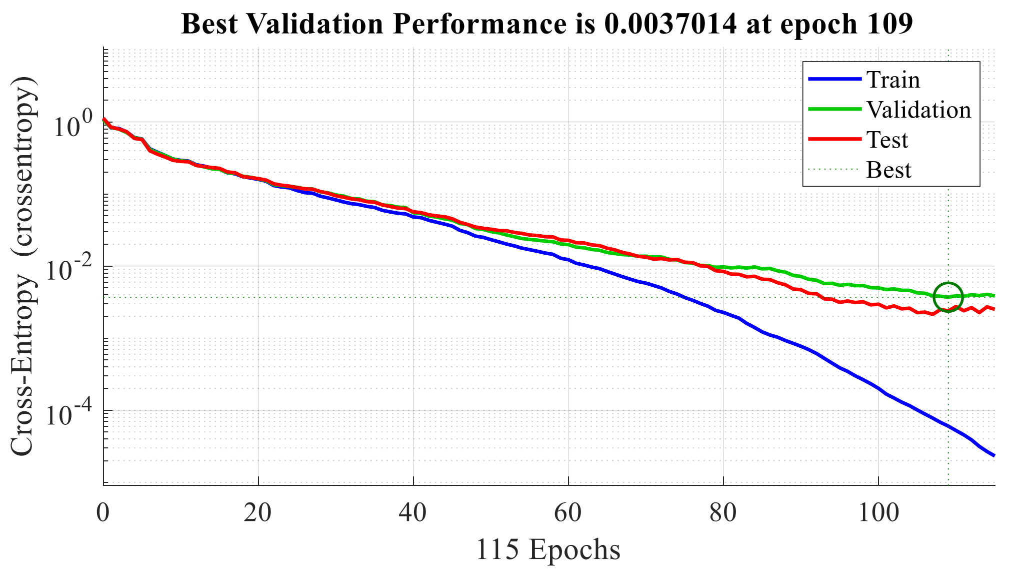
Figure 33. Up Versus Down, Tennis Ball.

To train this network 14,880 images were collected with 50% of the images containing the tennis ball of the upper half of the image and 50% on the lower half. The total number of samples is tabulated in Table 8. One-tenth of the data will be held out for post training performance testing.

Table 8. Up Versus Down Training Data.

	Number of Samples
Total Images	14880
Hold Out	1488
Training Remaining	13392

The performance plot for the training session is shown in Figure 34. While the confusion matrix is shown in Figure 35.



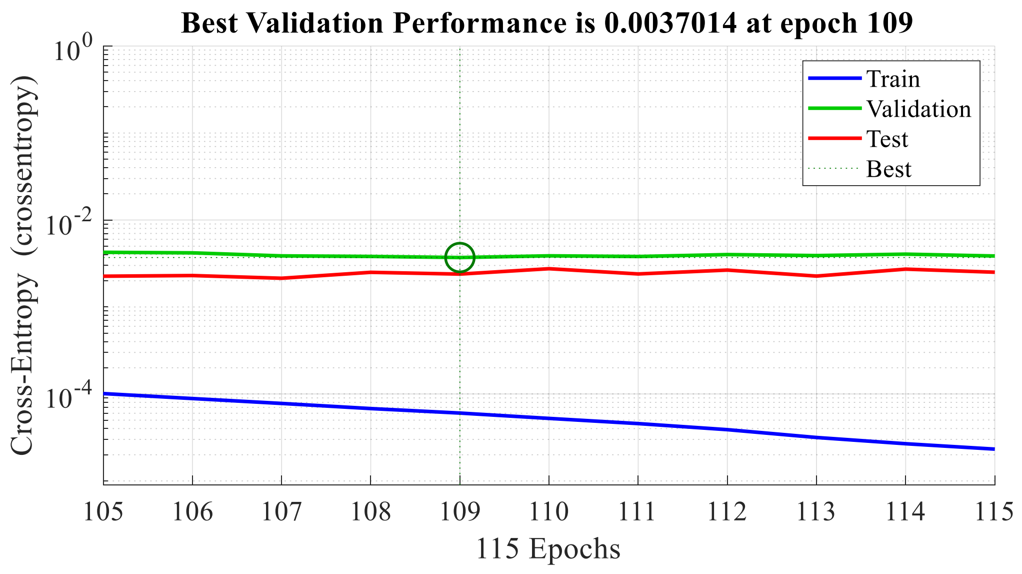


Figure 34. Up Versus Down Performance (top) detail zoom (bottom).

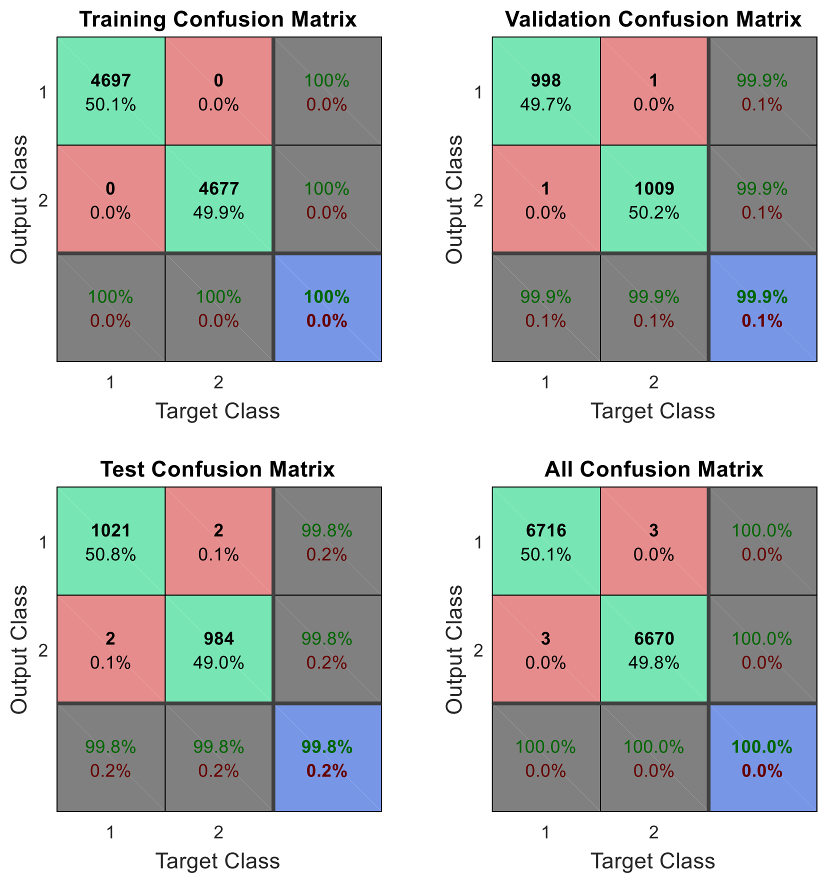


Figure 35. Up Versus Down Confusion Matrix.

