DEEP LEARNING TECHNIQUES TO ESTIMATE 3D POSITION IN STEREOSCOPIC IMAGERY

THESIS

J. Isaac Nicholson, 2d Lt, USAF

AFIT-ENG-MS-22-M-050

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-22-M-050

DEEP LEARNING TECHNIQUES TO ESTIMATE 3D POSITION IN STEREOSCOPIC IMAGERY

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

J. Isaac Nicholson, B.S.Co.E.

2d Lt, USAF

March 24, 2022

AFIT-ENG-MS-22-M-050

DEEP LEARNING TECHNIQUES TO ESTIMATE 3D POSITION IN STEREOSCOPIC IMAGERY

THESIS

J. Isaac Nicholson, B.S.Co.E.
2d Lt, USAF

Committee Membership:

W. Blair Watkinson II, Ph.D
Chair

Scott L. Nykl, Ph.D
Member

Clark N. Taylor, Ph.D
Member

Abstract

Current automated aerial refueling (AAR) efforts utilize machine vision algorithms to estimate the pose of a receiver aircraft. However, these algorithms are dependent on several conditions such as the availability of precise 3D aircraft models; the accuracy of the pipeline significantly degrades in the absence of high-quality information given beforehand. We propose a deep learning architecture that estimates the 3D position of an object based on stereoscopic imagery. We investigate the use of both machine learning techniques and neural networks to directly regress the 3D position of the receiver aircraft. We present a new position estimation framework that is based on the differences between two stereoscopic images without relying on the stereo block matching algorithm. We analyze the speed and accuracy of its predictions and demonstrate the effectiveness of the architecture in mitigating various visual occlusions.

Acknowledgements

I would like to begin by thanking God for all of His blessings to me. It is only through Him that I have the capability to perform this research.

My family has my deepest gratitude for always helping me, and for constantly encouraging me to do my best. Their support has been invaluable to me during this process.

I want to thank my church family for their love and care for me during this time. I am blessed by the relationships I have developed, the benevolence they have shown me, and their prayers for me.

I wish to thank my advisor, Col Watkinson, for his aid in my research, and for providing direction and pushing me to do more than I believed I was capable of doing.

I am appreciative of my committee, Dr. Nykl and Dr. Taylor, for their guidance during this process. Their feedback has been crucial in refining my research.

Finally, I would like to thank all of my friends. The camaraderie and friendships I have gained during this time are truly amazing.

J. Isaac Nicholson

Table of Contents

Page

Abstract iv

Acknowledgements v

List of Figures viii

List of Tables xi

I. Introduction 1

1.1 Motivation 1
1.2 Research Objectives 2
1.3 Thesis Overview 3

II. Regression Model 5

2.1 Regression Model Background and Literature Review 5
2.1.1 Automated Aerial Refueling Related Work 5
2.1.2 Regression Model Related Work 8
2.2 Regression Model Methodology 10
2.2.1 Regression Model Data Generation 11
2.2.2 Regression Model Architectures 12
2.3 Regression Model Results and Analysis 16
2.3.1 Regression Model Results 16
2.3.2 Regression Model Analysis 20
2.4 Regression Model Future Work 23

III. Convolutional Neural Networks 24

3.1 CNN Background and Literature Review 24
3.1.1 CNN Background 24
3.1.2 CNN Related Work 25
3.2 CNN Methodology 27
3.2.1 CNN Data Generation 27
3.2.2 CNN Architecture Details 31
3.3 CNN Results and Analysis 34
3.3.1 CNN Results 34
3.3.2 CNN Analysis 37

IV. Recurrent Neural Networks 39

4.1 RNN Background and Literature Review 39
4.1.1 RNN Background 39

Page

4.1.2 RNN Related Work . . . 40
4.2 RNN Methodology . . . 42
4.2.1 CNN Development . . . 43
4.2.2 RNN Data Generation . . . 45
4.2.3 RNN Architecture Details . . . 47
4.3 RNN Results and Analysis . . . 53
4.3.1 RNN Results . . . 53
4.3.2 RNN Analysis . . . 55

V. Conclusions . . . 63

5.1 Future Work . . . 63

Appendix A. CNN Model Architectures . . . 67

Bibliography . . . 74
Acronyms . . . 80

List of Figures

Figure		Page
1	A visual representation of the HOG technique. Image from [1].	10
2	Some example images used in training the machine learning models. Each image is annotated with the truth location ($\{x_{offset}, y_{offset}, z_{offset}\}$) underneath.	12
3	The coordinate system utilized. The red axis represents the X-axis, the green axis represents the Y-axis, and the blue axis represents the Z-axis. All arrows shown indicate the positive direction in that axis.	13
4	The distributions of the underlying data in each axis.	14
5	The errors in prediction versus actual values for the X, Y, and Z output values.	22
6	An example of the data distributions in each axis for each CNN dataset. This distribution is specifically from the full occlusion dataset, but all datasets used the same positioning methodology and thus have extremely similar distributions.	28
7	An example difference image used in training the CNN models.	29
8	An example difference image with background noise, boom occlusion, and rotational noise.	31
9	A visual representation of the convolutional layers in the YOLO-inspired architecture.	33
10	A graph of the errors in prediction of the CNN model as opposed to the true values along all axes.	36
11	A graph of the training and online validation RMSE (in meters) captured as the model trained.	38
12	The data distributions in each axis for the training set of the 12-50 meter CNN dataset.	44

Figure Page

13 An example of an image sequence used during training. Note that the only position annotated is the position of the receiver aircraft in the final image. 49

14 The data distributions in each axis for the training set of the 12-50 meter CNN-RNN dataset. 50

15 The data distributions in each axis for the training set of the 12-20 meter CNN-RNN dataset. 51

16 A visual depiction of the final CNN-RNN model architecture used to predict the location of the receiver aircraft. The CNN layers are encapsulated in the TimeDistributed layers. 52

17 A graph of the errors in prediction of the CNN model on the 12-50 meter CNN test set as opposed to the true values along all axes. All values are in meters. 57

18 A graph of the errors in prediction of the CNN model on the 12-50 meter CNN test set at the 12-20 meter range. This shows us how well the 12-50 meter CNN model performs near contact location. All values are in meters. 58

19 A graph of the errors in prediction of the CNN-RNN model on the 12-50 meter CNN-RNN dataset as opposed to the true values along all axes. All values are in meters. 60

20 A graph of the errors in prediction of the CNN-RNN model on the 12-20 meter dataset as opposed to the true values along all axes. All values are in meters. 62

21 The shallow *Small* architecture used in CNN research. 67

22 The convolutional layers of the *Medium* architecture used in CNN research. 68

23 The dense layers of the *Medium* architecture used in CNN research. 69

24 The convolutional layers of the *Large* architecture used in CNN research. 70

Figure Page

25 The dense layers of the *Large* architecture used in CNN research. 71

26 The convolutional layers of the *YOLO-inspired* architecture used in CNN research. 72

27 The dense layers of the *YOLO-inspired* architecture used in CNN research. 73

List of Tables

Table Page

1 This table shows all hyperparameters tested in the machine learning models. Note that linear regression models do not use the regularization parameter (C); these are only used in SVR models. 15

2 This table shows the training and validation error given by the various models tested. The comparison between them was used to prevent the use of over-fitting models. 17

3 This table shows the hyperparameters of the models selected by the halving grid search. 18

4 This table shows the validation error given by the various network architectures tested on the different datasets. The best model for each dataset are highlighted in bold. All numbers represent centimeters of RMSE on the offline validation set. 34

5 This table shows the error given by the best network architectures on the test sets. In each case, the model tested was the *YOLO-inspired* architecture, as these architectures had the lowest validation errors. 35

6 This table shows the runtimes of the different architectures in order to calculate a numerical prediction from an image. All times shown are in milliseconds. 37

7 This table shows the validation error given by the various CNN architectures tested on the 12-50 meter distance dataset. The best performing model and RMSE is highlighted in bold. 54

8 This table shows the validation error given by the various CNN-RNN architectures tested on the 12-50 meter CNN-RNN dataset. The best performing model and RMSE is highlighted in bold. 55

9 This table shows the validation error given by the various CNN-RNN architectures tested on the 12-20 meter distance dataset. The best performing model and RMSE is highlighted in bold. 55

Table Page

10 This table shows the comparison between the best-performing CNN models and the best performing CNN-RNN models on the two distance distributions. All numbers reported are centimeters of RMSE on the test set. 56

DEEP LEARNING TECHNIQUES TO ESTIMATE 3D POSITION IN STEREOSCOPIC IMAGERY

I. Introduction

1.1 Motivation

Aerial refueling is an integral part of modern warfare, as it extends the reach and effectiveness of airpower. First demonstrated in 1923, modern air-to-air refueling involves a tanker refueling a receiver via a boom or drogue and has become a crucial force multiplier and a critical enabler of global reach. Air-to-air refueling involves two aircraft operating within 15 feet (4.572 meters) of one another during the fuel transfer; maintaining aircrew proficiency comes with a substantial training cost. An emerging area of interest is automated aerial refueling (AAR). Automating the mid-flight refueling process opens up new capabilities for all missions, especially for unmanned aerial vehicle (UAV) missions. The latency inherent in UAV's current control mechanism makes it impossible for operators to exercise the real-time fine control necessary for air refueling. Therefore, UAV missions are limited in duration by the amount of fuel each aircraft can hold. In addition to remarkably extending the UAV mission capability, AAR can drastically reduce training costs.

Estimating pose from visual spectrum imagery is desirable as modern tanker aircraft have rear-facing cameras and the visual spectrum is robust to interference or jamming that may occur in a combat environment. Additionally, utilizing the existing technology on the receiver aircraft is desirable over technologies such as LIDAR or radar as it does not require the installation of new technologies to the current

tanker aircraft, and is thus cost-efficient. Convolutional neural network (CNN) models can effectively generalize to new and unseen data, and, coupled with recurrent neural network (RNN) models they can become robust to various visual degradations and occlusions. Accurately estimating the pose of an aircraft is a crucial step in the world of automated aerial refueling, as the process requires precise and timely pose estimations of the receiver aircraft relative to the tanker aircraft in order to avoid incurring any damage to either aircraft.

1.2 Research Objectives

The focus of this research is to explore the feasibility in using both machine learning and deep learning architectures to estimate the 3D position of a receiver aircraft using stereoscopic imagery. Current efforts rely on a pre-rendered 3D model of the receiver aircraft in order to accurately estimate its position, but this 3D model must be given to the algorithms beforehand in order to function appropriately. Further issues arise if the 3D model provided beforehand does not accurately reflect the 3D model computed in the stereoscopic imagery. We seek to make our architecture independent of any computationally expensive pre-processing methods or extraneous information that is not inherently present in the simulated stereoscopic imagery. As real-world data is costly to obtain, current research focuses on using the virtual world as an estimator for the real world. If successful, our work will prove that deep learning architectures can be made robust to a wide range of visual degradations present in the input imagery, so that any further work is only constrained on what is able to be simulated in the virtual world. This work will mainly be compared with current AAR efforts utilizing the iterative closest point (ICP) algorithm which is the current computer vision state-of-the-art. This research lays the groundwork in integrating CNN and RNN models to accurately estimate aircraft pose in the face of various

visual degradations and occlusions. This research seeks to investigate the following claims:

1. Pure machine learning models are sufficient to effectively estimate a 3D position of an object.

2. A CNN architecture can yield better results than a pure machine learning model, but still degrades in the face of occlusions.

3. An RNN architecture can improve upon a CNN architecture by providing extra data based on previous locations, thus reducing the inaccuracy caused by visual occlusions.

If successful, our system could be used as an alternative architecture for position (and later 6D pose) estimation in the AAR field, or as a starting framework in any field that seeks to directly regress 3D position from stereoscopic imagery using deep learning architectures.

1.3 Thesis Overview

One fundamental principle of machine learning and deep learning is to utilize the simplest model that is capable of learning the desired task. In order to follow this principle, this thesis is split into three main sections. Chapter II investigates a pure machine learning approach using support vector machines to regress the 3D position of a pure red sphere. We find that the machine learning model can give a general estimate, but is not accurate enough to support current AAR efforts. Chapter III covers a deep learning approach using CNNs to estimate the 3D position of an aircraft. We find that these models are very fast, and do nearly as well as the current industry standard at contact locations. However, these models degrade as the receiver aircraft moves further away from the tanker, and perform slightly worse in the face of

occlusions. Next, chapter IV analyzes an approach using CNNs and RNNs in order to predict the position of an aircraft using sequences of images as input. We find that these models are slower in their predictions, but enhance the accuracy of the prediction both at farther ranges and in the face of visual degradation. We find that a CNN-RNN model can provide predictions that are competitive with the current state-of-the-art, both at contact location and at farther distances, and despite visual occlusions. Finally, chapter V discusses the implications of this work, and provides avenues for future research in this area.

II. Regression Model

This chapter covers our work in estimating 3D position of an object using machine learning regression models. In section 2.1, the background and related work of automated aerial refueling (AAR) and machine learning methods are discussed. Our methodology is covered in section 2.2. All relevant results are covered in section 2.3, as well as an analysis of these results.

2.1 Background and Literature Review

2.1.1 Automated Aerial Refueling

Early work in the AAR field utilizes an extended Kalman filter which combines traditional navigation methods (global positioning systems, inertial navigation systems, etc.) with a new stereo vision pipeline in order to predict the location of the receiver aircraft in relation to the tanker aircraft [2]. They found that they could accurately estimate the location of the receiver with centimeter-level accuracy. However, much of the accuracy in the system came from the stereo camera portion. These insights prompted a focus on the field of computer vision due to the general availability of the visual spectrum and the existing cameras on current tanker aircraft.

The current AAR industry standard utilizes the iterative closest point (ICP) approach [3]. This approach is purely algorithmic, and thus yields consistent results. In this pipeline, the stereo block matching algorithm is first run on stereoscopic imagery in order to generate a sensed point cloud. The ICP algorithm serves to align this sensed point cloud with a truth point cloud given by a 3D model of the aircraft to estimate the pose of the receiver aircraft. This is done by first computing the center of mass of both point clouds. Once the center of the two clouds are found, a rotation which aligns the two point clouds can be mathematically computed. This process

is repeated until a local minimum is found or an iteration count is reached. The method proposed by [3] yielded results within 10 cm of the actual location based on Euclidean distance, and took around 180 milliseconds to complete the ICP portion of the pipeline.

The work of [4] continues this line of research by integrating the ICP pipeline into a virtual world. Their findings validate the virtual world as an accurate predictor of the AAR scenario, and allowed future work with a more realistic virtual setting in which to conduct experiments.

The boom caused significant issues with the initial ICP pipeline, as extra sensed points would be detected on the boom, while certain parts of the aircraft would be occluded by the boom and thus not included in the sensed point cloud. Due to the nature of the ICP algorithm, this often caused the algorithm to converge to an incorrect minimum where the orientation would be vastly off, and the location error would exceed the 10 centimeter requirement to perform aerial refueling operations. [5] uses shadow volumes to mitigate occlusions of the receiver aircraft caused by the boom. The effects of the boom were ultimately mitigated by implementing a distance filter, then creating a shelled model of the receiver which excluded all detected points normally occluded by the boom. This new shelled model would appropriately converge.

In the pipelines used by [4] and [5], the sensed point cloud would be generated by OpenCV's stereo block matching algorithm. In order to more accurately predict the location of the receiver aircraft with this algorithm, the highest resolution of images is preferred. However, running the stereo block matching algorithm on the full image at the highest resolution is very slow, and limits the overall speed of the pipeline. The use of neural networks to accelerate the ICP pipeline by locating the aircraft was the focus of [6]. In order to mitigate this timing bottleneck, a bounding box was applied

around the aircraft through the use of deep learning methods. This sped up the ICP pipeline by only running the stereo block matching algorithm at full resolution around the bounding box. The rest of the image could be run at a lower resolution or skipped entirely, increasing the speed at which the overall pipeline could be run.

After accelerating the stereo block matching portion of the pipeline, the use of Delaunay triangulations was used to accelerate the nearest-neighbor portion of the ICP pipeline [7]. This work focused on parallelization of the nearest neighbor matching on a GPU, and was able to achieve a 25x speedup using this method. As a result of this work, the entire pipeline, which includes both the stereo block matching algorithm and the ICP algorithm, could be completed in around 300 milliseconds if 4K images are used, or around 90 milliseconds if 2K images are used, with the nearest neighbor matching portion only requiring about 35 milliseconds of that time. This allows a refresh rate of 3 to 10 estimates per second depending on the image resolution used, allowing for timely control of the receiver aircraft.

In order to maximize accuracy, the stereo cameras need to be precisely calibrated before the stereo block matching takes place. Poor camera calibrations lead to poor sensed point clouds and a decrease in the accuracy of the ICP system. The use of convolutional neural network (CNN) models to perform stereo camera calibrations was investigated by [8]. This step is done as a pre-processing step and thus does not slow down the overall pipeline, but increases the accuracy of the point cloud produced by the stereo block matching algorithm.

All of these efforts have cumulatively made the ICP algorithm the foremost pose estimator in AAR research efforts. However, the ICP algorithm makes several critical assumptions, and performance degrades once these assumptions are not met. The ICP algorithm assumes the existence of an accurate 3D reference model of the receiver aircraft with which to compare the sensed model generated by the stereo block

matching algorithm. It is not adaptive to different aircraft configurations or weapons loadouts, as different loadouts would have different reference models, which would need to be accounted for in a real world scenario. The ICP algorithm also converges to the closest local minimum, which creates problems in the instance of a receiver aircraft that has significantly deviated from a traditional flight path. As the ICP algorithm needs a fairly accurate initial position with which to improve upon, a sufficiently incorrect initial guess could cause the algorithm to converge to an incorrect local minimum and thus provide inaccurate pose estimations. Finally, the ICP algorithm particularly degrades when there is an attempt to map points in the real-world that do not exist in the model. This occurs when the stereoscopic vision system detects parts of the boom or features on the receiver aircraft not present in the model. Neglecting to filter these additional points causes ICP to improperly converge and give poor estimates for the location of the aircraft.

2.1.2 Regression Model Related Work

Several research efforts have been performed in order to use machine learning techniques to estimate the position and orientation of a 3D object. We discuss these research efforts here, and comment on their applicability to our research.

The work of [9] seeks to use machine learning algorithms to estimate the pose of a tanker from the receiver's point of view. They achieve this by splitting their approach into three separate algorithms, and refining each algorithm based on a generated scenario. However, similar to ICP, they use a reference model of the tanker, and their work seeks to estimate the pose of the tanker from the perspective of the receiver.

In order for image data to be converted into numerical data that can be processed by machine learning methods, the important features must first be extracted. The

histogram of oriented gradients (HOG) technique is a powerful feature extractor that is often used to convert image data into numerical features suited for machine learning algorithms. The HOG technique was introduced by [10]. This technique takes images as input data, and outputs an array of histograms of oriented gradients. The technique begins with a cell size of one or more pixels. It then calculates the difference in pixel values along the major axes (X- and Y-directions), and stores these in a vector. After computing all sub-components of each cell's vector, it simplifies the cell down to the principal components of a vector: magnitude and direction. This process is repeated for each possible cell in the image, and each cell is then normalized within a block (where each block is typically a square array of cells) to correct for differences in lighting. The normalization process is performed on many overlapping blocks, until all cells have been normalized. The final output is a vector of numerical histograms that can be fed into a machine learning model. This model inherently performs some feature reduction, although the exact number of final features depends on several hyperparameters. A visualization is shown in fig. 1.

[10] used this technique as a feature extractor in order to classify whether a human was present in a video. Since then, this technique has been used as a powerful feature extractor in many other scenarios. The work of [11] uses the HOG technique to classify handwritten digits, proving it to be a powerful feature extractor. [12] uses the technique to estimate a course orientation of a vehicle, and [13] extends the feature extractor into the 3D domain in order to classify traffic activity in urban settings.

[14] utilizes a machine learning architecture for blob tracking through images. They utilize a form of principal component analysis (PCA) in order to help them create a machine learning architecture that can track an object of interest through several video frames.

Lastly, [15] seeks to combine the use of HOG features in order to regress the 6D

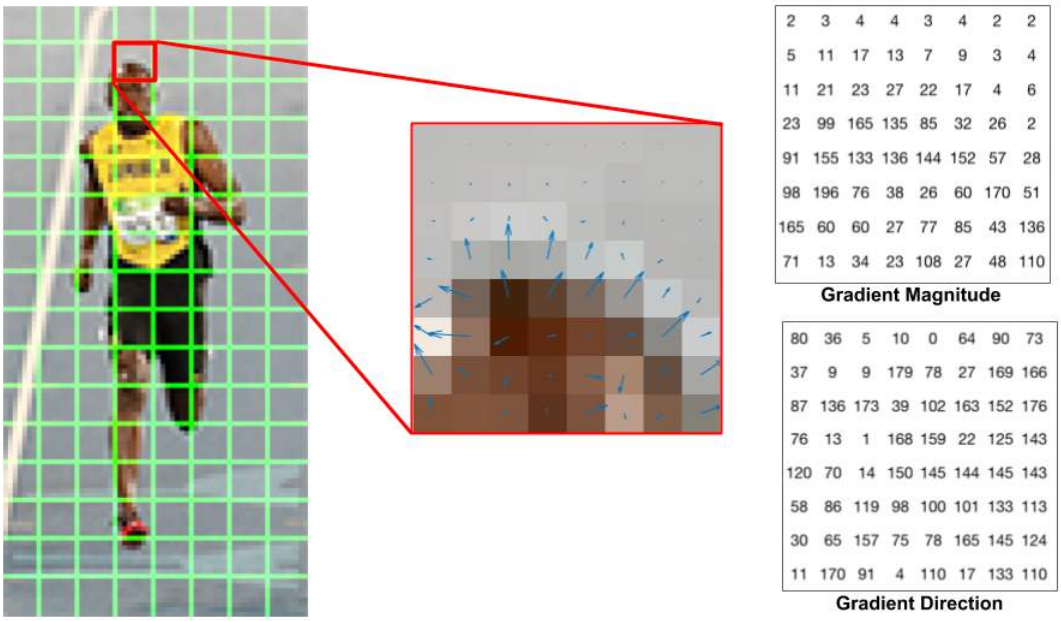


Figure 1: A visual representation of the HOG technique. Image from [1].

pose of an object. This work focuses on edge detectors paired with machine-learning techniques in order to regress the 6D position and orientation of objects. However, similar to ICP, their work utilizes a pre-rendered 3D model of the objects in question.

2.2 Methodology

The main focus of this work is to develop and test a method of position estimation for the receiver aircraft that is largely independent of many limiting factors upon which the current ICP industry standard relies. Our work seeks to reduce or eliminate the need for pre-built models of the receiver aircraft and stereo block matching, among others. By creating a system that is reliant only on input data, our system can be made robust to any limiting factor that is able to be simulated in the virtual world.

We trained three different linear regression models and analyzed them on how well they predict the position of a sphere given an array of histograms of oriented gradients. This array of HOGs comes from the pre-processing of an image. Model performances

are reported in terms of root mean squared error (RMSE) in section 2.3.1. The final model was tested against a pre-separated test set to determine how well the model generalizes to unseen data.

2.2.1 Data Generation

The data used was simulated using the AftrBurner engine developed by Dr. Scott Nykl. This engine, first introduced in [16], is a derivative of the STEAMiE engine as investigated in [17]. Our data is comprised of labeled images of red spheres, each with the offset vector of the sphere relative to the camera, in meters, as labeled truth data ($\{x_{offset}, y_{offset}, z_{offset}\}$). Some examples of images used for training are shown in fig. 2. The coordinate system used is shown in fig. 3. In this coordinate system, the camera is located at the origin and X-values increase to the aft of the tanker aircraft; Y-values increase from right to left (as viewed from the camera, looking aft along the X-axis), and Z-values increase with altitude.

In the data generated, x_{offset} values range from 20 to 44 meters in 4 meter increments. Each x_{offset} yields a Y-Z plane, on which the sphere must be fully visible. y_{offset} and z_{offset} values begin on the bottom-right of the visible area and sweep from right to left then from bottom to top over 4-meter increments, at whole meter values. 229 data points were generated using this strategy. This methodology yields a roughly uniform distribution of spheres along the y_{offset} and z_{offset} values, and a skewed distribution among x_{offset} values. These distributions are shown in fig. 4.

Within the HOG technique, there are several hyperparameters that can be defined:

cell size the pixel size of the HOG cell within which gradient values are calculated

block size the number of HOG cells which are combined into one histogram

number of bins the number of bins into which the magnitudes and gradients of the cells are placed

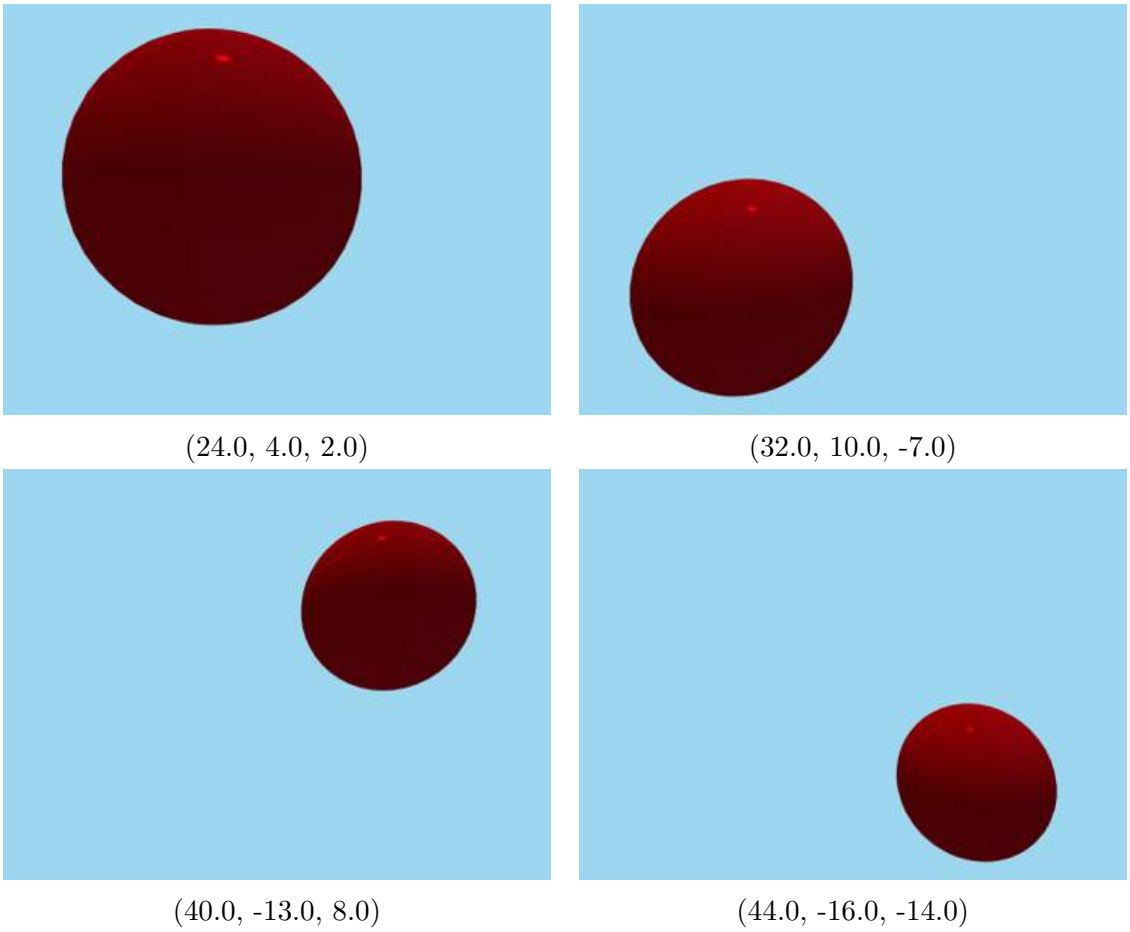


Figure 2: Some example images used in training the machine learning models. Each image is annotated with the truth location ($\{x_{offset}, y_{offset}, z_{offset}\}$) underneath.

In addition to the HOG technique hyperparameters, we tested color and grayscale images, various values of the regularization parameter (C) when a support vector regressor (SVR) kernel was used, and varying values for PCA. The full list of hyperparameters and all values tested are shown in table 1.

2.2.2 Architecture Details

For our supervised learning task, we test linear regression models and SVR models. We explored SVR models with both linear and Gaussian radial basis function (RBF) kernels. The output of each model is a vector denoting relative position of the sphere

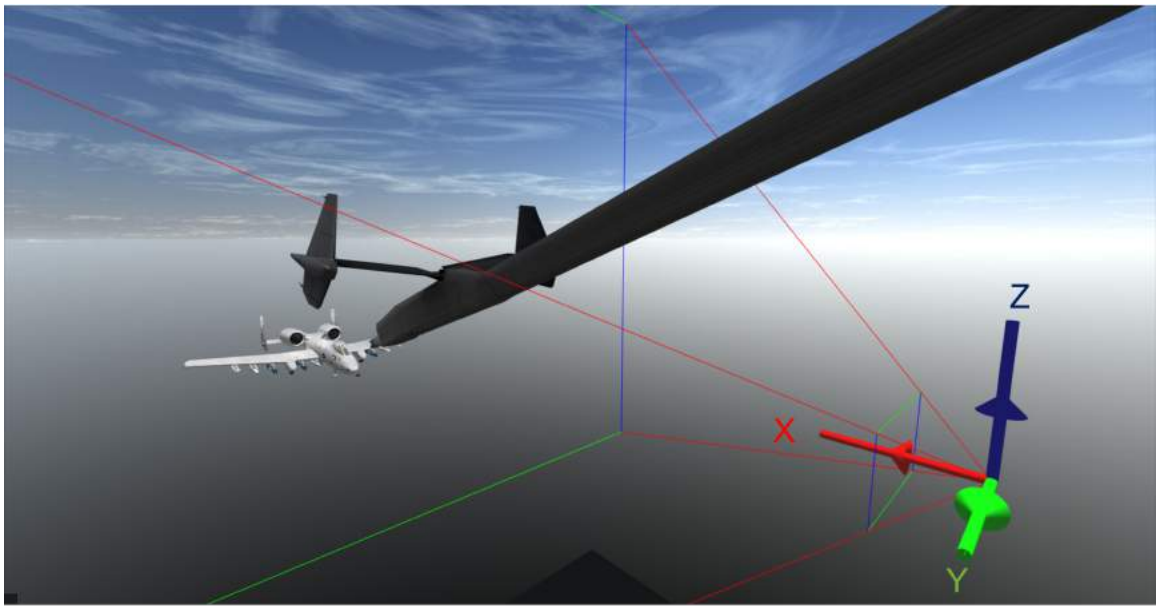


Figure 3: The coordinate system utilized. The red axis represents the X-axis, the green axis represents the Y-axis, and the blue axis represents the Z-axis. All arrows shown indicate the positive direction in that axis.

in relation to the camera ($\{\hat{x}_{offset}, \hat{y}_{offset}, \hat{z}_{offset}\}$). We can control how much freedom our models have to fit to the training data with the regularization parameter, or C. Low values of C allow the model more flexibility in fitting to the training data, but increase the risk that the model simply over-fits to the training data and is unable to generalize to new data. High values of C allow the model less freedom in fitting the training data, reducing the risk of over-fitting. The disadvantage of a high value of C is that the model generated might be too simple to fully capture the full complexity of the data. Finally, the PCA technique reduces the dimensionality of the data by selecting only the most impactful features of the data. The hyperparameter within this technique represents how much of the variance in the data should be explained by the features chosen. For example, a PCA value of 0.7 means that the most impactful features will be sequentially selected until 70% of the initial variance in the data is represented by the new feature set.

We used RMSE as the scoring function. RMSE represents the distance error, in

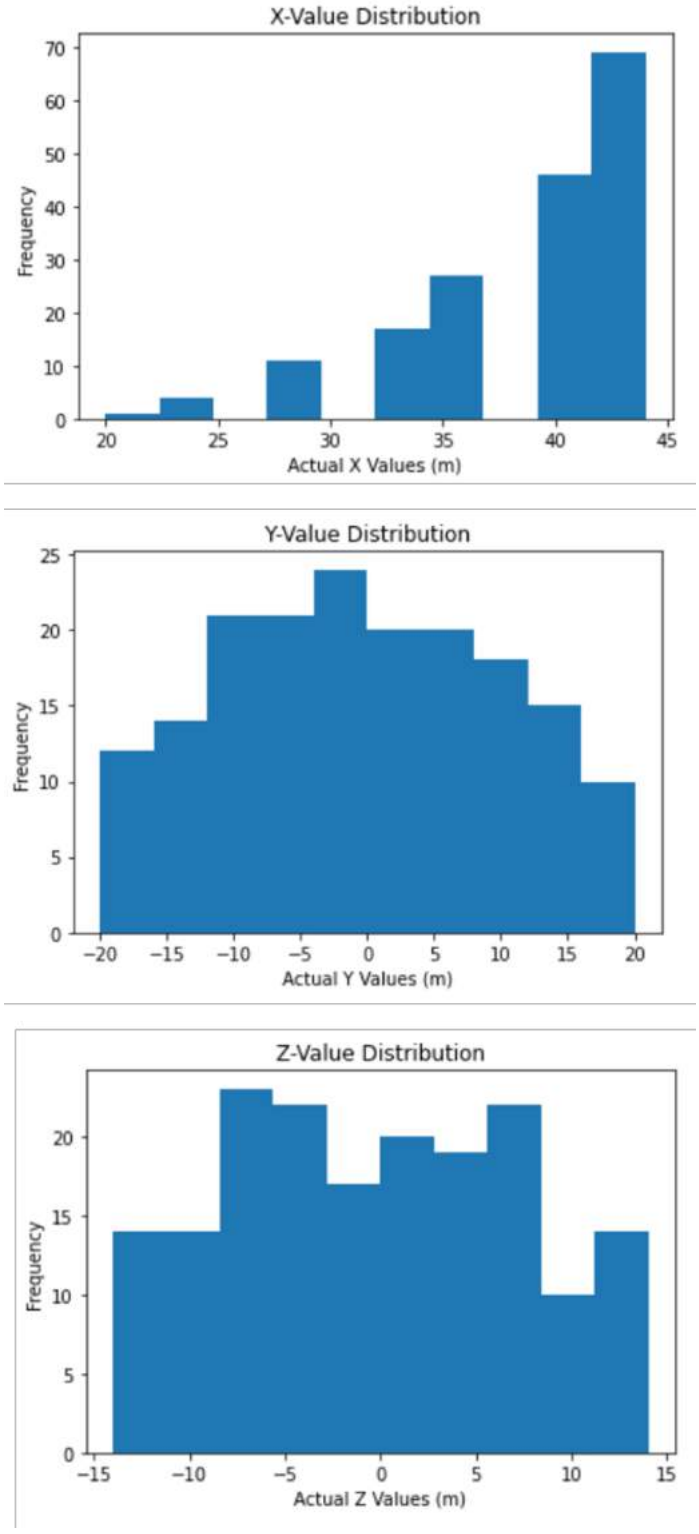


Figure 4: The distributions of the underlying data in each axis.

Hyperparameter	**Values Tested**
Color	Grayscale, Color
PCA	0.5, 0.7, 0.9
C (SVR models)	0.01, 1, 10, 30
Kernel (SVR models)	RBF, linear
HOG Hyperparameters	
Cell Size	8x8, 16x16
Number of bins	4, 5, 9
Cells per block	1, 2, 3

Table 1: This table shows all hyperparameters tested in the machine learning models. Note that linear regression models do not use the regularization parameter (C); these are only used in SVR models.

meters, between the predicted location and the actual location of the sphere. Finally, in order to test various hyperparameter sets, a halving grid search was utilized, with K-Fold cross-validation where K=5. The halving grid search technique is similar to the grid search technique in that initially all combinations are tested, albeit with a smaller resource pool. The search then eliminates low performing hyperparameter combinations, and devotes more resources to the better performing combinations. This process is repeated until the best hyperparameter combination is found.

In addition to the 10% of observations held out for the test set, we held out 10% of all observations to support over-fitting analysis. With the remaining observations, we applied a halving-grid search to the Cartesian product of all hyperparameter values listed in table 1. The halving grid search generated the model that provided the smallest cross-validation error. If the model over-fit the training data (as determined by the the training RMSE being at least an order of magnitude smaller than the RMSE of the model when applied to the held-out validation data), we removed the particular set of hyperparameters that generated the model, and applied the halving-grid search again. We repeated this process until we had a model that did not over-fit to the training data.

Initially, in our linear regression models, we evaluated models with and without

using the HOG technique. In testing, the grid search would select models without HOG as the best one. We discovered these models were severely over-fitting to the held-out data. We continued analysis exclusively with models incorporating the HOG technique. It was not until we introduced the PCA technique that we could avoid over-fitting and receive accurate estimates from our machine learning models.

For the SVR models, a similar process was followed. Like linear regression, the initial models would initially skip the HOG technique and over-fit to the training set. Once both the HOG technique and PCA were applied, the results were generally reasonable. However, in addition to the HOG and PCA technique, the regularization parameter (C) also had to be tuned to prevent over-fitting of the radial-basis function model.

The halving-grid search method selected a set a of candidate models for further analysis. We evaluated the validation and training error of the 5 best models and chose the best model for final performance analysis against the test set. To conduct this final analysis, we retrained the model using the full set of training data (including the held-out data for over-fitting analysis) and evaluated its performance on the test data. The next section compares the performance of the 5 candidate models and analyzes the final model.

2.3 Results and Analysis

2.3.1 Results

The results of the best models generated by the methodology described in section 2.2 are depicted in table 2.

Technique	
Validation Error (m)	**Training Error (m)**
Linear Regression with only HOG	
3.215286	1.71975e-14
SVR without HOG	
1.734658	0.097682
SVR with HOG	
5.461971	5.052607
SVR with HOG and PCA (Linear Kernel)	
3.891731	3.655514
Linear Regression with HOG and PCA	
3.379964	3.063465
SVR with HOG and PCA (RBF Kernel)	
1.049932	0.566847

Table 2: This table shows the training and validation error given by the various models tested. The comparison between them was used to prevent the use of overfitting models.