The performance plot validation and testing categories reached a convergent solution after 109 epochs. While the confusion matrix only failed to classify 6 images from the training data. As for the post training performance the neural network accurately classified 99.86% of the never before seen held-out training data.

4.1.3. Neural Network Forward Versus Back

For this network, the output should determine whether the tennis ball is located closer or further away from the multirotor UAV. This output may then be utilized to develop commanded values to the multirotor UAV. With the intension of keeping the object within a set distance, as shown in Figure 36.

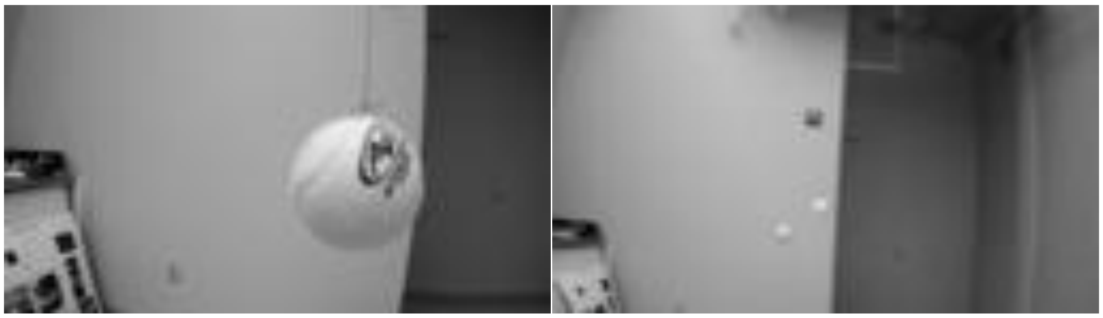


Figure 36. Tennis Ball Close (left) and Far (right).

To train this network 14,110 images were collected with 50% of the images containing the tennis ball near the drone and 50% further away. The total number of samples is tabulated in Table 9. One-tenth of the data will be held out for post training performance testing.

Table 9. Forward Versus Back Training Data.

	Number of Samples
Total Images	14110
Hold Out	1411
Training Remaining	12699

The performance plot for the training session is shown in Figure 37. While the confusion matrix is shown in Figure 38.

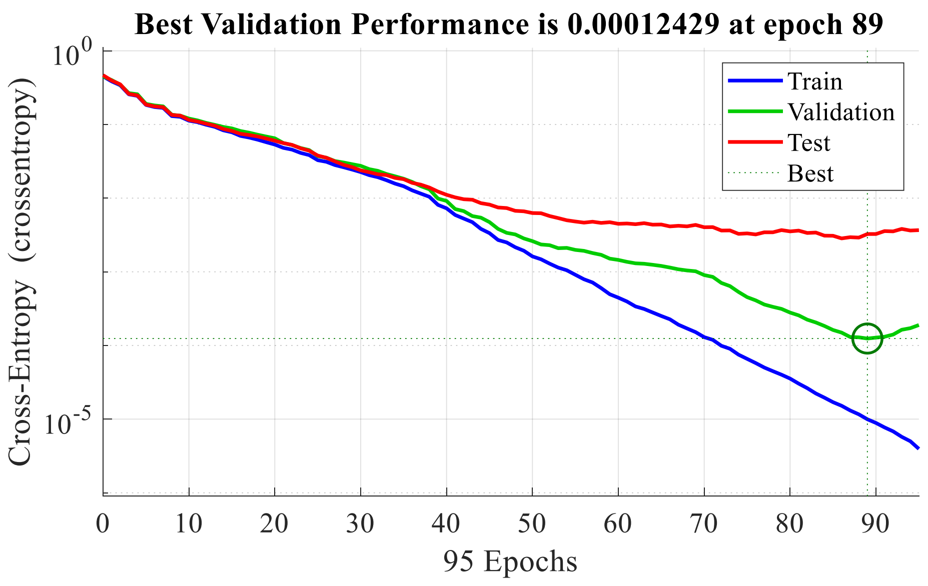


Figure 37. Forward Versus Back Performance.

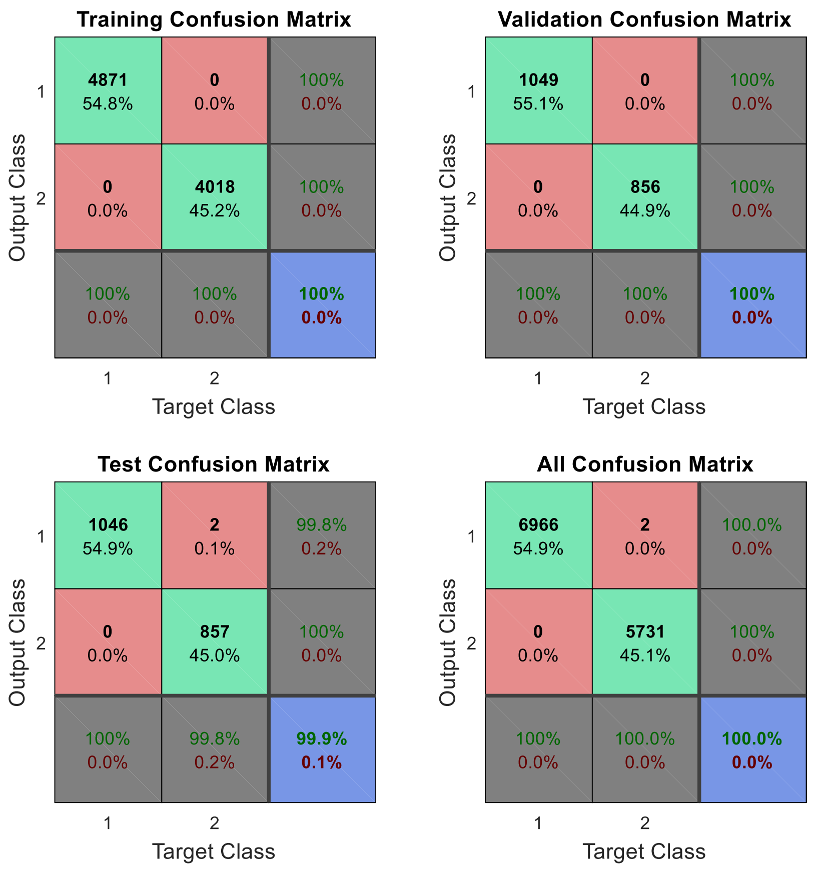


Figure 38. Forward Versus Back Confusion Matrix.

The performance plot validation and testing categories reached a convergent solution after 89 epochs. Although the testing and validation data sets reached convergences at two different epochs. This behavior may indicate that more training data may be needed to improve training. While the confusion matrix only failed to classify 2 images from the training data. As for the post training performance the neural network accurately classified 100.00% of the never before seen held-out training data. Two of these metrics indicate that the network is properly trained but further testing with real time dynamic data may be needed to confirm performance.

4.2. Pre-recorded Simulation Testing

Using the neural network developed to determine if the tennis ball is on the left or right half of the image frame, a simple experiment can be conducted to check the outputted values of the network with pre-recorded data. In this case within Simulink a 15 second video of the tennis ball may be captured and passed through the neural network to visualize the outputted values. For this test the camera is moved such that the tennis ball moves between the left and right sides of the frame. Figure 39 shows the plotted output of the neural network, where if the output is 0 the tennis ball is on the left side of the frame and if the value is 1 the tennis ball is on the right side of the frame.

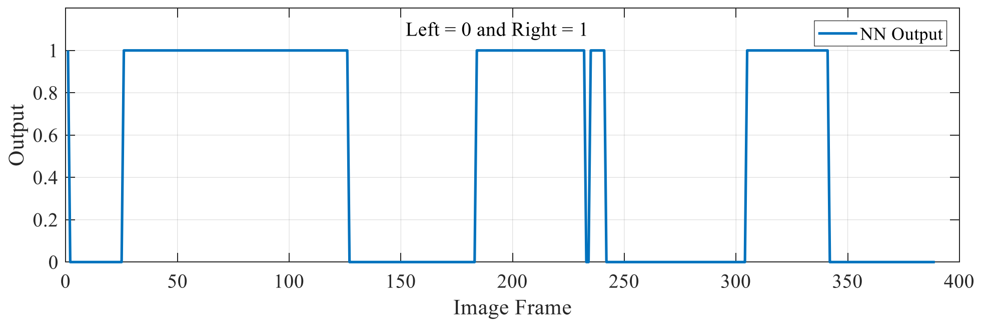


Figure 39. Pre-recorded Simulation Results.

Referencing this outputted response a few observations can relating to the behavior of the neural networks. It is seen that for relatively smooth motion from left to right or right to left that a sharp transition occurs. This behavior is contradictory to an expected outcome for values near the transition, or center, of the image frame for a binary output model. It was expected that a hysteresis effect would occur at this transition zone when the network could oscillate between left and right as each case is difficult to predict in reference to the other. A few conditions could have a direct effect to this behavior including; the translational speed of the multirotor over the transition center line of the image frame, the quality of provided training data, or the classification output behavior of the neural network. Regardless of the causes of this behavior the simulated response provides a useful insight to the real-time response of the neural network. Therefore, using this outputted value of 0 or 1 the neural networks may now be implemented in a multirotor dynamics model to test their ability to provide reliable commanded values to the testbed.

4.3. Simulated Multirotor Dynamics

Within Simulink a dynamic model for the guidance and control of a multirotor UAV may be created and utilized for redemptory simulation experimentation to validate and verify that the neural networks can provide relevant command data to a flight controller in real time. The basis for this model utilizes governing equations of motion for a multirotor derived by Tain-Sou Tsay [18]. This model allows utilizes four input commanded values to be implemented by the user to simulate real-time flight. These commanded values are input into a tuned PI controller as shown in Figure 40.

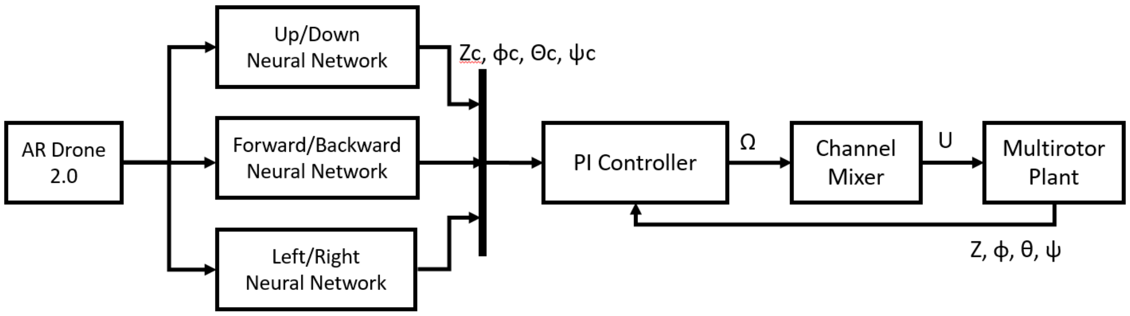


Figure 40. Dynamic Multirotor Model.

4.3.1. Commanded Value Description and Controller Methods

The four commanded values z_c, ϕ_c, θ_c, and ψ_c correspond to the height, roll, pitch, and yaw of the multirotor accordingly. Implementing the three developed neural networks within the model, three of the commanded values z_c, ϕ_c, and θ_c may be adjusted given the network outputs while the last commanded value ψ_c is left constant at 0. If the up versus down neural network determines the tennis ball is in the upper half of the image or if the ball is in the lower half of the image then the commanded value for z_c, or height, may be incremented ±0.005 m. If the in the left versus right neural network the ball is within the left or right half of the image the ϕ_c, or roll angle, commanded value may be incremented ±0.01 radians. Finally, if the ball is near or further away than the trained distance limit, the θ_c, or pitch angle, commanded value may be incremented ±0.01 radians. Each of these incremental changes may be adjusted to increase or dial back the responsiveness of the multirotor.