Model	**Cell Size**	**Block Size**	**Bin Count**	**PCA**	**C**	**Color**
Linear Regression with only HOG	16x16	1x1	5	N/A	N/A	RGB
SVR with HOG	8x8	1x1	5	N/A	1	RGB
SVR with HOG and PCA (Linear Kernel)	8x8	1x1	5	0.7	30	Grayscale
Linear Regression with HOG and PCA	8x8	1x1	5	0.7	N/A	Grayscale
SVR with HOG and PCA (RBF Kernel)	8x8	1x1	5	0.5	30	Grayscale

Table 3: This table shows the hyperparameters of the models selected by the halving grid search.

For this task, we anticipated the Gaussian kernel SVR would yield the best performance. We also anticipated the $\hat{y}_{offset}$ and $\hat{z}_{offset}$ values would be much more accurate than $\hat{x}_{offset}$ values, due to the nature of the HOG method and the fact that as the sphere is moved away from the camera, the same offset distance in either the Y- or Z-axes will yield a smaller difference in the pixels of the captured images. While the first prediction was accurate, the second one was incorrect, as the largest errors were observed in the y_{offset} predictions. One possible explanation for the increased inaccuracy of the $\hat{y}_{offset}$ values would be the effective range of the data distribution in each axis. Whereas the x_{offset} and z_{offset} had ranges of roughly 25 and 30 meters, respectively, the y_{offset} distribution had a 40 meter range of values. With the increased range, the models were simply incapable of extrapolating well to the extreme y_{offset} values, and accuracy along this axis suffered.

The final selected model (shown as the last model in table 2 and table 3) was a support vector regressor using a radial-basis function kernel, where both the HOG and PCA techniques were applied to the input data. The hyperparameters used for the pipeline were a C of 30 with respect to the support vector regressor and a PCA value of 0.5 (indicating that features were selected until 50% of the variation in the original training data was explained by the chosen subset of features). For the histogram of oriented gradients technique, the best parameters were 5 orientation bins, a cell size of 8x8 pixels, and a block size of 1 cell per block. Additionally, grayscale images performed slightly better than color images, but this hyperparameter seemed to provide the least difference in comparison to the other parameters.

The first several models would bypass the regularization techniques (HOG preprocessing, high regularization parameters, etc.) and simply over-fit to the data. These models appeared to do extremely well during training, but had poor results on the validation set. This is because the data had too many features, and was suffering

from the curse of dimensionality: with the large amount of features relative to the number of data points, the models were prone to over-fitting. By using PCA to reduce the number of features, we eliminated the over-fitting problem. Once regularized, the pure linear regression model actually outperformed the SVR model with a linear kernel. However, as neither the linear model nor the SVR model with the linear kernel provided the lowest validation error of all models, they were not selected, and the test set was not used on these models.

Once all models were investigated and the final model was selected and trained, the test set was used to predict performance on unseen data. The final model yielded a test RMSE of 1.145 meters, indicating that, on average, the predictions made by the model would be within 1.145 meters of the actual location. The R^2 value was also calculated to be 0.9754. This indicates that roughly 97.54% of the variation in the true output was captured by the model's outputs. This indicates a successful model that is able to generalize well to unseen data.

2.3.2 Analysis

We performed an analysis on both the training and test sets, as shown in fig. 5. From this we see that the residuals for the $\hat{x}_{offset}$ are of a different form than those of $\hat{y}_{offset}$ and $\hat{z}_{offset}$. However, we also see that the predictions get worse for the X values as the distance from the camera increases (as predicted). We suspect this is due to the skewed nature of the data in the X-axis; there is a lack of sufficient data to allow the model to generalize at close range, which creates errors at low values of $\hat{x}_{offset}$ on the test set. This is not desirable, as during an aerial refueling scenario it is critical that the model be accurate in the closer $\hat{x}_{offset}$ positions in which the receiver aircraft will be refueling.

We notice a similar trend in both $\hat{y}_{offset}$ and $\hat{z}_{offset}$ values in that the model predicts

the median values very well, but performs poorly on both very high and very low values. This would indicate that there is some non-linearity that is occurring in the y_{offset} and z_{offset} values not being captured by the model. This makes sense, as the way in which the size of a sphere changes in the Y- and Z- axis as projected onto the image plane is non-linear with respect to changes in distance from the camera. One way to mitigate this effect would be to increase C to allow the model to represent less linear functions. However, this also produces a significant risk of over-fitting, and hence is not preferred. Ultimately, the need to have the most accurate position predictions at extreme values of y_{offset} and z_{offset} is less critical during an aerial refueling scenario, and does not warrant the risk of over-fitting to correct.

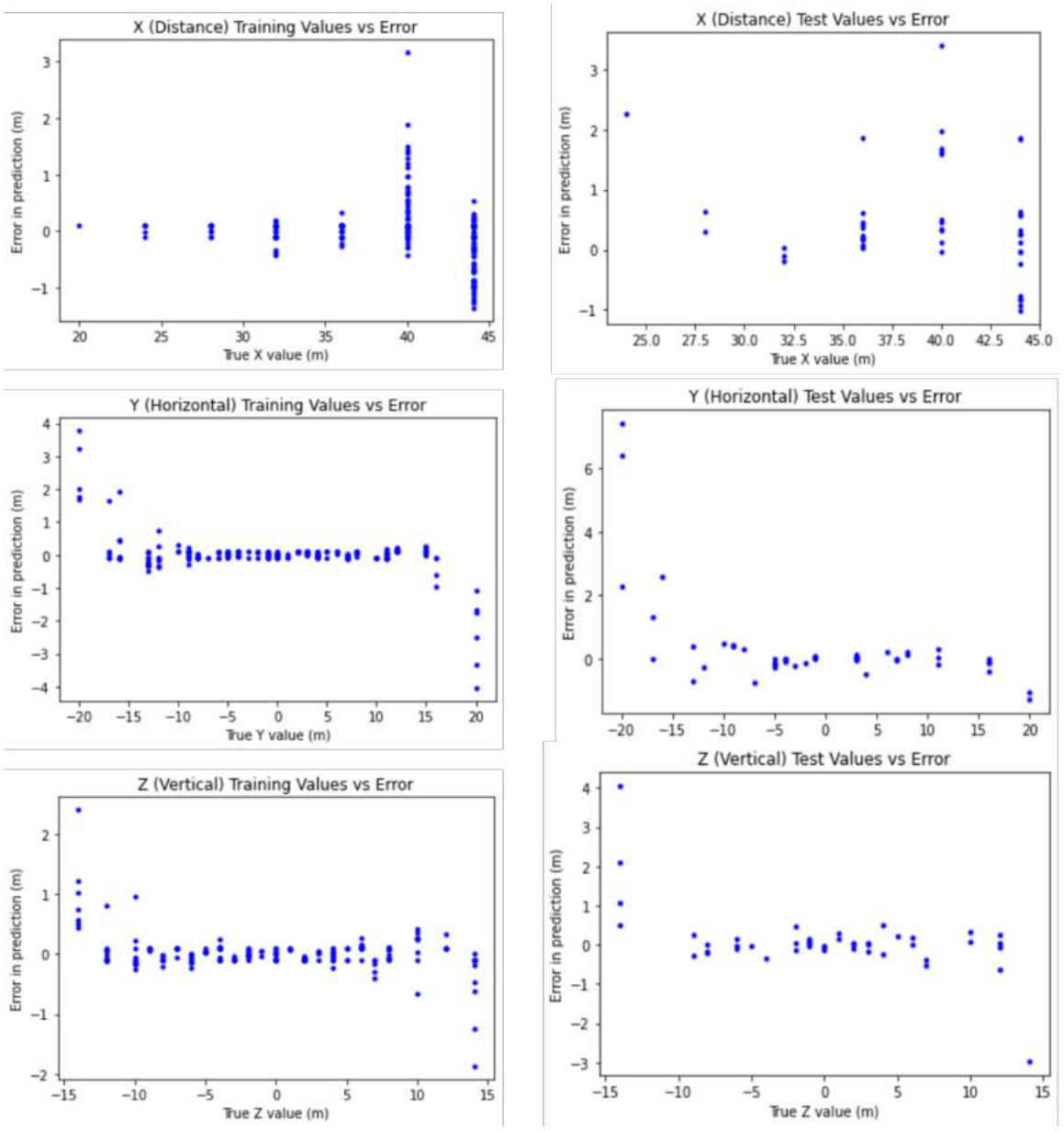


Figure 5: The errors in prediction versus actual values for the X, Y, and Z output values.

2.4 Future Work

In the area of machine learning, there are several avenues that could be explored in order to enhance the overall performance of the models. The first (and most obvious) would be to generate more data on which to train the model. However, it would also be beneficial to generate data in a smarter manner, so that the distribution of data in all axes is more uniform. Additionally, having a smaller sphere size would allow for a wider variety of y_{offset} and z_{offset} values at low x_{offset} values. In order to better model non-linearity, some possible future endeavors could include the testing of polynomial kernels, the separation of models used to estimate the different output values ($\hat{x}_{offset}$, $\hat{y}_{offset}$, and $\hat{z}_{offset}$), or further pre-processing that more accurately aids in regression tasks as opposed to classification tasks.

As a sphere is vastly more simplistic than an aircraft, we consider the results of this section to be the best-case scenario. Thus, we doubt that the statistical machine learning models explored contain the capability to regress the position of a receiver aircraft within 10 centimeters of its actual location. Therefore, future chapters explore the use of higher performing, deep learning models. In the world of vision-related tasks, CNN models have become the standard method of interpreting spatial data. Chapter III explores these architectures, and investigates their ability to interpret data with significantly more complexity and noise.

III. Convolutional Neural Networks

This section discusses the research performed with convolutional neural network (CNN) models. We begin by discussing background and related work in section 3.1. We then describe the methodology of this research in section 3.2. Finally, the results of our work and an analysis of these results are presented in section 3.3.

3.1 Background and Literature Review

3.1.1 Background

A CNN is a deep learning framework comprised of convolutional layers and fully connected layers. This concept was first explored in depth by [18], as they sought to utilize this network model to distinguish individual handwritten digits. Since that time, it has become the main architecture with which to analyze visual data.

The concept of binocular stereopsis is fundamental in the field of computer vision. This concept allows for the triangulation of an object in three dimensional space given two images and the offset between the two cameras. This process of comparing two images and calculating the disparity is known as stereo block matching. Stereo block matching first finds some feature in one of the two binocular images. It then performs a search for the same feature in a predefined region of the other image. If found, the three dimensional location of that object can be computed using the offset of the cameras and the relative location of the object within each image. Current computer vision efforts heavily utilize this process in order to locate a given object in three dimensions. However, this process is often very slow and computationally intensive, as the search algorithm bottlenecks the process while trying to find the exact pixel where both images match.

As current automated aerial refueling (AAR) efforts investigate using two cameras

on the same plane facing roughly the same direction, we can reasonably assume that the aircraft will appear in both images. In addition, due to the nature of front-facing parallel cameras, objects that are closer to the two cameras will inherently create a larger disparity in the images than objects that are farther from the camera plane. As such, we investigate the use of using difference images, where the left and right binocular images are subtracted and only the differences between the images remain. This allows us to avoid the search process entirely, which increases the rate at which we can make predictions.

3.1.2 Related Work

The use of CNNs to estimate a 6D pose is not new; several existing networks use a CNN architecture to estimate the 6D pose of one (or more) objects. Many utilize a monocular camera, and several use pre-built 3D models to estimate the 6D pose of an object. A number of these architectures predict accurate 6D poses despite occlusions.

In their work, [19] seeks to perform pose estimation using binocular imagery and the Perspective-n-Point (PnP) algorithm. However, in order to do so, they make alterations to the receiver unmanned aerial vehicle (UAV) being used, whereas we seek to avoid making any modifications to the receiver aircraft.

The work of [20] uses detected 3D keypoints while utilizing pre-built models to regress pose. [21] proposes a similar pipeline, but instead uses the PnP and random sample consensus (RANSAC) algorithms to map predictions to pre-built 3D models and generate object pose.

[22] uses known 3D models to build a probabilistic model, then compares observations to the probability distribution. This model was shown to be robust to occlusions, and largely shape-agnostic in its results.

[23] proposes a method in which the separate components of 6D pose estimation

are decoupled. The resulting network independently estimates semantic labeling, 3D translation, and 3D rotation, then combines the individual results for the full 6D pose estimation.

[24] proposes EfficientPose, a new architecture which seeks to create a near real-time model by extending the EfficientDet detection architecture proposed by [25] to estimate the 6D pose. Both EfficientDet and EfficientPose networks split the estimation of translation and rotation into two sub-networks. Finally, [26] propose a single-shot approach inspired by the YOLO computer vision algorithm. Their network initially estimates 3D control points, then uses the PnP algorithm to calculate a 6D pose. This work is unique in that only the bounding box of a known 3D model (as opposed to the full image) is used.

[27] proposes a system in which a CNN model is used to detect key features, and then the distance is computed by using the pixel disparities in binocular images. This work is one of the few that performs an actual numerical measurement rather than simple detection. However, only the distance information is obtained (as opposed to the full 3D position).

[28] present a dataset of stereoscopic images with corresponding high-quality labeled depth images. The purpose of these datasets is to train neural networks to correctly predict the depth of a stereoscopic scene at all points. While similar to our work, we seek to accurately regress the full 3D position of a single object rather than the depth information within a scene, and thus these datasets were not utilized.

In their work, [29] analyzes a Siamese CNN architecture for stereo image rectification. [30] argues that the brain utilizes *missing* areas in binocular images as well as similar patches, and that these regions should also be considered when performing pose estimation from stereoscopic imagery. Traditional stereo block matching seeks to find similar image patches, then utilize epipolar geometry to estimate depth (and

ultimately position).

In reviewing the current state-of-the-art for 6D pose estimation using deep learning techniques, several themes emerge. First, we observe that the task of 6D pose estimation has been proven to be solvable by these networks. However, these networks rely heavily on 3D models. The networks we propose utilize stereoscopic cameras, obviating the need for a 3D model of the aircraft.

3.2 Methodology

3.2.1 Data Generation

In order to effectively train a CNN, a large amount of labeled data must be generated for the network to learn how to generalize results. In order to obtain a dataset of this size, we again utilize the AftrBurner engine to yield images of an aircraft in various locations. For each CNN trained, an A-10 aircraft model was used as the receiver aircraft. The receiver aircraft was simulated in random locations ranging from 12 to 20 meters away from the cameras. The locations were also constrained in that each image needed the aircraft to be within the field of view of the cameras, yielding offsets of up to 8 meters in the Y-axis and 5 meters in the Z-axis. Data was collected with a uniform X-value distribution. As the entire frustum was available to sample, this creates a roughly Gaussian distribution in both the Y- and Z-axes. An example of these distributions are shown in fig. 6.

Each dataset generated consists of 46,656 images. These datasets were then split into training, validation, and test sets. Datasets used to train each CNN consisted of 30,338 images, and 5,353 images were set apart for online validation after each epoch. An offline validation set consisting of 6,299 images was set apart in order to directly compare models with one another. Finally, a test set of the remaining 4,666 images was used to test how well the final selected model would perform on new, unseen

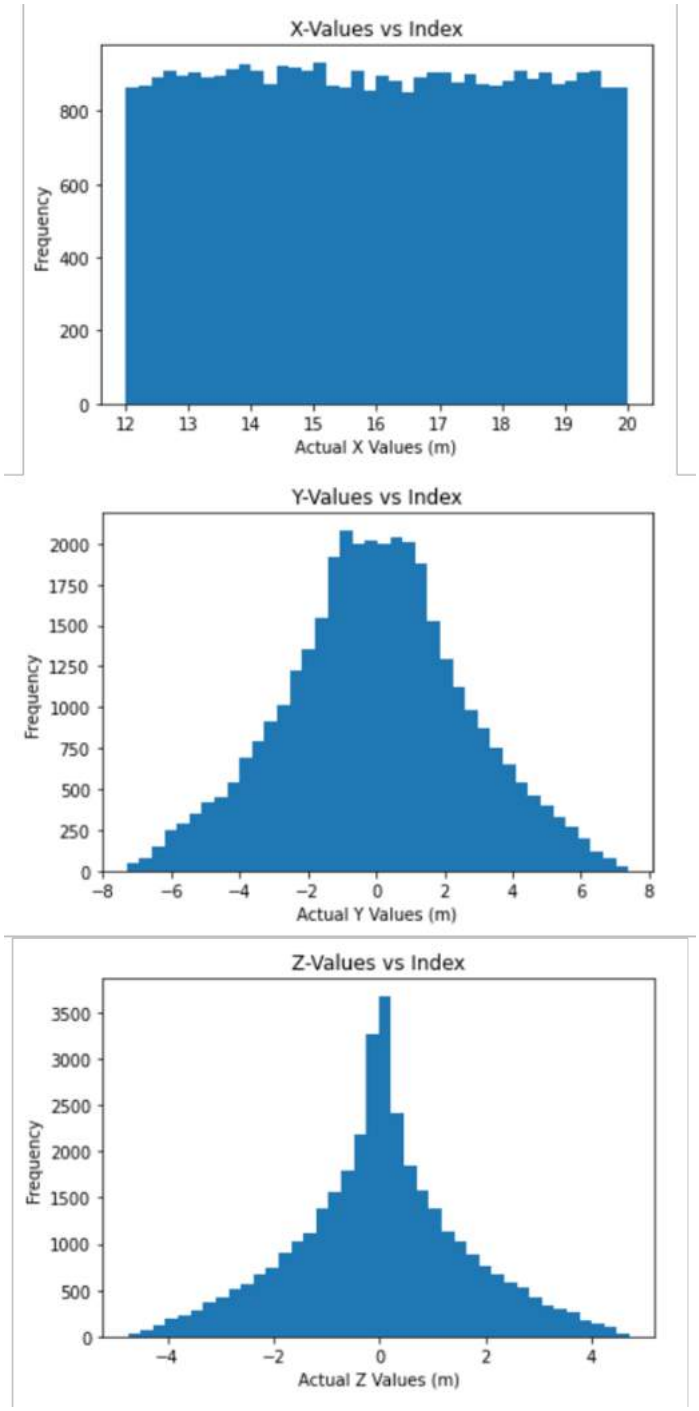


Figure 6: An example of the data distributions in each axis for each CNN dataset. This distribution is specifically from the full occlusion dataset, but all datasets used the same positioning methodology and thus have extremely similar distributions.

data.

We generated images using two cameras positioned 1 meter to the right and 1 meter to the left of the rear of the tanker ((0, 1, 0) and (0, -1, 0) in the coordinate space defined with the rear of the tanker as the origin). Next, we took the absolute difference of these two images. This new "difference image" is saved and is used as the input to our CNN models. An example difference image is shown in fig. 7.

Left Image

Right Image

Difference Image

Figure 7: An example difference image used in training the CNN models.