The commanded values may then be compared to the current state of the multirotor, either calculated through known multirotor dynamics, as in this example, or observed through the indoor testbed. An associated state error may then be generated and passed through a tuned PI, proportional integral, controller. A block representation of this controller is shown in Figure 41. For each of the commanded values there will be a PI controller associated with it.

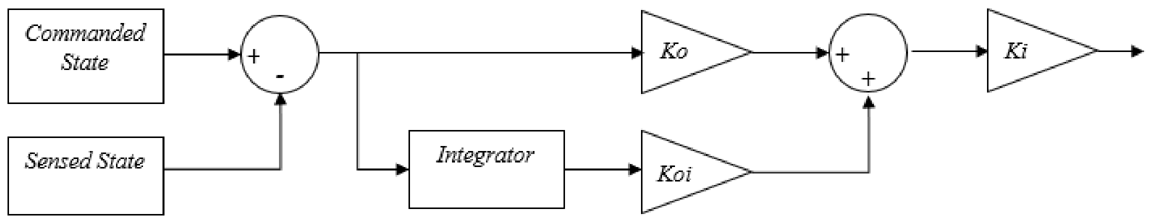


Figure 41. Block Diagram of PI Controller

For the simulated experiment utilizing this multirotor dynamic model the proportional and integral gains as tabulated in Table 10.

Table 10. PI Controller Gains

z_c		θ_c	
Kiz	10	Kiq	3
Koz	0.5	Koq	1
Kozi	0	Koqi	0
ϕ_c		ψ_c	
Kip	3	Kir	0.5
Kop	1	Kor	0.5
Kopi	0	Kori	0

For further detail Figure 42, shows a more accurate but complex representation of the PI controller and channel mixer. Following the standard PI controller additional state values are utilized to create the PI controller governing equations shown as follows in Eq. (18).

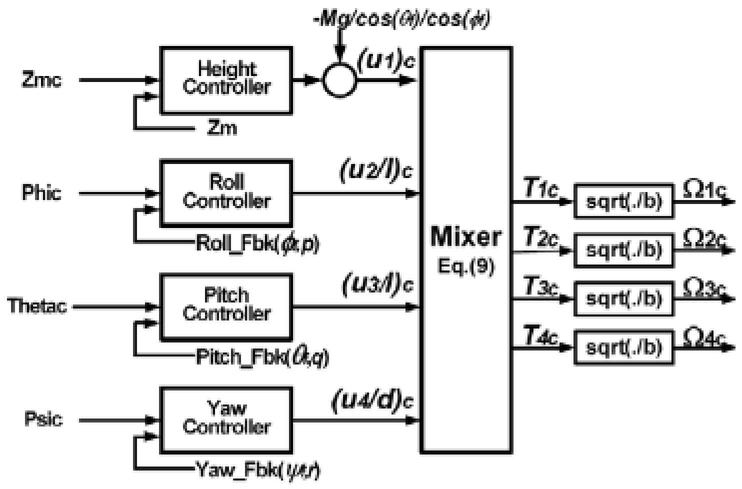


Figure 42. PI Controller Schematic.

$$\begin{gathered}
u_{1c} = K_{iz}\left[\left(K_{oz} - \frac{K_{ozi}}{s}\right)(Z_c - Z_f) - w_f\right] - \frac{mg}{\cos\theta_f \cos\phi_f} \\
u_{2c} = K_{ip}\left[\left(K_{op} - \frac{K_{opi}}{s}\right)(\phi_c - \phi_f) - p_f\right] \\
u_{3c} = K_{iq}\left[\left(K_{oq} - \frac{K_{oqi}}{s}\right)(\theta_c - \theta_f) - q_f\right] \\
u_{4c} = K_{ir}\left[\left(K_{or} - \frac{K_{ori}}{s}\right)(\psi_c - \psi_f) - r_f\right]
\end{gathered} \tag{18}$$

Where variables denoted by an *f* indicated sensed or observed state values and *p*, *q*, and *r* represent angular rates in the body axis. With a tuned output from the PI controller the four commanded thrusts, u_c, terms must then be mixed and converted to corresponding thrust for each rotor using the following equation:

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} -1/4 & 0 & +1/2 & -1/4 \\ -1/4 & -1/2 & 0 & +1/4 \\ -1/4 & 0 & -1/2 & -1/4 \\ -1/4 & -1/2 & 0 & +1/4 \end{bmatrix} \begin{bmatrix} -(T_1 + T_2 + T_3 + T_4) \\ T_4 - T_2 \\ T_1 - T_3 \\ -T_1 + T_2 - T_3 + T_4 \end{bmatrix} \tag{19}$$

Using this generated thrust, an associated rotor angular rate is calculated using the following relationship:

$$\Omega_i = \frac{\sqrt{T_i}}{b} \tag{20}$$

Where the index *i* represents the rotor number, *b* is the damping coefficient of the air, and *Ω* is the rotor angular rate. These associated rotor angular rates may then be passed through the multirotor plant to generate new state conditions.

4.3.2. Multirotor Plant Dynamics

The multirotor plant is more descriptively a set of kinematic relationships capable of determining state variables given the commanded rotor angular rates. The following equations works back from the angular rate to the thrust generated:

$$\begin{gathered}
u_1 = -b(\Omega_1 + \Omega_2 + \Omega_3 + \Omega_4) \\
u_2 = -lb(\Omega_4 - \Omega_2) \\
u_3 = -lb(\Omega_1 - \Omega_3) \\
u_4 = -db(-\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4)
\end{gathered} \tag{21}$$

Where d is the ratio of thrust to the angular moment and *l* is the position of the rotor from the center of gravity. Each of these u_i components are then utilized to derive; the body-axis velocities *u*, *v*, and *w*; the angular rates *p*, *q*, and *r*; and the attitude angles ϕ, θ, and ψ as given in the following equation:

$$\begin{bmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{bmatrix} = \frac{1}{m}\begin{bmatrix}F_x\\ F_y\\ F_z\end{bmatrix} + g\begin{bmatrix}-\sin\theta\\ \cos\theta\sin\phi\\ \cos\theta\sin\phi\end{bmatrix} - \begin{bmatrix}qw - rv\\ ru - pw\\ pv - qu\end{bmatrix} \qquad (22)$$

Where g is the gravitational constant, m is the total mass of the multirotor, and F_x, F_y, and F_z are the three axis forces given by the following:

$$\begin{bmatrix}F_x\\ F_y\\ F_z\end{bmatrix} = \begin{bmatrix}0\\ 0\\ u_1\end{bmatrix} + \begin{bmatrix}C_{fx}\\ C_{fy}\\ C_{fz}\end{bmatrix} \qquad (23)$$

In this case C_{fx}, C_{fy}, and C_{fz} are aerodynamic forces in the three-axes. Using these relationships, the new state variables of the multirotor are then found and may be used for the next sampling step of the multirotor model using the following two equations.