The images generated were originally 640x360 pixels. This image size was used for the shallow networks, while all other networks used images scaled down to a 320x180

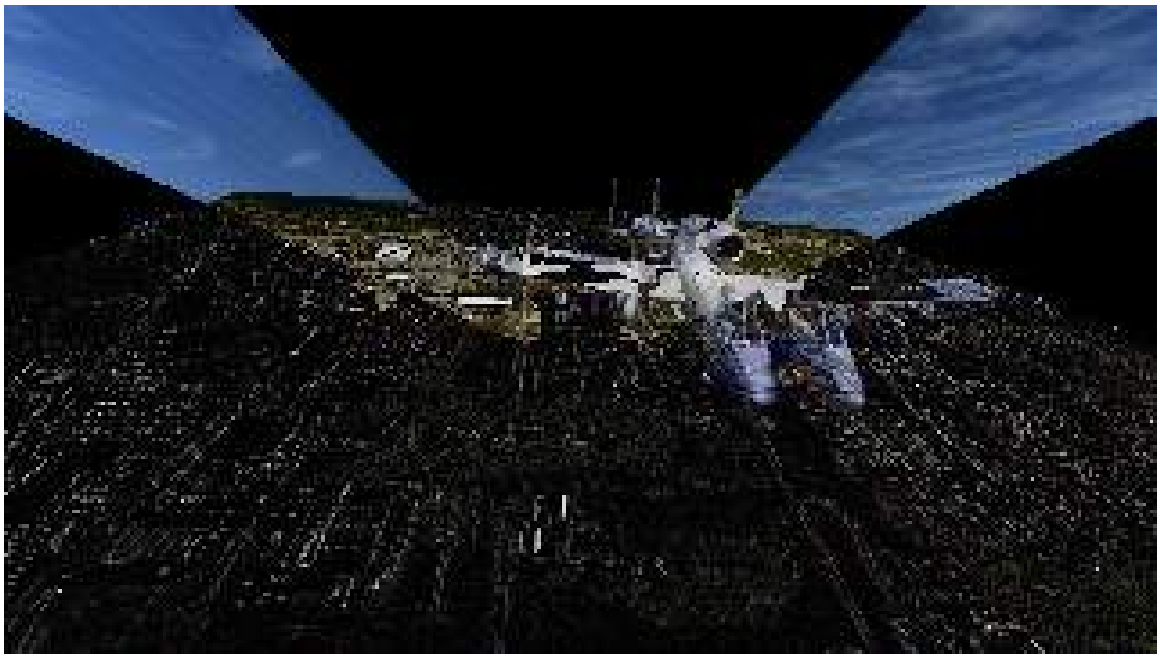
pixel size. Using fewer pixel feature inputs to our CNN allowed us to maintain somewhat smaller networks as depth increased, reduce data storage requirements, and minimize network training time.

In order to ensure our models are generalizable, several different parameters were modified as new datasets were created to ensure the model is robust to noise and visual degradations. Among those simulated were presence of the boom, background noise, and receiver aircraft rotational noise. By simulating these in the virtual environment, we can add robustness to our networks while simulating real-world visual degradations. Our datasets assume a static boom positioned in between the cameras. When added, rotational noise was modeled with a uniform distribution from -5 to 5 degrees in all rotational axes (roll, pitch, and yaw). Background noise is added by including satellite imagery below the receiver aircraft. An example difference image with visual degradations is shown in fig. 8.

Additional assumptions about general aerial refueling were made during dataset creation. We assume that the background is significantly farther from the cameras than the receiver aircraft and boom, so that, given an ideal or fully known camera calibration, the difference captured in the rectified stereoscopic images is minor and only represented by a few pixels. We assume that the boom, background, and cameras are static; these two assumptions were made in the interest of simplicity, and are not representative of a real aerial refueling scenario. Furthermore, we assume that the cameras are positioned in such a manner that a majority of the receiver aircraft is visible by at least one of the cameras. This assumption also includes the case when the left and right cameras are able to view separate regions of the aircraft, but the majority of the aircraft is captured by at least one camera. This assumption only breaks down when a large portion of the aircraft is occluded by the boom in both images. We assume that the cameras exist on the same Y-Z plane as the origin, and

Left Image

Right Image



Difference Image

Figure 8: An example difference image with background noise, boom occlusion, and rotational noise.

that they are facing directly along the X-axis. This is also not necessarily indicative of the real world, and we leave the study of the effects that these factors have on model accuracy to future work.

3.2.2 Architecture Details

We trained four different model architectures, and we recorded the resulting model's root mean squared error (RMSE) on the validation set. By using the valida-

tion set to select the final model, we ensure no cross-contamination of training and test sets: no data from the test set ever factored into model training or selection.

The four models tested involved custom architectures with convolutional layers and fully-connected layers inspired by the work of [31], as that work sought to solve a similar problem as the one we are solving. The first model, *Small*, was a relatively shallow architecture with 6 convolutional layers and 4 fully connected layers before the output. Due to the relatively small size of the network, an image size of 640x360 pixels was used in the training process. This image size caused the final model to have 447,613,795 trainable parameters, mostly in the transition between the convolutional layers and the fully-connected layers. The next model, *Medium*, was a slightly deeper network consisting of 9 convolutional layers and 6 fully connected layers. As a smaller (320x180 pixel) image size was used, this model only has 100,353,923 trainable parameters. Next, we trained and tested a model with 9 convolutional layers and 7 fully connected layers, *Large*. As the fully connected layers of this models are larger, there are 200,763,843 trainable parameters for this model. The final model is a deep model inspired by the concepts used in the YOLO algorithm (introduced by [32]), *YOLO-inspired*. It uses 12 convolutional layers and 4 fully connected layers. However, there are max pooling layers after the 3rd, 9th, and 12th convolutional layer, and an up-sampling layer after the 6th convolutional layer. This allows the network to identify the important features while ignoring much of the noise. This model uses an image size of 320x180 pixels, and only contains 5,281,579 trainable parameters. A diagram of this model's convolutional layers are shown in fig. 9. A full model summary for each architecture is provided in appendix A.

A leaky rectified linear unit (ReLU) activation function was used in all models. This activation function differs from the normal ReLU function in that the leaky ReLU maintains a small negative slope when the neuron is inactive, rather than having a

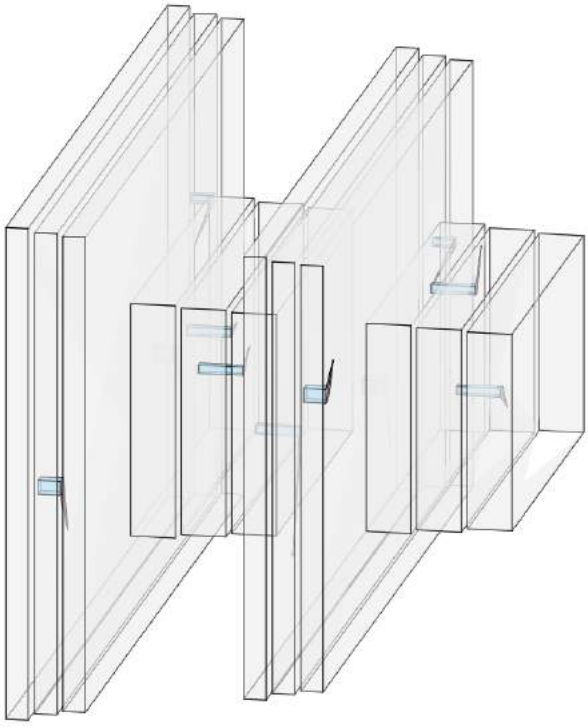


Figure 9: A visual representation of the convolutional layers in the YOLO-inspired architecture.

slope of zero. This prevents neurons from "dying" during training as a result of having no way to reactivate after deactivating within the normal ReLU paradigm.

All models were trained for a maximum of 100 epochs. However, models that did not improve in validation accuracy (provided by the online validation set) for 20 epochs were stopped early. This prevents a model from simply overfitting to the training set rather than generalizing to the data as a whole.

Model training was conducted in an environment where multiple GPU units were available. In general, two Nvidia V100 GPUs were used to train each model. Training time varied based on the model and image size, but generally took around 400 to 600 seconds (7-10 minutes) per epoch, yielding an overall training time of up to 17 hours per model. The model size also varied on architecture, but ranged from 40 MB for

the smallest architecture (*YOLO-inspired*) to 3.33 GB for the largest architecture (shallow network with a larger image size). A smaller network size allows the code to be deployable in an environment where storage and computational resources are limited.

3.3 Results and Analysis

3.3.1 Results

We find that for all datasets, the *YOLO-inspired* architecture is the best performing architecture. A breakdown of the validation RMSE as compared to the dataset and architecture is given in table 4.

Degradations Present in Dataset	**Small**	**Medium**	**Large**	**YOLO**
Background	6.866 cm	5.64 cm	5.10 cm	**3.50 cm**
Background and Rotations	14.30 cm	11.68 cm	9.47 cm	**6.16 cm**
Background, Rotations, and Boom	14.93 cm	10.53 cm	10.21 cm	**8.39 cm**

Table 4: This table shows the validation error given by the various network architectures tested on the different datasets. The best model for each dataset are highlighted in bold. All numbers represent centimeters of RMSE on the offline validation set.

Once the best overall model was selected for each dataset, we used the test set to determine how well the model was able to generalize to completely unseen data. Doing this after the model was chosen ensured that the test set performance did not factor into our model decision at all. The results of the best model on the test set is given in table 5.

A plot of the residuals vs the true X, Y, and Z values for the best model on the full occlusion (background, rotational, and boom noise present) dataset is shown in fig. 10. We see that the model performs better in all three axes when the aircraft is closer to the receiver than when it is farther away. This is expected: as we are using very small images (320x180 pixels), when the object moves away from the cameras two

Dataset	**RMSE (cm)**
Background	3.56
Background and Rotations	6.28
Background, Rotations, and Boom	8.36

Table 5: This table shows the error given by the best network architectures on the test sets. In each case, the model tested was the *YOLO-inspired* architecture, as these architectures had the lowest validation errors.

similar positions will begin to map to the same pixel location and network accuracy will decrease. In addition, as we are utilizing difference images, the disparity between the two images will also decrease as an object moves further away from the camera. Since the disparity between the images is what is being given to the network, a smaller disparity will inherently give less accuracy as the aircraft begins to fade into the background noise.

Overall, we find that a CNN architecture can accurately estimate the position within an average of 10 centimeters when the receiver aircraft is near the contact position (roughly 12-20 meters away from the receiver), and that outlier predictions are still within a meter in each axis. We find that various visual degradations, such as background noise, rotational noise in the receiver aircraft, and the presence of the boom degrade the performance of the architecture, but not significantly enough that the prediction is invalid (especially if the model is made robust to the visual degradation within the simulated dataset).

In the case of the full occlusion dataset (where all visual degradations were present), we see the network accurately predicting the location within a meter along all axes despite all three occlusions being present in the dataset. The average error in prediction is roughly 8.36 cm on the test set, which is close to, but not quite strictly better than, the current industry standard iterative closest point (ICP) performance despite the presence of numerous occlusions and visual degradations.

In addition to accuracy, the architecture must run sufficiently quickly to provide

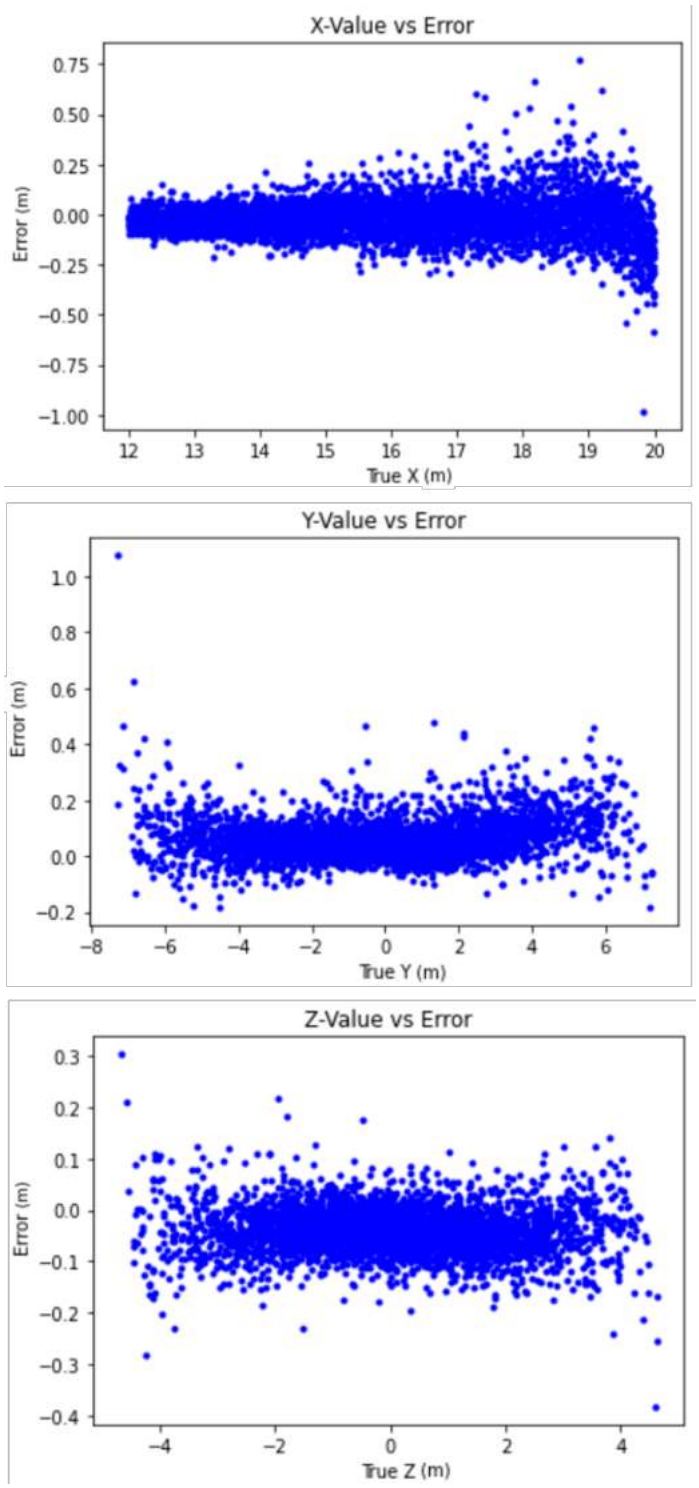


Figure 10: A graph of the errors in prediction of the CNN model as opposed to the true values along all axes.

a near real-time estimate for the position of the aircraft in order to be useful in AAR applications. As the models are simply a series of matrix transformations, once trained they are able to run in near constant time. The full occlusion model takes only 12.397 milliseconds to generate its prediction. The run-time of each model is given in table 6. Furthermore, the only preprocessing that the model requires is to scale the stereoscopic imagery down to an image size of 640x360 pixels and take the absolute difference between the left and right images.

Model	**Runtime (ms)**
Small	13.720
Medium	13.886
Large	12.435
YOLO-inspired	12.397

Table 6: This table shows the runtimes of the different architectures in order to calculate a numerical prediction from an image. All times shown are in milliseconds.

3.3.2 Analysis

Shown in fig. 11 is a graph of the RMSE of the online validation set as the model was training. In this graph, we see that the performance on the training set gradually decreases as the model learns the optimal parameters. We also see the error in the validation set slowly decrease over time, but remain higher than that of the training set. This is expected of a deep learning architecture, as the model can only fit to the instances that it has seen. The fact that the validation loss retains a decreasing trend is indicative that the model has not completely over-fit to the training data at the expense of losing generalizability. Further, the fact that the performance on the test set is near the performance on the validation set (as shown in table 4 and table 5) for each model further proves the fact that the models are able to generalize to new data without a significant loss of accuracy.

Overall, we find that a pure CNN model can yield fairly accurate results in the

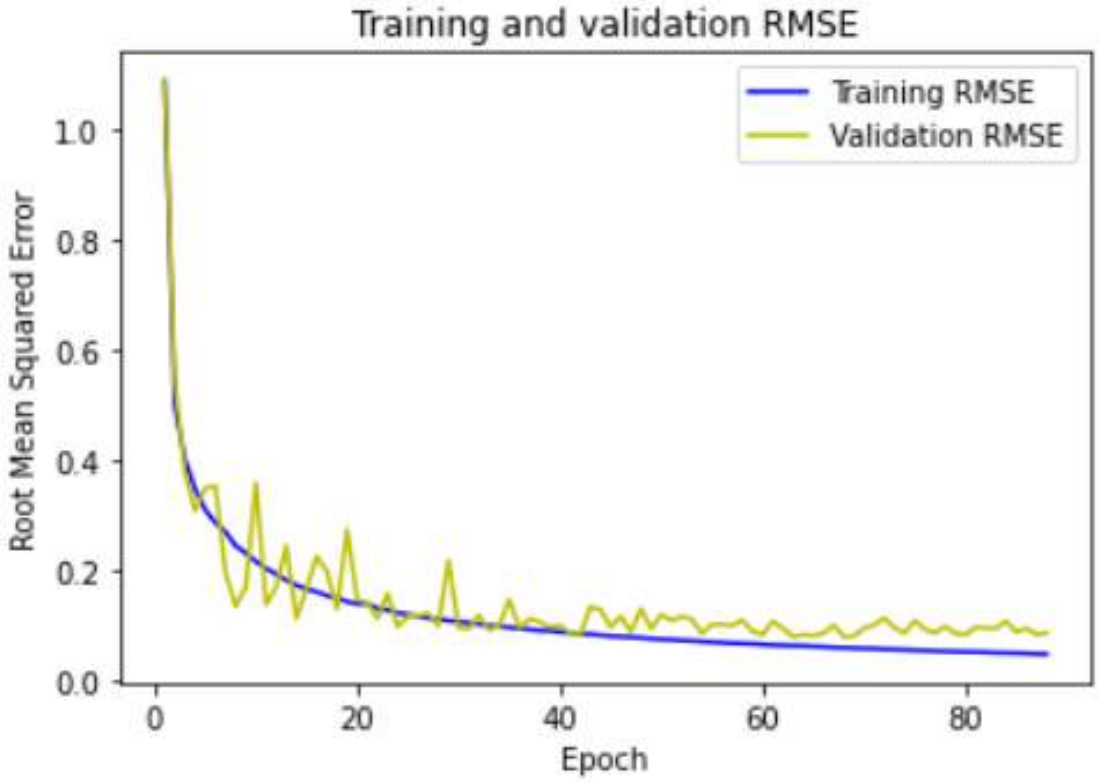


Figure 11: A graph of the training and online validation RMSE (in meters) captured as the model trained.

instance that the receiver aircraft is near the contact position relative to the tanker aircraft (12-20 meters away). This architecture is dependent only on input images in order to make a prediction, and can provide a position estimation in very little time. Further, we find that by modelling a type of visual degradation in the virtual world while creating a dataset, the CNN architecture can be made inherently robust to that degradation (at the cost of some accuracy overall). In chapter IV, we explore the idea of recurrent neural network (RNN) architectures in order to further improve the accuracy of the model, as well as the range at which we can accurately predict the receiver aircraft's position.

IV. Recurrent Neural Networks

The focus of this chapter is to detail all work performed relating to recurrent neural network (RNN) models. The background and related work for RNN models is discussed in section 4.1. We discuss the methodology used in our research in section 4.2. Lastly, section 4.3 discusses the results of our work, and provides an analysis of these results.

4.1 Background and Literature Review

4.1.1 Background

RNNs are a class of networks that excel in handling temporal data. These networks function by passing information back in time via a feedback loop as a state, then combine the previous state with new information when predicting an output. This structure allows the networks to learn complex temporal patterns present in data and make predictions based on these patterns. With the resurgence in deep learning research, the RNN framework has become the prominent deep learning framework with which to analyze sequential data.

Long short-term memory (LSTM) cells are an advancement to the recurrent deep learning field introduced by [33]. These cells function similarly to a standard RNN cell, but rather than simply receiving a state from the previous iteration of the sequence, these cells use two cell state inputs in addition to the current iteration's input. This allows the network to better learn long-term dependencies in the data while still allowing short-term dependencies and current iteration data to influence the output.