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & \tan\theta\sin\phi & \tan\theta\cos\phi\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sec\theta\sin\phi & \sec\theta\cos\phi\end{bmatrix} + \begin{bmatrix}p\\ q\\ r\end{bmatrix} \qquad (24)$$

$$\begin{bmatrix}\dot{X}\\ \dot{Y}\\ \dot{Z}\end{bmatrix} = \begin{bmatrix}\cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi\\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi\\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta\end{bmatrix}\begin{bmatrix}u\\ v\\ w\end{bmatrix} \qquad (25)$$

4.3.3. Simulated Multirotor Object Tracking

Using the developed model three simulated maneuvers may now be performed to test the proposed concept of object tracking using three neural networks for three of the command inputs. The first neural network utilizes the up versus down model to provide an updated z_c value. By supplying a prerecorded video where the camera is moved up or down continuously in a smooth motion while holding other degrees of freedom constant. Figure 43 shows a time history of the commanded height in response to the video stimuli provided. The model and neural network react and provide continuous feedback in both the positive and negative *z* direction.

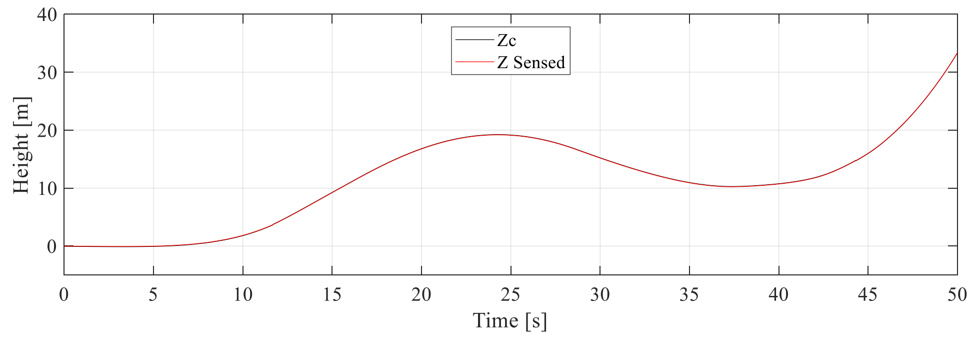


Figure 43. Simulated Z response.

Then, supplying a prerecorded video where the camera is moved left or right continuously in a smooth motion while holding other degrees of freedom constant, we may utilize the left versus right neural network to provide a commanded roll value, ϕ_c. The time history for this maneuver is presented in Figure 44. As is previously observed the neural network does provide contiguous and relevant command data to the multirotor, but some interesting behavior can be observed near the transition point between the two cases. From these points an occurrence know as hysteresis is affecting the commanded output behavior. This behavior is likely to occur when an image from the camera could be equally classified as wither of the two cases. Regardless of this behavior the sensed state of the multirotor remains generally unaffected and continues to track the object.

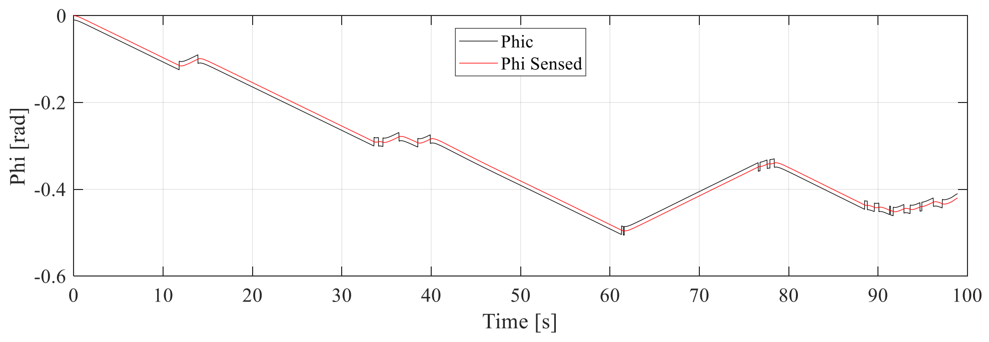


Figure 44. Simulated ϕ response.

Finally, supplying a prerecorded video where the camera is moved forward or back continuously in a smooth motion while holding other degrees of freedom constant, we may utilize the forward versus back neural network to provide a commanded pitch value, θ_c. As with the previous two cases a time history is shown in Figure 45. A similar behavior may be observed near the transition point of the motion, where some amount of hysteresis causes some uncertainty of correct classification. Although, the sensed state of the multirotor tends to remain near the average of these points.

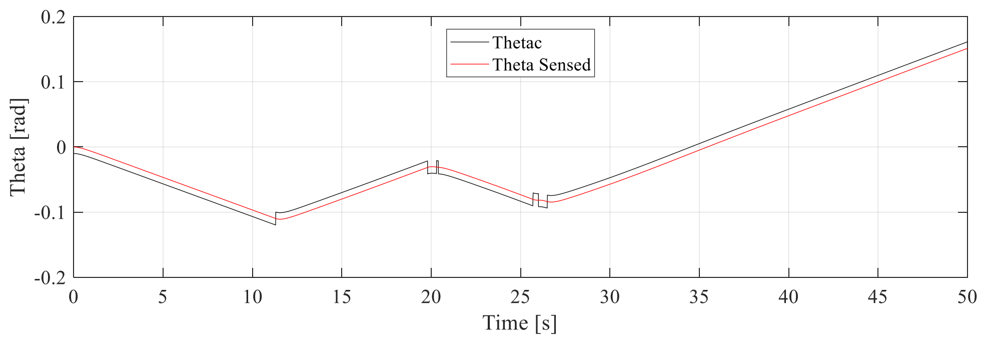


Figure 45. Simulated θ response.

From these simulations it is determined that the three neural networks may be implemented with a multirotor controller to provide relevant command values for object tracking flight. The next step to be explored within this thesis is to now integrate the neural networks within the flight controller for the Indoor Multirotor Testbed.

4.4. Experimental Indoor Multirotor Object Tracking

Within the Indoor Multirotor Testbed, the first change between the model and the previously presented simulated multirotor dynamics is inputted command values. The simulated multirotor controller provided a height, roll, pitch, and yaw while the testbed utilizes a commanded *x*, *y*, and *z* location with respect to the system origin. Like how the neural networks provided an incremental control to the commanded values depending on which classified state the object was within; the same process may then be applied to the new commanded values. The first step to correct this differential input type is to simplify the experimental conditions the multirotor is presented.

To accomplish this several assumptions should be made about the tacking maneuver. Within the test bed the multirotor should be placed such that the camera faces the positive x-axis direction with the object of interest being located about two meters in front of the UAV. The second assumption is that the yaw, ψ_c, remains constant at 0 degrees keeping the UAV orientation fixed within the X-Y plane. The third assumption for experimental tests of the neural networks is that the Indoor Multirotor Testbed with use of the Motive software will be used to bring the multirotor into a hovering state near the object, where the neural networks will then take control of the commanded values for the UAV.

A block diagram of the model is shown in Figure 46.

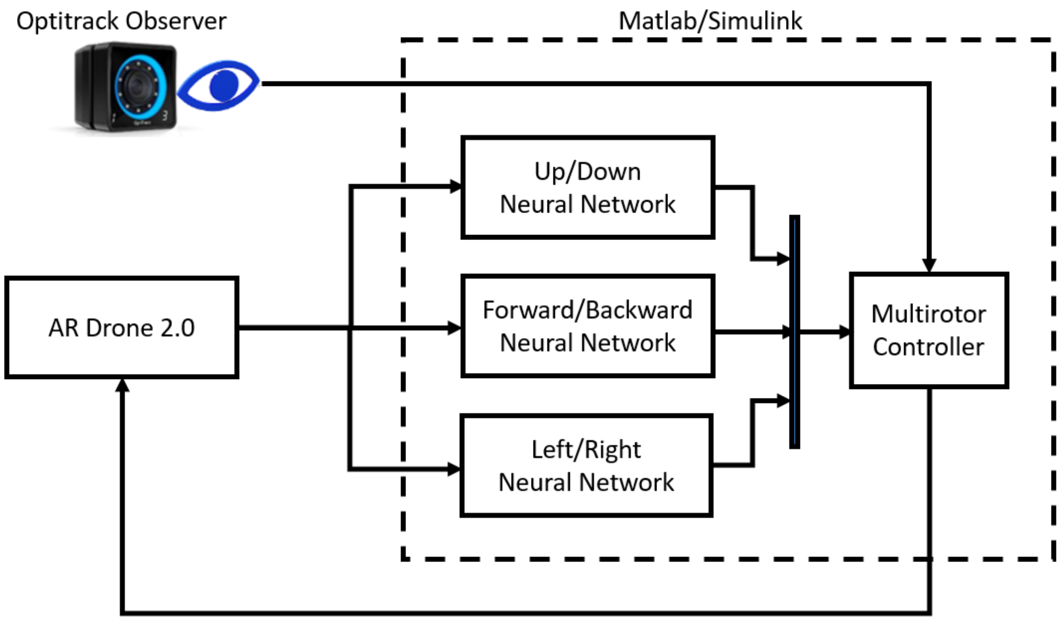


Figure 46. Indoor Multirotor Controller.

The up versus down network will be utilized to provide a commanded height value, z_c. If the object, within the image frame, is located on the upper half of the image the commanded value may be incremented up by a set value to center the object. If the object is in the lower half of the image frame the commanded value may be incremented down to center the object.

The forward versus back neural network will provide a commanded x-axis value, x_c. If the object, within the image frame, is located near the multirotor the commanded value will be incremented down so that the UAV moves away from the object. If the object is farther away from the multirotor the commanded value will be incremented up so that the UAV moves closer to the object.

For the final commanded value, the y-axis variable, y_c, will be incremented up if the object is in the right half of the image frame. If the object is in the left half of the image frame the UAV the commanded value will be incremented down. To determine this the left versus right neural network will be used.

4.4.1. Issues Facing Experimental Testing of a Neural Network Integrated Controller

Although the description of the how a neural network is implemented within the Indoor Multirotor Testbed was presented in the previous section, several design faults within the model have occurred. The presence of these issues prevents the full experimentation and testing of the neural networks with real-time flight maneuvers.

The first of these issues is the latency, or delay, of the TCP communication to present an image of the current state of the multirotor to the neural networks in an efficient sample time. This latency would restrict the ability of the multirotor to react to real-time object occurrences. The second issue to overcome is a result of the configuration protocols of a Simulink model. As a Simulink model is initiated, the entire model experiences a configuration period for the model to run smoothly. This means that the first sampling step of the model take considerably more time to run than the following simulation. If a Simulink file has several parallel processes being initiated, a large delay may occur slowing the initial processes of the model. The final issue directly relates to the two previous issues and in practice is most likely a symptom of these issues. As the Indoor Multirotor Testbed utilizes several separate models in communication with one another, the controller model and the flight streaming model, each of these models must remain in sync and run at the same sampling rate. If one of these models becomes delayed for even a small time period, the communications may time out and cancel the model.

When performing an in-flight test of the neural networks and the multirotor controller model fails to initial before the system timeout. The multirotor may initiate flight without proper commanded values and may suffer from a fly away event. A fly away is an event where the UAV may take off and be unresponsive to control commands and crash into its surroundings causing damage or serious injuries to those around the UAVs environment. Until these issues are resolved or address further experimentation will be let on hold.

CHAPTER FIVE

CONCLUSION

5.1. Summary

This thesis has presented the methods for development of a neural network-based object detector for multirotor object tracking. The methods presented to train shallow neural networks was validated using a simplified object detection experiment. Then, using these methods three neural networks were trained for three separate cases; to judge if a tennis ball is in the upper or lower half of an image frame, to determine if the tennis ball is in the left or right side of the image frame, and whether the tennis ball in near or further away from the camera. Each of these neural networks was independently tested with a large sample of held-out, never before seen, training data. Using this data, each of the networks demonstrated an accuracy of over 98.7% when classifying the images into the two possible cases.