In the domain of aerial refueling, there are several key assumptions which compel us to investigate the effectiveness of RNNs. We can reasonably assume that the current position of a receiver aircraft is highly correlated with previous positions,

especially as the frequency of input images increases. Further, we can assume that there are physical laws that constrain the movement of the receiver aircraft that prevent it from moving a significant distance in a relatively short amount of time. By utilizing previous receiver aircraft positions, our architecture is able to estimate higher order equations of motion, and use these equations to more accurately predict the current position of the receiver aircraft. This also helps mitigate various factors that occlude significant portions of the receiver aircraft, such as the boom or the field of view of the cameras.

4.1.2 Related Work

As convolutional neural network (CNN) architectures excel in interpreting spatial data and RNN frameworks excel in handling temporal data, several research efforts exist which utilize both CNNs and RNNs to perform some task on a video sample or series of sequential images. The work of [34] combines the two frameworks in order to label the action sequence that occurs in a video clip. The work of [35] combines the two architectures in an unsupervised attempt to generate intermediate images in a given video sequence.

The work of [36] uses a combination of CNNs, RNNs, and deep Q-learning in order to play Atari games. Of note is that in this research, the models were able to predict future frames even with a lack of current input data, which helped in their analysis of which action to choose.

In their research, [37] shows that both adaptive filters and neural networks have the ability to mitigate occlusions when detecting a moving object. They test both recursive filters and simple neural networks made only of multilayer perceptrons. They conclude that neural networks are the preferred method moving forward due to their adaptibility to noisy data. This work is continued in [38], where they verify

that their methods extend to stereoscopic imagery.

RNNs have also been shown to aid in the mitigation of occlusions. The work of [39] uses an RNN network to track occluded people in sequential images. This is performed by utilizing the known trajectory of the person both before and after the occlusion, and estimating if the two trajectories align. Their work shows the ability to track an object that is fully occluded in some frames, although the focus is mainly on object detection rather than precise position regression.

The research performed by [40] creates a deep learning architecture called an interaction network that is able to learn the physical dynamics of systems, given the knowledge that each observed object in the system interacts in some manner. [41] continues this work by combining CNNs, RNNs, and interaction networks in an architecture called a visual interaction network. This framework extends the previous work in that the architecture can infer the dynamics of the specific systems after three sequential images, and predict the future states of these systems.

The work of [42] seeks to estimate the effects of Newtonian physics in a 3D scene. Their work shows that a CNN-RNN architecture is able to learn long-term dependencies, and can understand a 3D scene well enough to predict the future scene given a force annotated on the image. However, the force and the physics equations are provided to the model rather than inferred directly by the model.

[43] shows that RNN networks are able to sufficiently learn spatio-temporal relations in a graphical context. Finally, [44] performs similar work to ours in that they attempt to predict the future states in a system without first learning the underlying physics of the system. However, their focus relies primarily on long-term estimations rather than utilizing previous information to enhance a single (current) prediction.

4.2 Methodology

As the RNN is a deep learning temporal analysis technique, we extended the range of analysis from a receiver positioned 12-20 meters behind the tanker as used in the previous chapter to a receiver positioned 12-50 meters to better assess the ability for a deep learning model to estimate the position of a receiver aircraft in motion. This distance range also provides a better comparison with iterative closest point (ICP) work. Our deep learning models consisted of CNN layers followed by RNN layers. In this section, we describe our development of CNN-RNN models to estimate the receiver aircraft position.

Our CNN-RNN architectures reuse the CNN architectures from the previous chapter as shown in table 7 to analyze the spatial image data before processing those features through the RNN layers. As the models developed in the chapter III utilized data from the 12-20 meter range, section 4.2.1 describes how we generated CNN layers for our CNN-RNN models.

As an RNN is a temporal analysis tool, the data inputs to an RNN model are typically a temporal data sequence. Section 4.2.2 describes how we generated image sequence data across the 12-50 meter range to train, validate, and test our candidate CNN-RNN architectures.

For the purposes of analyzing the effectiveness of RNNs at estimating receiver position, we created three architecture variants. In the first architecture, *Transfer-train*, we initialized the CNN layers with the best CNN model and allowed the weights of all layers to change throughout the training process. The second architecture, *Transfer-static*, also used the best CNN model, but the weights of the CNN layers were kept fixed throughout the training process. Third, our *Full-train* architecture utilizes CNN layers that were initialized randomly and learned throughout the training process. We used a new YOLO-inspired architecture for the CNN portion of the *Full-train* CNN-

RNN model. We describe these architectures in more detail in section 4.2.3.

To provide a basis of comparison between the CNN models developed in chapter III and the CNN-RNN models developed in this chapter, we also used a 12-20 meter dataset to assess the effectiveness of an RNN at improving receiver aircraft position estimates. We provide this analysis, along with the analysis of the CNN-RNN performance at the extended 12-50 meter range in section 4.3.

4.2.1 CNN Development

In order to accelerate training of CNN-RNN models, we experimented with pre-trained CNN layers. This section describes how we created those layers.

As the CNN models of the previous chapter were created using a dataset of receiver positions 12-20 meters behind the tanker, we created a new dataset for training CNNs containing difference images of a receiver aircraft positioned 12-50 meters in the X-direction (behind the tanker and aft of the camera). As we sought to ensure the aircraft was visible in each shot, this translated to a range of Y (horizontal) offsets from -30 to 30 meters at maximum distance and a range of Z (vertical) offsets from -25 to 25 meters. These distributions are shown in fig. 12. All other parameters for this dataset mirror those of the CNN datasets: 46,656 images split into four main partitions (30,338 for training, 5,353 for online validation, 6,299 for validation after training, and 6,299 for testing generalizability). We used this dataset to pre-train the CNN-layers of in the *Transfer-static* and *Transfer-train* CNN-RNN models. The images used to train the CNNs used in the models trained on the 12-50 meter dataset consisted of both the background noise and boom occlusion. However, for the models trained on the 12-20 meter dataset, we used the CNN models trained in the previous chapter using just background noise (no boom occlusion).

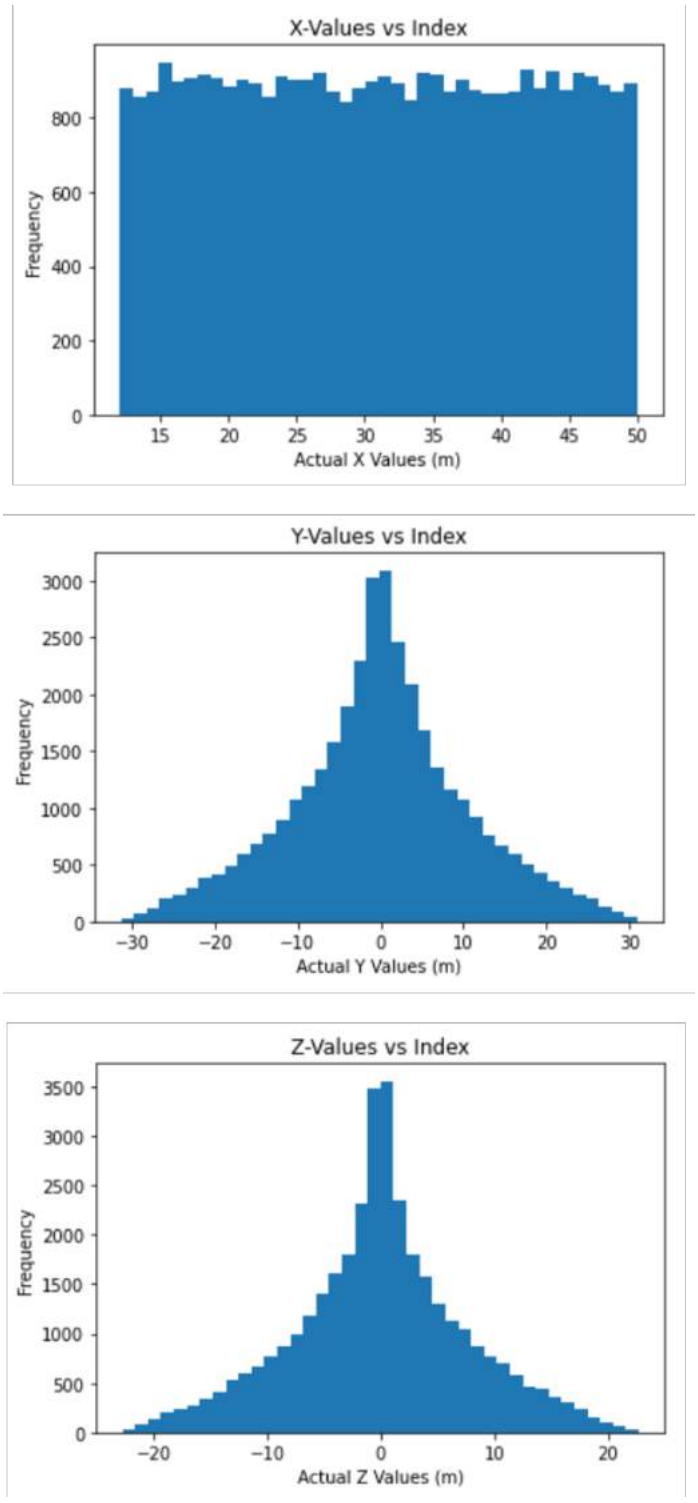


Figure 12: The data distributions in each axis for the training set of the 12-50 meter CNN dataset.

4.2.2 Data Generation

For the CNN-RNN dataset, we created temporally sequenced data with a series of 4 difference images for each observation. The objective of the CNN-RNN model is to predict the X, Y, and Z position of the aircraft in the fourth image using nothing other than the sequence of 4 difference images. The sequence of images begins with the receiver aircraft in an initial position. The receiver aircraft is then initialized with an initial velocity and acceleration. This velocity and acceleration is applied to the aircraft's initial position to generate a sequence of 4 images. As it is unlikely that a receiver aircraft would accelerate horizontally in a real scenario, acceleration in the Y-axis was not simulated. An example sequence is shown in fig. 13.

To ensure that the final position of the aircraft used for prediction remained in-frame, we randomly generated observations by first selecting an aircraft position $P_\omega = (x_\omega, y_\omega, z_\omega)$ that ensured that the majority of the aircraft (at least parts of both wings and the body) were in view. Then, to determine an initial aircraft position $P_0 = (x_0, y_0, z_0)$, we moved the receiver aircraft backwards 13 meters from P_ω so that $P_0 = (x_\omega + 13, y_\omega, z_\omega) = (x_0, y_0, z_0)$. This adjustment ensured that the final position of the aircraft after several seconds would remain in frame in cases of initial positions near the frame edges, or cases in which the random parameters created significant movement over the image sequence.

The initial velocity vector v_0 was a randomly selected vector in the range $([-0.25, -1.0], [0.025, 0.1], [0.0625, 0.25])$ meters per second. The aircraft acceleration a_0 remained constant throughout the 4 image sequence. We applied zero acceleration along the Y axis, but the acceleration vector along the other axes was selected from a normal distribution $\theta = (\mu, \sigma)$ having a mean of μ and standard deviation of σ. The distribution of initial acceleration vectors along the X axis was $\theta_x = (0.0, 0.4)$ and along the Z axis was $\theta_z = (0.0, 0.1)$.

Each observation for the CNN-RNN model was a four difference-image sequence, where the first difference image was captured at the receiver aircraft's initial position P_0. The second and subsequent positions were calculated using the equation $P_t = a_0t^2 + v_0t + P_0$, where $t = \{1, 2, 3\}$. Difference images were captured at each value of t, and the series of four images was stored as an observation in the 12-50 meter dataset.

We repeated this process to generate the 12-20 meter dataset, with a few changes. First, we constrained X values of P_ω to a range of $[12, 15]$. We then selected a P_0 by moving the aircraft backwards only 3 meters instead of 13, so that $P_0 = (x_\omega+3, y_\omega, z_\omega)$. Finally, we changed the frame rate to 4 images per second instead of 1 image per second ($t = \{0.25, 0.5, 0.75\}$). These changes were necessary to ensure that the final position of the aircraft remained in view. The changes had a consequent effect that the difference in initial position and final position of an aircraft was greater in the 12-50 meter dataset than in the 12-20 meter dataset, and the distance travelled between frames was greater in the 12-50 meter dataset than in the 12-20 meter dataset.

The distribution of final positions of receiver aircraft in the 12-50 meter dataset is provided in fig. 14, and the distribution of final position in the 12-20 meter dataset is in fig. 15. In both datasets, the distribution of final positions was approximately normal in all axes. In the 12-50 meter dataset, the X-values ranged from 12 to 45 meters and the Y- and Z-values were in the range $[-8, 8]$ meters. In the 12-20 meter dataset, the X-values ranged from 12-20 meters, the Y-values ranged from -4 to 4 meters, and the Z-values ranged from -2.5 to 2.5 meters. Note that while the initial velocity was always positive, the initial acceleration could be negative, resulting in some aircraft actually moving backwards relative to the tanker throughout the image sequence.

In order to sufficiently train the CNN-RNN model, even more data had to be

generated than was created for the pure CNN models. For our models, the datasets used consists of 15,625 instances, each consisting of 4 difference-image sequences. We divided these into a training set (11,952 instances), validation set (2,110), and a test set (1,563). As each instance contained four images, the training set had 47,808 images (more than were in the entirety of any CNN dataset), the validation set contained 8,440 images, and the test sets contained 6,252 images, for a combined 62,500 images total. Both the 12-50 meter dataset and the 12-20 meter dataset had 62,500 images.

The only degradations we introduced in the RNN datasets were the background noise and the boom; we did not simulate rotational noise for RNN training. All underlying assumptions of the CNN datasets are also present in the RNN datasets. In addition, we assume that the movement of the receiver aircraft is simple enough to be modeled by a second order polynomial in each axis; this assumption is not necessarily indicative of the real world; however, for short periods of time, it accurately reflects aircraft movement.

4.2.3 Architecture Details

As discussed, we designed three CNN-RNN models. Each model consists of a series of CNN layers followed by a series of RNN layers. The input to the CNN layers is the sequence of 4 difference-images. The output of the CNN layers is a prediction of the receiver's X, Y, and Z position of in each frame. The RNN layers consist of three independent networks which each attempt to predict the aircraft position along one of the coordinate axes. In other words, the CNN's predictions for the X values are fed into one RNN, the Y values are fed into another, and the Z values are fed into the third. The results of the three independent sub-network RNNs are then concatenated to provide a single position prediction for the receiver position in the fourth frame.

The CNN layers mirror the CNN architecture discussed in chapter III. For each

RNN sub-network, we use a single layer LSTM with 2,048 hidden units. A visualization of our network architecture is depicted in fig. 16.

We chose this architecture because of the successful results of [36]. The independent RNN sub-networks allowed each RNN to learn an independent movement along each axis. This allows the overall CNN-RNN models to accurately predict the receiver location given a variety of complex movement patterns made by the receiver aircraft.

We incorporated this basic CNN-RNN architecture in each of the models, *Transfer-train*, *Transfer-static*, *Full-train*. The CNN layers used in the *Transfer-train* and *Transfer-static* were selected from the best performing models as evaluated on the validation dataset. We used a *YOLO-inspired* CNN in the *Full-train* model. We generated each of these three models on both the 12-20 meter and the 12-50 meter datasets.

After selecting the best CNN model, each CNN-RNN model was trained for 100 epochs. To prevent wasted training time, we established early stopping criteria if the training loss did not decrease by a pre-determined amount for 20 epochs.

In the following section, we discuss the results and analyze the models we created.

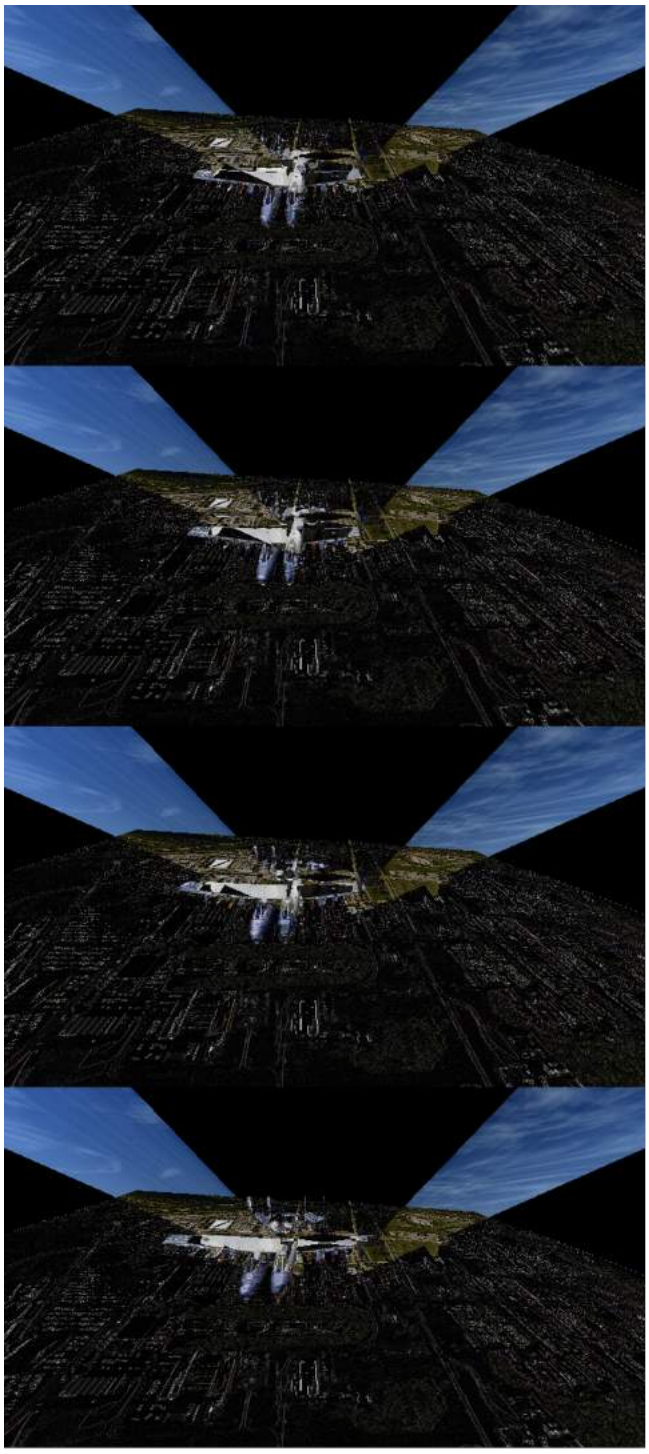


Figure 13: An example of an image sequence used during training. Note that the only position annotated is the position of the receiver aircraft in the final image.

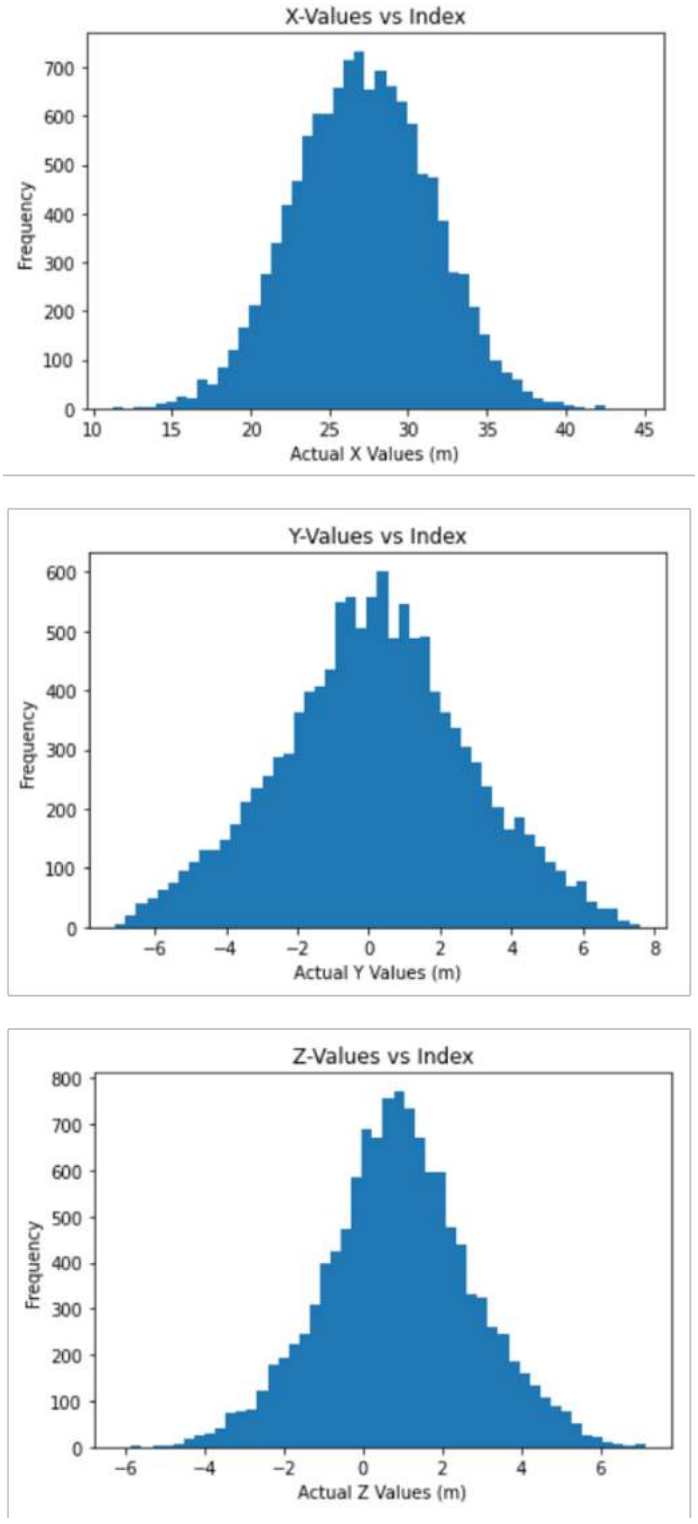


Figure 14: The data distributions in each axis for the training set of the 12-50 meter CNN-RNN dataset.

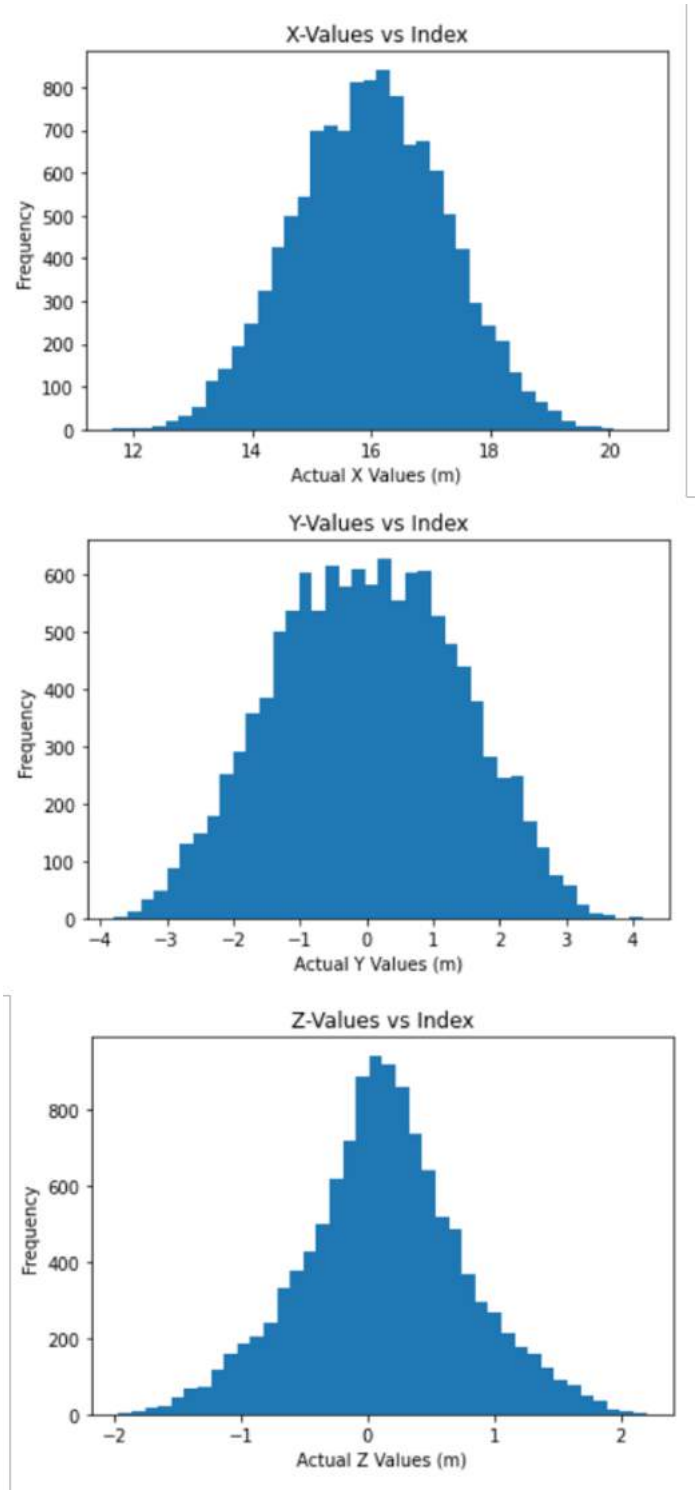


Figure 15: The data distributions in each axis for the training set of the 12-20 meter CNN-RNN dataset.

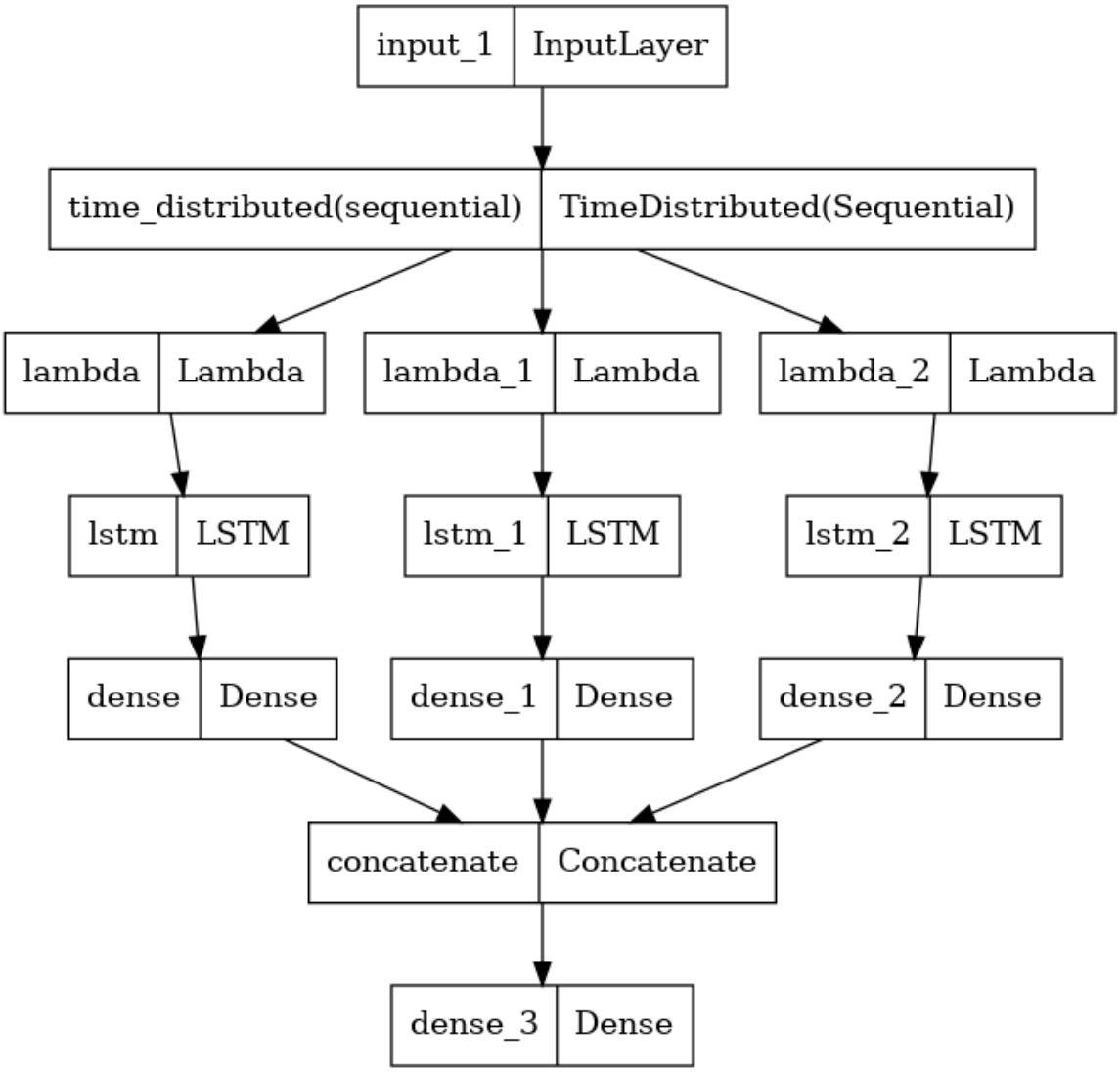


Figure 16: A visual depiction of the final CNN-RNN model architecture used to predict the location of the receiver aircraft. The CNN layers are encapsulated in the TimeDistributed layers.

4.3 Results and Analysis

4.3.1 Results

We found that the best performing CNN architecture on the 12-50 meter dataset is the medium-sized architecture. The model achieved a test root mean squared error (RMSE) of 32.54 centimeters. Effectively, this means that the CNN layers could, on average, accurately predict the aircraft's position within 33 centimeters of its position throughout the 12-50 meter range. The performance of each model architecture tested on the 12-50 meter dataset are shown in table 7. Based on these results, we chose the *Medium* CNN to train the CNN-RNN model on the 12-50 meter dataset.

The best performing CNN architecture on the 12-20 meter dataset, the *YOLO-inspired* architecture, was reused from the previous chapter, and it had a test RMSE 3.56 centimeters. See table 5 for the performance of each of the CNN models evaluated on the 12-20 meter dataset.

The models trained on the 12-50 meter dataset using a *Medium* CNN architecture had 150,740,882 parameters. In the *Transfer-train* model, all of these are trainable. In the *Transfer-static* model, only the weights leading into the RNN layer are trainable, reducing the number of trainable parameters to 50,386,959. In the *Full-train* model, we used another YOLO-inspired CNN architecture. This architecture is similar to the *YOLO-inspired* architecture used to train CNN models, but an increased stride in the deeper convolutional layers vastly decreased the number of parameters within the CNN portion. Overall, the *Full-train* CNN-RNN architecture contains 50,435,898 parameters, all of which are initialized with random values and are trainable.

The models trained on the 12-20 meter dataset using the *YOLO-inspired* CNN architecture had 55,668,538 parameters in the case of *Transfer-train* and *Transfer-static*. The *Full-train* architecture contains 50,435,898 parameters. In the *Transfer-train* and *Full-train* models, all of the parameters are trainable, but they are initialized

to trained values in the former model and random values in the latter model. In *Transfer-static*, only 50,386,959 of these parameters are trainable.

Model	**RMSE (cm)**
Small	37.41
Medium	**34.15**
Large	36.40
YOLO-inspired	34.31

Table 7: This table shows the validation error given by the various CNN architectures tested on the 12-50 meter distance dataset. The best performing model and RMSE is highlighted in bold.

Turning now to the performance of the complete CNN-RNN model trained on the 12-50 meter dataset, we report the error of each trained model when applied to the validation dataset in table 8. We note that the *Transfer-train* model out-performs the the other models with a validation error of 6.19 centimeters. When we applied this model to the test dataset, it accurately predicted the receiver position within 6.69 centimeters, on average, over the 12-50 meter range.

We report the performance of the trained CNN-RNN models on the 12-20 meter datasets in table 9. As in the models trained on the 12-50 meter dataset, the *Transfer-train* model out performs the other models, accurately predicting the aircraft's position an average of 1.61 centimeters of its true position over the 12-20 meter range when applied to the validation dataset. The performance on the test dataset demonstrates this model has found a general solution, with an average error of 1.43 centimeters. See table 10 for a summary of test performance of the best model trained on each dataset.

We find that in all cases, the CNN-RNN model improved upon the results yielded by the pure CNN model. Overall, the best performing model architecture on both datasets was the transfer-learning architecture where all parameters were trainable. A direct comparison of CNN and CNN-RNN models is shown in table 10.

Model	RMSE (cm)
Transfer-train	**6.19**
Transfer-static	24.78
Full-train	11.17

Table 8: This table shows the validation error given by the various CNN-RNN architectures tested on the 12-50 meter CNN-RNN dataset. The best performing model and RMSE is highlighted in bold.

Model	RMSE (cm)
Transfer-train	**1.61**
Transfer-static	55.33
Full-train	3.17

Table 9: This table shows the validation error given by the various CNN-RNN architectures tested on the 12-20 meter distance dataset. The best performing model and RMSE is highlighted in bold.

4.3.2 Analysis

Comparing the results of the CNN layers, the performance of the models trained on the 12-50 meter dataset is much worse than the models trained on the 12-20 meter dataset. We provide a residual error analysis of the best 12-50 meter model *Medium* in fig. 17. Refer to fig. 10 for a residual error analysis of the 12-20 meter models. In the 12-50 meter dataset, we see that the accuracy along the X-axis degrades significantly with distance from the tanker aircraft, with errors in excess of 2 meters at a distance of 50 meters. Errors along the Y- and Z-axes are also greater near the bounds of the range than at the center. However, this model performs comparably with the 12-20 meter CNN models near contact location. The residuals of this area specifically are shown in fig. 18.

Recall that in our data generation process, only aircraft that were near the maximum distance aft of the tanker could be near the extremes of the Y- and Z-axis ranges. Thus, we suspect that the underlying position that results in large error predictions along the X-axis is also generating large errors along the Y- and Z-axes as

Dataset	CNN model	CNN-RNN model
12-50 meters	32.54 cm	6.69 cm
12-20 meters	3.56 cm	1.43 cm

Table 10: This table shows the comparison between the best-performing CNN models and the best performing CNN-RNN models on the two distance distributions. All numbers reported are centimeters of RMSE on the test set.

well. We suspect there are three causes for these errors near the extreme aft of the range of data. First, the farther the receiver aircraft is from the cameras, the smaller the features are that are being detected. Second, as an object moves farther from the camera, each pixel represents a larger area of the real-world aircraft. Third, the relative size of an image of an aircraft at 50 meters and at the extreme edges of the Y- and Z-axes (near the corners of the image) will have the same size and shape as an aircraft that is much closer to the tanker and nearer to the X-axis. Consequently, a CNN might not effectively distinguish the position of two aircraft that have a similar size and shape in the image.

RNNs significantly improve the performance of the models, as demonstrated in the residual analysis shown in fig. 19. In these graphs, we report the predicted position error of the best CNN-RNN model (*Transfer-train* with *Medium* CNN layers) trained on the 12-50 meter dataset. These graphs demonstrate that errors mainly fall within 1 meter in all axes, with some outliers. The CNN-RNN model provides a significantly improved position estimate than the CNN model without the temporal analysis provided by the RNN. We also see that the CNN-RNN model has a fairly consistent error along the entire range for all axes, as opposed to functioning significantly worse at larger distances. This is due to the smoothing nature of the LSTM cells: if the receiver aircraft begins moving away from the tanker, the RNN can use the trajectory information from the previous locations to better predict where the receiver aircraft will be. We still see a subtle trend of increasing error at the edges

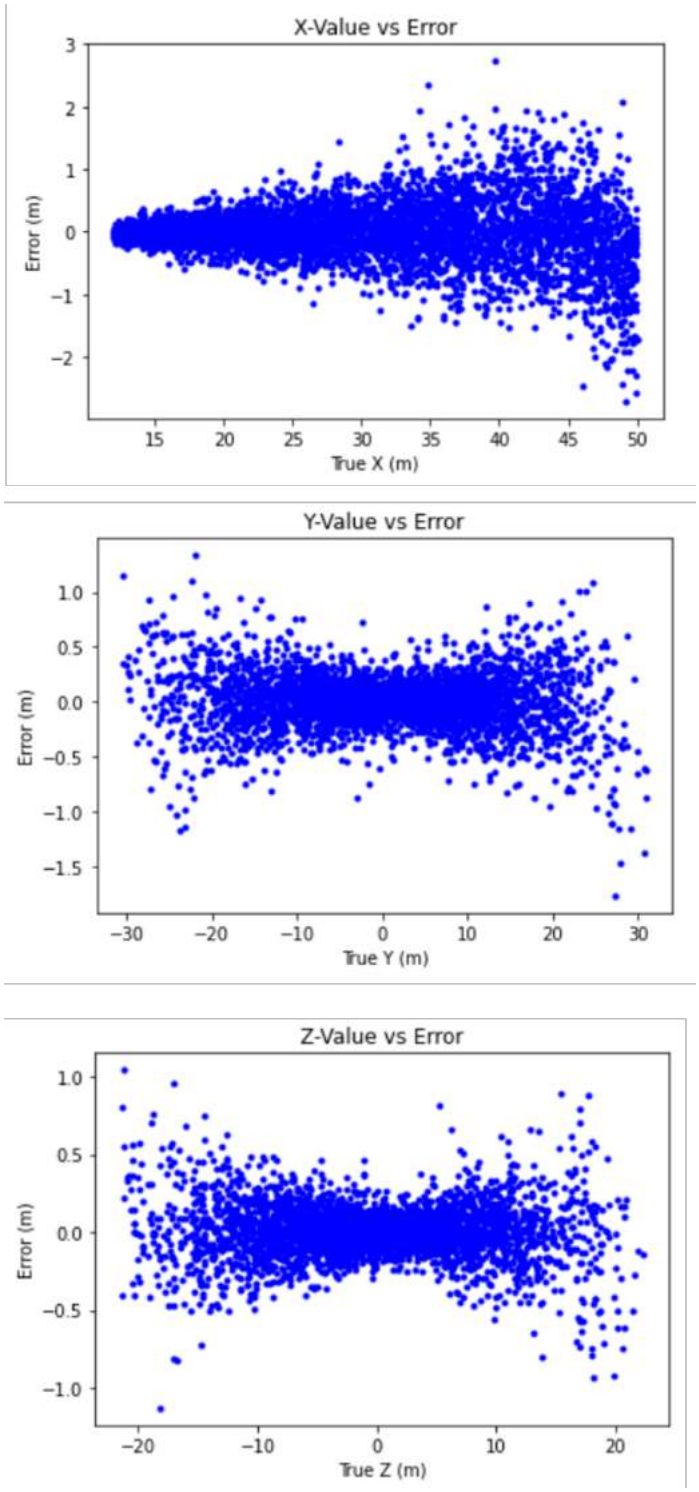


Figure 17: A graph of the errors in prediction of the CNN model on the 12-50 meter CNN test set as opposed to the true values along all axes. All values are in meters.