Following these performance tests, the neural networks were than implemented into a simulated multirotor UAV model that simulates the dynamic response of a multirotor. With this model each of the three neural networks was independently implemented within the model to test the real-time object tracking performance with prerecorded video. With all three neural networks the dynamic model showed that the neural networks could provide relevant command data to a simulated multirotor.

Finally, this thesis outlined how to implement the three neural networks within the Indoor Multirotor Testbed in the ARTLAB at South Dakota State University. Although, latency issues within the communication protocols of the model prevented real-time flight response for object tracking.

5.2. Recommendation

For future work several recommendations can be made to expand upon and further many of the methods and concepts presented. The first of these suggested recommendations is that further experimentation be conducted to determine the optimum number of neurons within the hidden layer of the neural network. Due to the computational time to train and test neural networks for object detection it become difficult to test a wide range of networks using dozens of different neuron counts, but with enough study an optimum neuron count may show that more or less neuron may be needed to effectively track an object.

The second recommendation is that a wider assortment of neural networks be explored and objectively compared. As this thesis utilized shallow neural networks, neural networks with a single hidden layer, a deeper neural network, or even a convoluted neural network, may reach a compromise in accuracy and computational speed performance.

Next, more study and experimentation may be made in the determination of resolution quality and its effects object tracking performance. If a scenario does not require a high level of resolution fidelity at what point does decreasing the pixel count have an adverse effect on the neural network. This same methodology may also be applied to removing the color from the input image. More study could show that leaving the three-pixel color components in tact may result in better neural network performance, albeit resulting in higher computational times.

The final recommendation be that more effort and research be done to decrease the computational load of the multirotor model when processing live image data. This same study should be assessed with the neural networks themselves to increase the overall response rate of the system. This may provide for new opportunities to further the used of neural networks as a high-fidelity object-based detection and tracking method. If presented in a faster acting method that utilizes less computational power the neural network methods may be implemented in small more mobile computer systems, like mobile smart phones, for highly accessible controllers for autonomous flight in applications where computational resources are scarce.

References

[1] F. Morbidi, R. Cano and D. Lara, "Minimum-Energy Path Generation for a," in *IEE International Conference on Robotics and Automation*, Stockholm, 2016.

[2] R. Mahony, V. Kumar and P. Corke, "Multirotor Aerial Vehicles: Modeling,," *IEEE Robotics & Automation Magazine,* vol. 19, no. 3, pp. 20-32, 2012.

[3] S. Hrabar and G. Sukhatme, "Vision-Based Navigation through Urban Canyons," *Journal of Field Robotics,* vol. 26, pp. 431-452, 2009.

[4] R. P. Padhy, S. Verma, S. Ahmad, S. K. Choudhury and P. K. Sa, "Deep Neural Network for Autonomous UAV Navigation in Indoor," in *International Conference on Robotics and Smart Manufacturing (RoSMa2018),* Chenni, India, 2018.

[5] A. Kouris and C.-S. Bouganis, "Learning to Fly by MySelf:A Self-Supervised CNN-based Approach for Autonomous Navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems,* Madrid, Spain, 2018.

[6] D. Gandhi, L. Pinto and A. Gupta, "Learning to Fly by Crashing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems,* Vancouver, BC, Canada, 2017.

[7] M. Unander, "Autonomous flying of quadrotor for 3D modeling and inspection of mineshaft," Digitala Vetenskapliga Arkivet, 2018.

[8] N. Smolyanskiy, A. Kamenev, J. Smith and S. Birchfield, "Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vancouver, BC, Canada, 2017.

[9] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review,* vol. 65, no. 6, pp. 386-408, 1958.

[10] M. Nielson, "Neural Networks and Deep Learning," Determination Press, 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com/. [Accessed 2 July 2018].

[11] A. Cauchy, "M´ethode g´en´erale pour la r´esolution des syst`emes d'´equations," in *C. R. Acad. Sci*, Paris, 1847, pp. 536-538.

[12] S. Ruder, "An overview of gradient descent optimization algorithms," 2016.

[13] R. Rojas, Neural Networks - A Systematic Introduction, New York: Springer-Vertag, 1996.

[14] M. Sundermeyer, I. Oparin, J.-L. Gauvain, B. Freiberg, R. Schluter and H. Ney, "COMPARISON OF FEEDFORWARD AND RECURRENTNEURAL NETWORK LANGUAGE MODELS," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, BC, Canada, 2013.

[15] L. Ba and R. Caruana, "Do Deep Nets Really Need to be Deep?," in *Advances in Neural Information Processing Systems*, Montreal Canada, 2014.

[16] J. d. Villiers and E. Barnard, "Backpropagation Neural Nets with One and Two Hidden Layers," *IEEE TRANSACTIONS ON NEURAL NETWORKS,* vol. 4, no. 1, pp. 136-141, 1992.

[17] K. Guye, "Development of an Indoor Multirotor Testbed for Experimentation on Autonomous Guidance Strategies," Open, 2018. [Online]. Available: http://openprairie.sdstate.edu/etd/2972. [Accessed 2019].

[18] T. S. Tsay, "Guidance and Control Laws for Quadrotor UAV," 2014.

APPENDIX I

A. Video to Image Conversion

```
%Video conversion Script
%Input a single Video to convert to '.jpg' file type for each frame
%For more info:
https://www.mathworks.com/help/matlab/ref/videoreader.html
clear all
close all
clc

vname_str = '.mp4';   %File Name; input video file name

v = VideoReader(vname_str); %Video read
% D = v.Duration;   %Video duration time, [s]
% F = v.FrameRate;  %Avg video frame rate [frames/s]
% n = v.NumberOfFrames; %Number of Frames

%File name and directory name
newSubFolder = erase(vname_str,".mov");

%Create the folder if it doesn't exist already. Stores images in folder
if ~exist(newSubFolder, 'dir')
  mkdir(newSubFolder);
end

ii = 1; %Increment Loop
while hasFrame(v)
   A = imresize(readFrame(v),0.1); %Resize image
   img = rgb2gray(A); %Convert to greyscale
   filename = [sprintf('img_%03d',ii) '.jpg'];  %Name image
   fullname = fullfile(newSubFolder,filename);  %Save location
   imwrite(img,fullname)    %Save frame
   ii = ii+1;   %increment
end
```

B. JPG to CSV File Conversion

```
%Converts all .jpg files to a single .csv file with the first column
%indicating expected output for Neural Network training
clear all
close all
clc

%Set Pixel Count
[m,n] = size(imread('.jpg')); %Insert the first .jpg file

myFolder = ''; %Set folder containing .jpg images
filePattern = fullfile(myFolder, '*.jpg'); %Determines file pattern
theFiles = dir(filePattern); %Creates dir of file pattern

for k = 1 : length(theFiles) %Loops for number of images in folder
A = imread([sprintf('img_%3d',k) '.jpg']); %Read image
B = reshape(A',[1,m*n]); %Reshape matrix into single row array
C(k,:) = [1,B];   %Output 1, Output 2
end

csvwrite('.csv',C); %Insert desired file name
```

C. CSV Combiner

```
%Combines .csv files into a random arrangement keeping columns
consistent for a particular row
clear all
close all
clc

%Open files, add as needed
A = csvread('.csv'); %File 1
B = csvread('.csv'); %File 2
C = csvread('.csv'); %File 3
D = csvread('.csv'); %File 4

E  = cat(1,A,B,C,D); %Combines files

random_E = E(randperm(size(E, 1)), :); %Rearranges file

csvwrite('train.csv',random_E) %Write new .csv file
```

D. Training Preparation

```
%Prepares and segregates training data for neural network training
clear all
close all
clc

tr = csvread('.csv', 1, 0);          % read train.csv

n = size(tr, 1);                     % number of samples in the dataset
targets  = tr(:,1);                  % 1st column is |label|
targets(targets == 0) = 10;          % use '10' to present '0'
targetsd = dummyvar(targets);        % convert label into a dummy
variable
inputs = tr(:,2:end);                % the rest of columns are
predictors

inputs = inputs';                    % transpose input
targets = targets';                  % transpose target
targetsd = targetsd';                % transpose dummy variable

rng(1);                              % for reproducibility
h = 3;                               % hold out value
c = cvpartition(n,'Holdout',n/h);    % hold out 1/h of the dataset

Xtrain = inputs(:, training(c));     % h-1/h of the input for training
Ytrain = targetsd(:, training(c));   % h-1/h of the target for training
Xtest = inputs(:, test(c));          % 1/h of the input for testing
Ytest = targets(test(c));            % 1/h of the target for testing
Ytestd = targetsd(:, test(c));       % 1/h of the dummy variable for
                                       testing
```

E. Python Video Socket Stream

```
%Access IP Camera in Python OpenCV

import cv2
import socket
import numpy


% Define Socket Address for Python Simulink Communication
host = '127.0.0.1'                            % IP Address, Local Address
port = 8000                                        % Port, use Empty Port

sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM) % Create Socket
for Stream
sock.bind((host,port))                                 % Bind IP and Port

sock.listen(5)             % Listen for Accept Confirmation from Matlab
(clientsocket, address) = sock.accept()                     % Accept

stream = cv2.VideoCapture('tcp://192.168.1.1:5555')      % Connect to AR
Drone 2.0 Video Stream

while True:                                        % Loop While Connected

    r, frame = stream.read()                           % Read Image Frame

    %Process Image Data for use in Matlab and Simulink
      % Collect Iinformation of the Incoming Image Type
    info = numpy.iinfo(frame.dtype)
      % Normalize the data between 0 - 1
    frame = frame.astype(numpy.float64) / info.max
    frame = 255 * frame                                 % Now scale by 255
    frame = frame.astype(numpy.uint8)      % Convert to unit8 File Type

      % Convert to Grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    %cv2.imshow('IP Camera stream',gray)                  % Show Image

    %cv2.imwrite('Image.jpg',frame)                       % Save Image

    clientsocket.send(gray)                             % Send to Socket

    if cv2.waitKey(1) & 0xFF == ord('q'):                % Exit Program
        break

cv2.destroyAllWindows()                                   % Close Windows
```

F. Simulated Multirotor Dynamics

```
%Sets Constant Parameters then executes Sim File (DC2)
clear all
close all
clc

%% %Simulation Parameters
sim_time = 200; %Sets Simulation run duration
dt = 0.01; %Sets Simulation step time

%% %Parrot Specifications
b = 1.2953*10^-5; %Dampening Coefficient of rotor
l = 0.315; %position of propeller from the central gravity
d = 0.008; %ratio of thrust to angular moment
m = 0.429; %Parrot mass
g = 9.81; %Gravity Constant
Cfx = 0; %Aerodynamic Forces along x-axis, zero for low speed
operations
Cfy = 0; %Aerodynamic Forces along y-axis, zero for low speed
operations
Cfz = 0; %Aerodynamic Forces along z-axis, zero for low speed
operations

%Moment of Inertia Values
Ixx = 1; %Moment of Inertia along x-axis
Iyy = 1; %Moment of Inertia along y-axis
Izz = 1; %Moment of Inertia along z-axis

%% %Initial Conditions
x_0 = 0; %Initial value of position along x
y_0 = 0; %Initial value of position along y
z_0 = 0; %Initial value of position along z
u_0 = 0; %Initial value of velocity along x_b
v_0 = 0; %Initial value of velocity along y_b
w_0 = 0; %Initial value of velocity along z_b
p_0 = 0; %Initial value of angular rate along x_b
q_0 = 0; %Initial value of angular rate along y_b
r_0 = 0; %Initial value of angular rate along z_b
phi_0 = 0; %Initial value of Euler angle around x
theta_0 = 0; %Initial value of Euler angle around y
psi_0 = 0; %Initial value of Euler angle around z

%Combined Initial values in vector form
X0 = [x_0; y_0; z_0; u_0; v_0; w_0; p_0; q_0; r_0; phi_0; theta_0;
psi_0];

%% %Command Input


%PI Controller Gains for Z
Kiz = 10;
Koz = 0.5;
```

```
Kozi = 0;
%PI Controller Gains for Phi
Kip = 3;
Kop = 1;
Kopi = 0;
%PI Controller Gains for Theta
Kiq = 3;
Koq = 1;
Koqi = 0;
%PI Controller Gains for Psi
Kir = 0.5;
Kor = 0.5;
Kori = 0;

%% %Simulation Run
sim('DC2');
```