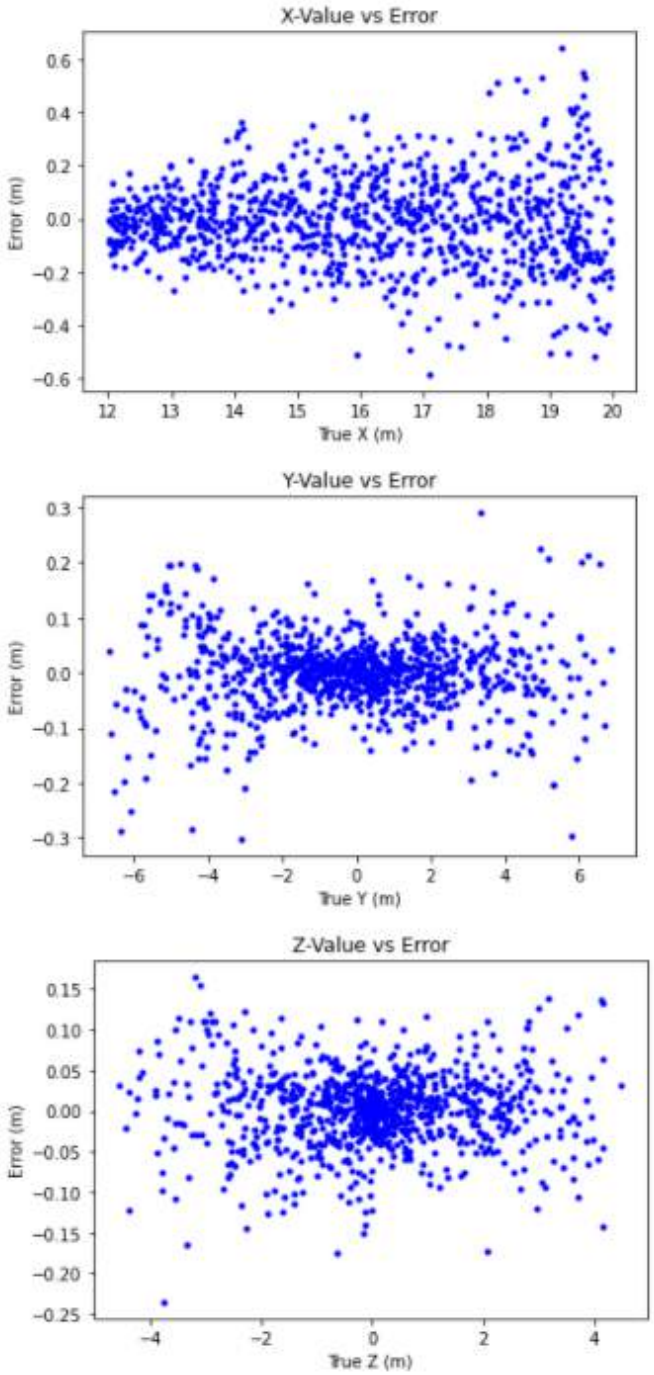


Figure 18: A graph of the errors in prediction of the CNN model on the 12-50 meter CNN test set at the 12-20 meter range. This shows us how well the 12-50 meter CNN model performs near contact location. All values are in meters.

of the full range of values in each axis, but this trend is very slight and noticeably smaller than the errors presented by the CNN models. This indicates that the errors

caused by certain aircraft positions yielding similar image sizes but vastly different X, Y, and Z positions can be overcome with a sequence of images processed through a CNN-RNN model. Further analysis shows that the model is most accurate along the Z-axis and least accurate along the X-axis. We also see in the CNN-RNN model that the axes become less codependent: an incorrect prediction along one axis is less correlated with incorrect predictions along the other axes.

Turning to the 12-20 meter dataset, a graph of the errors of the best performing CNN-RNN architecture (*Transfer-train* with *YOLO-inspired* CNN layers) on the 12-20 meter dataset is shown in fig. 20. We find that a CNN-RNN architecture can provide accurate position estimates in all axes near contact location, with most predictions accurate within 10 centimeters of their respective axis. We again find that the Z-axis is the best axis in terms of accuracy, but also find that the X- and Y-axes are roughly equal in their relative accuracies. Finally, we see a very subtle trend of increased inaccuracy near the extremes of all axes.

Of note is the performance of the *Transfer-static* model on the 12-20 meter dataset. This model performs significantly worse than any other model on either dataset (see table 9 which shows an RMSE error of over 55 centimeters, an order of magnitude greater than the position error of the other models). As the CNN model used in the transfer training was not trained with any boom occlusion, and the CNN layers were frozen throughout training, the mitigation of this occlusion needed to come strictly from training the LSTM sub-networks. This leads us to conclude that the LSTM sub-networks alone are not sufficient to handle significant visual degradation (such as the boom) independent of the CNN sub-network. However, once the CNN portion of the model was allowed to adjust to the visual degradation (as in *Transfer-train*), we saw a significant increase in accuracy.

We also observe that the best CNN-RNN network, *Transfer-train*, outperformed

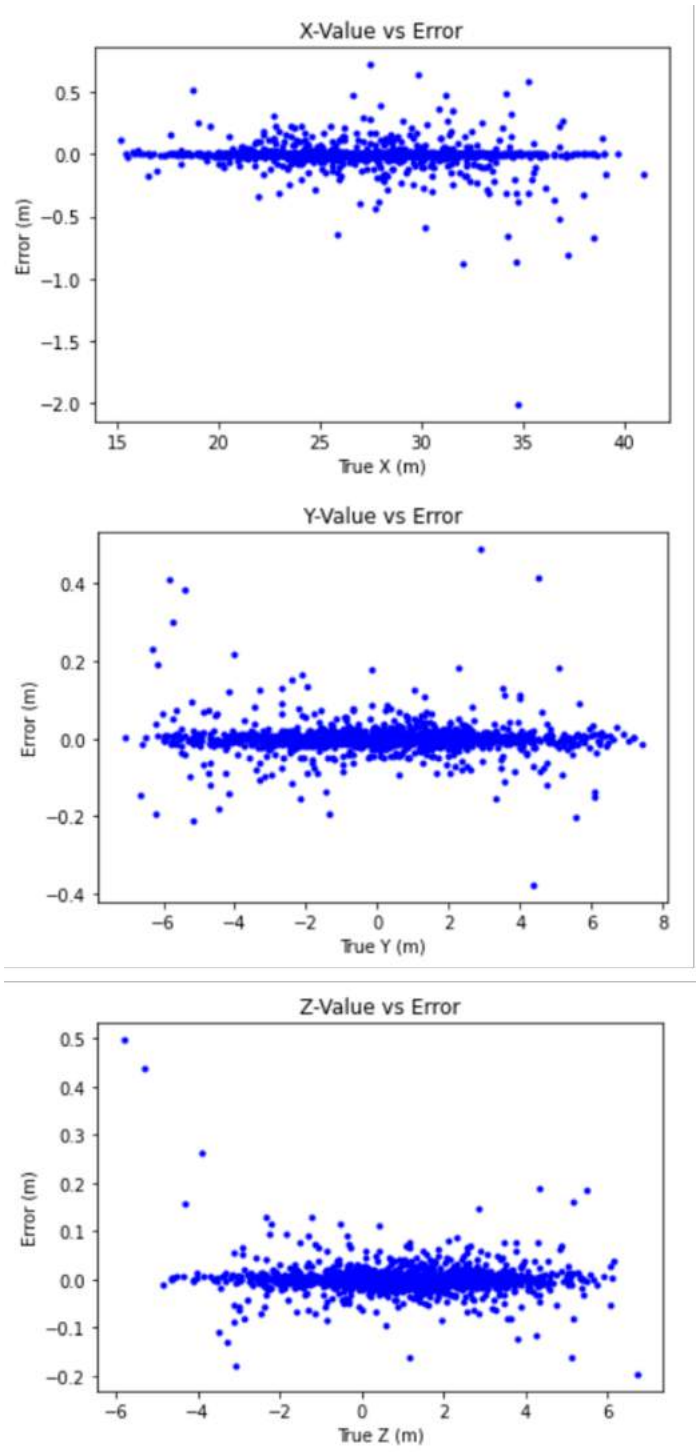


Figure 19: A graph of the errors in prediction of the CNN-RNN model on the 12-50 meter CNN-RNN dataset as opposed to the true values along all axes. All values are in meters.

Full-train. Moreover, *Transfer-train* met the early termination criteria at 93 epochs, which means that it had not reduced the error by a predetermined amount over 5 epochs. In contrast, the *Full-train* terminated after 75 epochs. This indicates a success for transfer learning. By transferring the learned weights, *Transfer-train* was able to find a more optimal model than *Full-train* could with random initialization.

The timing for the CNN-RNN models, however, is significantly slower than the pure CNN models. The CNN-RNN networks took over 300 milliseconds to make a prediction whereas the CNN required only 12 milliseconds. While this is a 2,500% slowdown, one factor to note is that the model was required to perform an estimation for every image it received. In a deployed model, the architecture would simply remember the previous estimates as opposed to recalculating them every time a new image was received. Additionally, all four images have to be read into memory while making a prediction, then deleted afterwards, whereas the pure CNN model only performs this operation once. While the architecture is still larger and therefore slower than a pure CNN model, the 2,500% slowdown is the upper limit on the amount of time this architecture would take to make a prediction. Overall, we'd expect the actual performance in a deployed model to be approximately a quarter of our performance, or around 75 milliseconds.

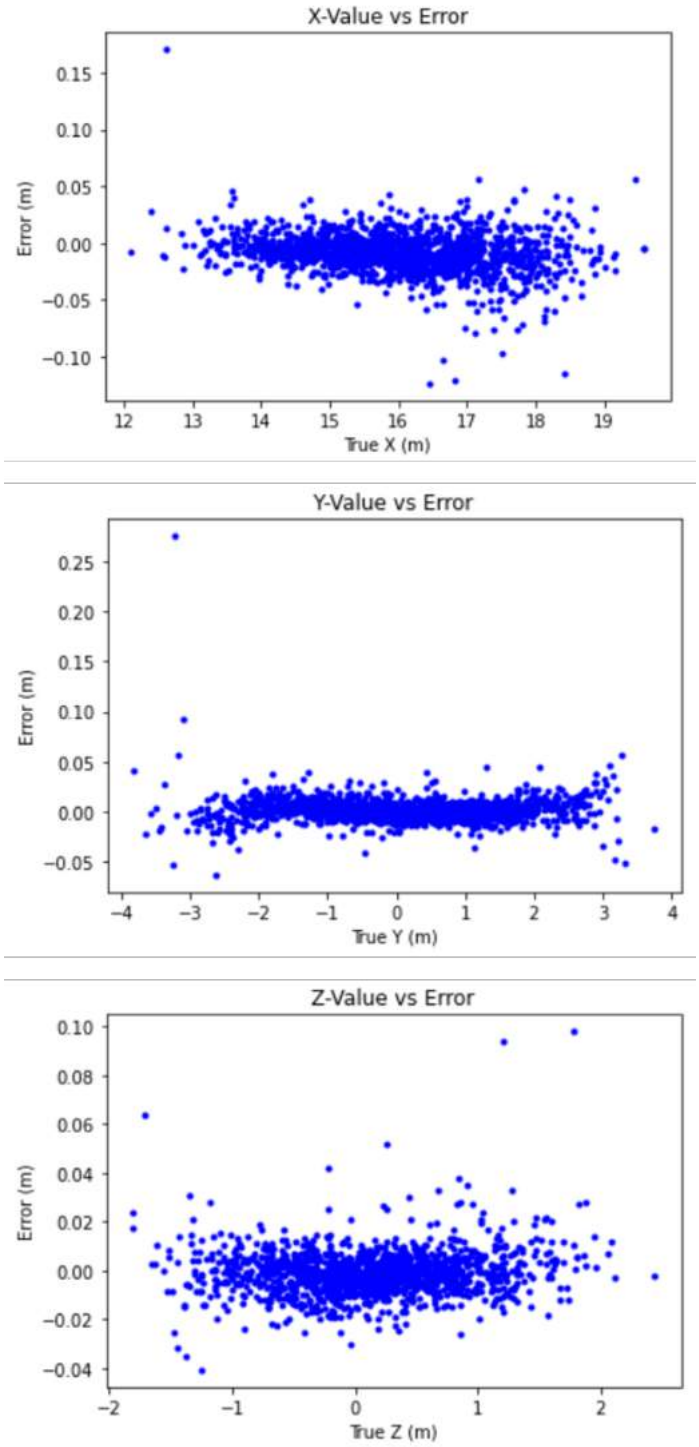


Figure 20: A graph of the errors in prediction of the CNN-RNN model on the 12-20 meter dataset as opposed to the true values along all axes. All values are in meters.

V. Conclusions

In this thesis, we investigated the use of both machine learning and deep learning models for an accurate and timely 3D position estimation. We find that while statistical machine learning models provide some promise, they ultimately are insufficient at estimating 3D position to the accuracy needed to perform automated aerial refueling (AAR), and the more powerful deep learning models are needed.

We then analyze the effectiveness of convolutional neural network (CNN) architectures at estimating a 3D position utilizing only a difference image between stereoscopic cameras. We seek to utilize these models in order to reduce the need for the pre-processing steps required in current AAR efforts, as deep learning methods have been shown to be robust to noisy data. We find these results to be satisfactory at close range, while also running in very low amounts of time and with very little required pre-processing.

Finally, we investigate the usage of recurrent neural network (RNN) networks to aid in the mitigation of occlusions and to extend the accuracy of CNN models over a wider range. We find that an RNN architecture can vastly improve the accuracy of the prediction, but this comes at the expense of prediction speed. This research lays the foundation for a new avenue in AAR efforts that takes only images as input, and utilizes the intrinsic properties of neural networks to provide a robust solution to the pose estimation problem in the face of object occlusions.

5.1 Future Work

As we have demonstrated the feasibility of deep learning architecture to accurately estimate receiver aircraft position with only input images, and with visual occlusions, there are several avenues which would build upon this work and ultimately prove

useful to AAR as a whole. These are briefly discussed here.

Early research sought to estimate orientation (roll, pitch, and yaw) in addition to position for the full 6D pose estimation. However, we did not pursue orientation prediction analysis, as the orientation of a receiver aircraft in the refueling envelope tends to remain close to straight and level (0 degrees of offset in each rotational axis). Further, orientation estimation significantly increased the scope of data needed to train a successful model, and caused a decrease in the accuracy of the position estimation. Future work may be performed to use deep learning architectures to estimate orientation, but it is our recommendation to keep a separate orientation network rather than directly regressing position and orientation within the same network.

Further research could be done to test the applicability of a single network to predict multiple aircraft types, such as an F-15 or a C-5, or future work could involve a classifier that identifies the receiver aircraft and intelligently selects the appropriate pose prediction network. As this research assumed a completely ideal camera structure, future work could involve testing the robustness of the architectures to minor noise in camera orientations or different intrinsic camera calibrations.

We generated all of our data in a virtual world. Further research is required to validate the applicability of networks trained on simulated imagery into real-world scenarios. If the virtual worlds are found to be highly transferable to real-world scenarios, it allows extensive work to be performed with essentially unlimited amounts of simulated data. These networks can then be deployed to real-world scenarios where large-scale data collection is prohibitive in some manner.

We suspect that the nature of the small image size is advantageous in generalizing virtually generated data into real-world scenarios. As our images are scaled down to only 320x180 pixel images, much of the finer details are lost. Similarly, should a real-

world image be scaled down significantly in size, much of the finer details would be lost as well. We hypothesize that there is more similarity between low-resolution images of virtually generated scenes and real-world scenes than high-resolution images of the same scenes. However, further testing would be necessary to verify this as fact in order to confirm that training models in a virtual environment for use in a real-world environment is a viable strategy.

Another avenue of research to be explored is that of different aircraft configurations. We suspect that a deep learning architecture like those we propose will be inherently robust to different aircraft configurations, as these models learn by finding a generalized solution to the task they are given. Since the key features of an aircraft stay consistent across separate configurations, the models will simply learn to utilize the consistent features of each aircraft to calculate a position estimate, and the accuracy will not considerably decrease.

In addition to the visual degradation tested in this research, further research could be performed in order to ensure the architectures are robust to an array of adverse visibility conditions. Apart from those we tested, it is common for aerial refueling to occur despite glint, glare, fog, and low-light conditions, among other visual impediments. Future research could be performed by simulating these conditions in the virtual world to ensure the models are made robust to them.

The final step in AAR is to automate the control of the boom. The models we propose could substantially aid in this process by providing accurate position estimations to the pipeline. Nevertheless, substantial future work is required in order to achieve fine control of the boom, as this step is crucial in ensuring neither aircraft is damaged during the refueling process.

Ultimately, we find the use of neural networks to directly regress 3D position is largely unexplored, but may provide an alternative solution that mitigates many of

the issues plaguing modern AAR methods. Further, this work is applicable in other fields which also seek to estimate 3D position from stereoscopic imagery.

Appendix A. CNN Model Architectures

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 640, 360, 8)       608

 leaky_re_lu (LeakyReLU)     (None, 640, 360, 8)       0

 conv2d_1 (Conv2D)           (None, 636, 356, 8)       1608

 leaky_re_lu_1 (LeakyReLU)   (None, 636, 356, 8)       0

 conv2d_2 (Conv2D)           (None, 634, 354, 8)       584

 leaky_re_lu_2 (LeakyReLU)   (None, 634, 354, 8)       0

 conv2d_3 (Conv2D)           (None, 632, 352, 16)      1168

 leaky_re_lu_3 (LeakyReLU)   (None, 632, 352, 16)      0

 conv2d_4 (Conv2D)           (None, 630, 350, 16)      2320

 leaky_re_lu_4 (LeakyReLU)   (None, 630, 350, 16)      0

 conv2d_5 (Conv2D)           (None, 628, 348, 16)      2320

 leaky_re_lu_5 (LeakyReLU)   (None, 628, 348, 16)      0

 flatten (Flatten)           (None, 3496704)           0

 dense (Dense)               (None, 128)               447578240

 leaky_re_lu_6 (LeakyReLU)   (None, 128)               0

 dense_1 (Dense)             (None, 128)               16512

 leaky_re_lu_7 (LeakyReLU)   (None, 128)               0

 dense_2 (Dense)             (None, 64)                8256

 leaky_re_lu_8 (LeakyReLU)   (None, 64)                0

 dense_3 (Dense)             (None, 32)                2080

 leaky_re_lu_9 (LeakyReLU)   (None, 32)                0

 dense_4 (Dense)             (None, 3)                 99

 activation (Activation)     (None, 3)                 0

=================================================================
Total params: 447,613,795
Trainable params: 447,613,795
Non-trainable params: 0
_________________________________________________________________
```

Figure 21: The shallow *Small* architecture used in CNN research.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 320, 180, 8)       608
leaky_re_lu (LeakyReLU)      (None, 320, 180, 8)       0
conv2d_1 (Conv2D)            (None, 316, 176, 8)       1608
leaky_re_lu_1 (LeakyReLU)    (None, 316, 176, 8)       0
conv2d_2 (Conv2D)            (None, 314, 174, 8)       584
leaky_re_lu_2 (LeakyReLU)    (None, 314, 174, 8)       0
conv2d_3 (Conv2D)            (None, 312, 172, 16)      1168
leaky_re_lu_3 (LeakyReLU)    (None, 312, 172, 16)      0
conv2d_4 (Conv2D)            (None, 310, 170, 16)      2320
leaky_re_lu_4 (LeakyReLU)    (None, 310, 170, 16)      0
conv2d_5 (Conv2D)            (None, 308, 168, 16)      2320
leaky_re_lu_5 (LeakyReLU)    (None, 308, 168, 16)      0
conv2d_6 (Conv2D)            (None, 306, 166, 32)      4640
leaky_re_lu_6 (LeakyReLU)    (None, 306, 166, 32)      0
conv2d_7 (Conv2D)            (None, 304, 164, 32)      9248
leaky_re_lu_7 (LeakyReLU)    (None, 304, 164, 32)      0
conv2d_8 (Conv2D)            (None, 302, 162, 32)      9248
leaky_re_lu_8 (LeakyReLU)    (None, 302, 162, 32)      0
flatten (Flatten)            (None, 1565568)           0
```

Figure 22: The convolutional layers of the *Medium* architecture used in CNN research.

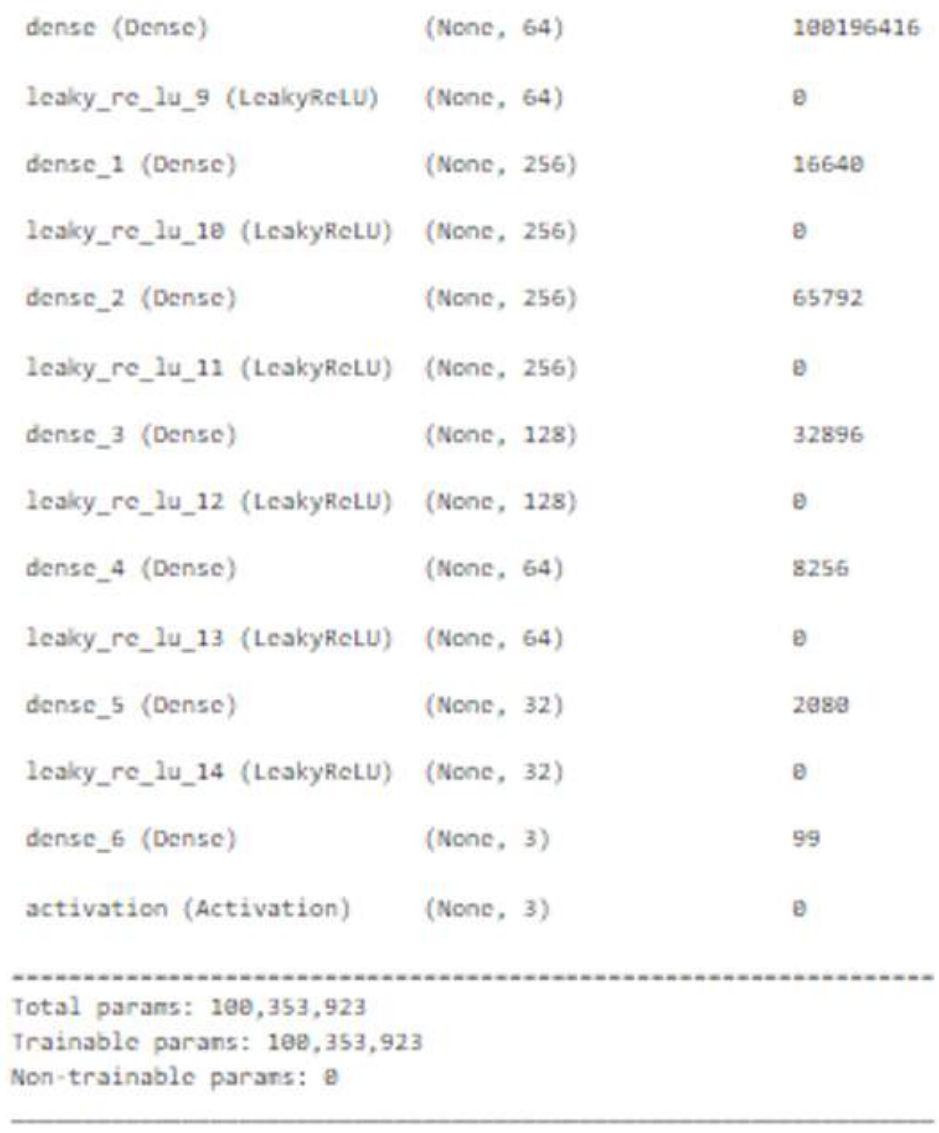
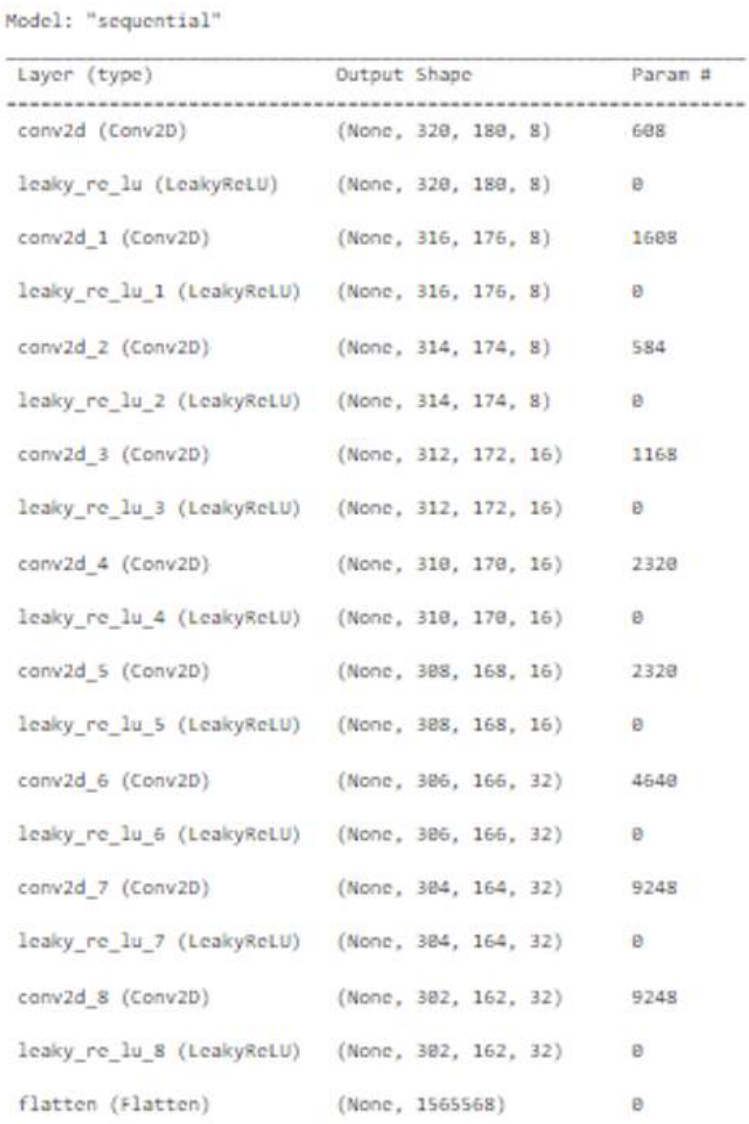


Figure 23: The dense layers of the *Medium* architecture used in CNN research.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 320, 180, 8)       608
leaky_re_lu (LeakyReLU)      (None, 320, 180, 8)       0
conv2d_1 (Conv2D)            (None, 316, 176, 8)       1608
leaky_re_lu_1 (LeakyReLU)    (None, 316, 176, 8)       0
conv2d_2 (Conv2D)            (None, 314, 174, 8)       584
leaky_re_lu_2 (LeakyReLU)    (None, 314, 174, 8)       0
conv2d_3 (Conv2D)            (None, 312, 172, 16)      1168
leaky_re_lu_3 (LeakyReLU)    (None, 312, 172, 16)      0
conv2d_4 (Conv2D)            (None, 310, 170, 16)      2320
leaky_re_lu_4 (LeakyReLU)    (None, 310, 170, 16)      0
conv2d_5 (Conv2D)            (None, 308, 168, 16)      2320
leaky_re_lu_5 (LeakyReLU)    (None, 308, 168, 16)      0
conv2d_6 (Conv2D)            (None, 306, 166, 32)      4640
leaky_re_lu_6 (LeakyReLU)    (None, 306, 166, 32)      0
conv2d_7 (Conv2D)            (None, 304, 164, 32)      9248
leaky_re_lu_7 (LeakyReLU)    (None, 304, 164, 32)      0
conv2d_8 (Conv2D)            (None, 302, 162, 32)      9248
leaky_re_lu_8 (LeakyReLU)    (None, 302, 162, 32)      0
flatten (Flatten)            (None, 1565568)           0
```

Figure 24: The convolutional layers of the *Large* architecture used in CNN research.

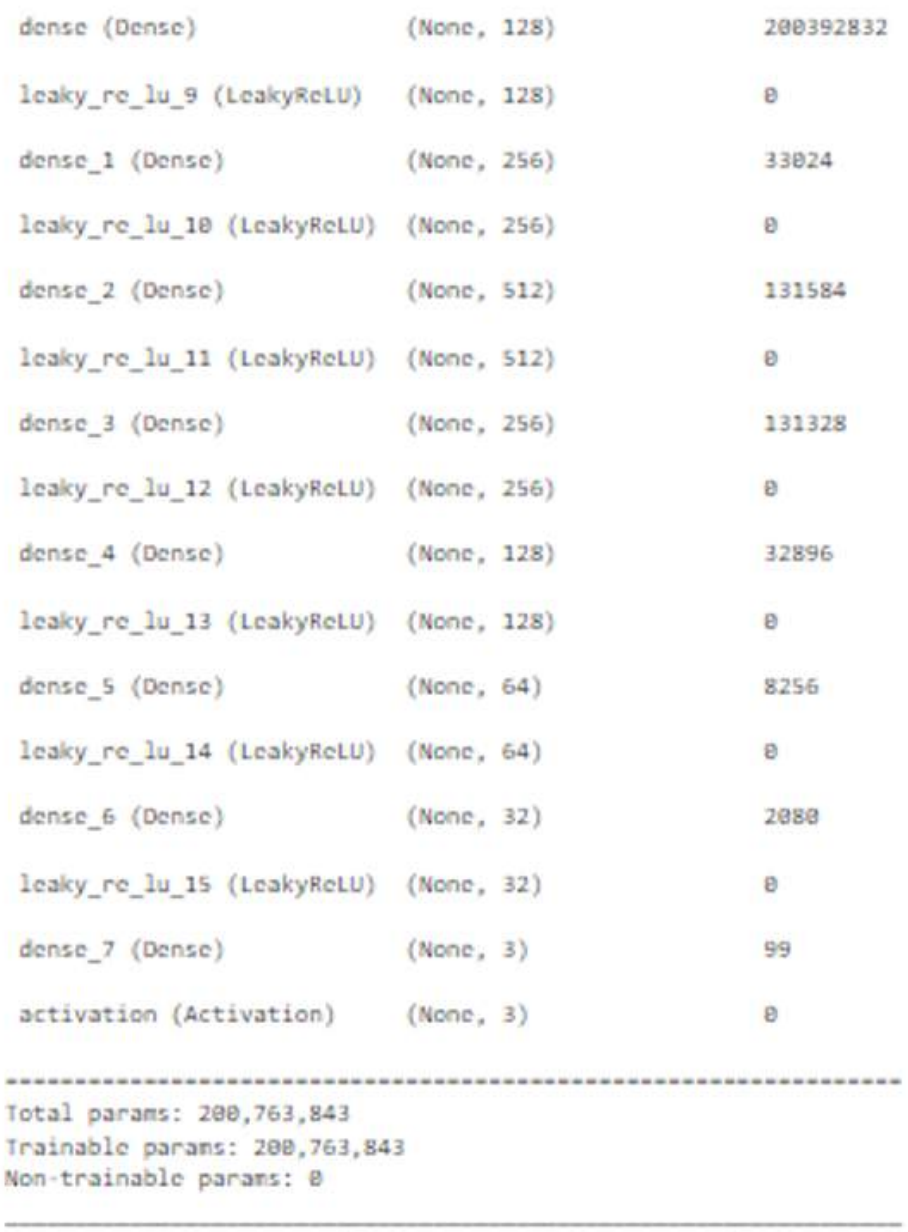
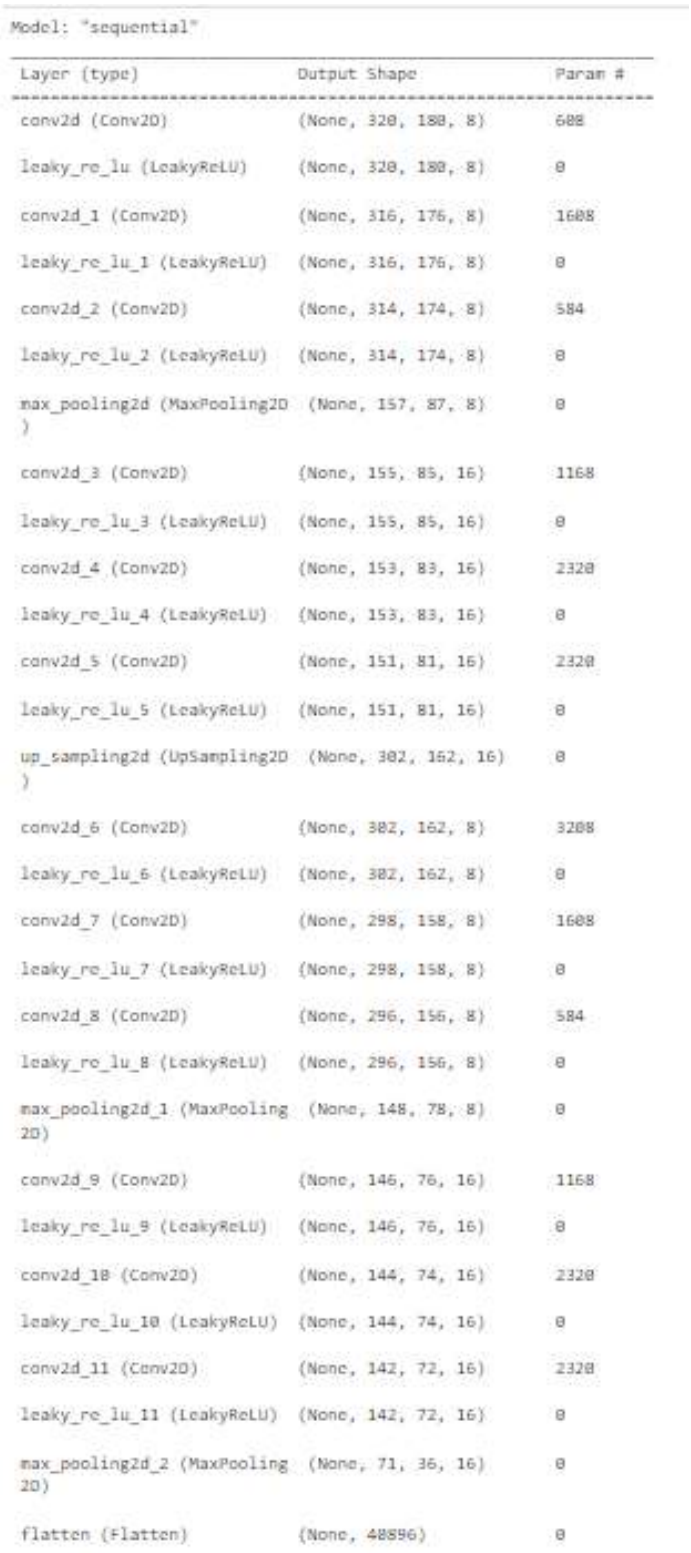


Figure 25: The dense layers of the *Large* architecture used in CNN research.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 320, 180, 8)       608

leaky_re_lu (LeakyReLU)      (None, 320, 180, 8)       0

conv2d_1 (Conv2D)            (None, 316, 176, 8)       1608

leaky_re_lu_1 (LeakyReLU)    (None, 316, 176, 8)       0

conv2d_2 (Conv2D)            (None, 314, 174, 8)       584

leaky_re_lu_2 (LeakyReLU)    (None, 314, 174, 8)       0

max_pooling2d (MaxPooling2D  (None, 157, 87, 8)        0
)

conv2d_3 (Conv2D)            (None, 155, 85, 16)       1168

leaky_re_lu_3 (LeakyReLU)    (None, 155, 85, 16)       0

conv2d_4 (Conv2D)            (None, 153, 83, 16)       2320

leaky_re_lu_4 (LeakyReLU)    (None, 153, 83, 16)       0

conv2d_5 (Conv2D)            (None, 151, 81, 16)       2320

leaky_re_lu_5 (LeakyReLU)    (None, 151, 81, 16)       0

up_sampling2d (UpSampling2D  (None, 302, 162, 16)      0
)

conv2d_6 (Conv2D)            (None, 302, 162, 8)       3208

leaky_re_lu_6 (LeakyReLU)    (None, 302, 162, 8)       0

conv2d_7 (Conv2D)            (None, 298, 158, 8)       1608

leaky_re_lu_7 (LeakyReLU)    (None, 298, 158, 8)       0

conv2d_8 (Conv2D)            (None, 296, 156, 8)       584

leaky_re_lu_8 (LeakyReLU)    (None, 296, 156, 8)       0

max_pooling2d_1 (MaxPooling  (None, 148, 78, 8)        0
2D)

conv2d_9 (Conv2D)            (None, 146, 76, 16)       1168

leaky_re_lu_9 (LeakyReLU)    (None, 146, 76, 16)       0

conv2d_10 (Conv2D)           (None, 144, 74, 16)       2320

leaky_re_lu_10 (LeakyReLU)   (None, 144, 74, 16)       0

conv2d_11 (Conv2D)           (None, 142, 72, 16)       2320

leaky_re_lu_11 (LeakyReLU)   (None, 142, 72, 16)       0

max_pooling2d_2 (MaxPooling  (None, 71, 36, 16)        0
2D)

flatten (Flatten)            (None, 40896)             0
```

Figure 26: The convolutional layers of the *YOLO-inspired* architecture used in CNN research.

```
dense (Dense)                (None, 128)               5234816

leaky_re_lu_12 (LeakyReLU)   (None, 128)               0

dense_1 (Dense)              (None, 128)               16512

leaky_re_lu_13 (LeakyReLU)   (None, 128)               0

dense_2 (Dense)              (None, 64)                8256

leaky_re_lu_14 (LeakyReLU)   (None, 64)                0

dense_3 (Dense)              (None, 32)                2080

leaky_re_lu_15 (LeakyReLU)   (None, 32)                0

dense_4 (Dense)              (None, 3)                 99

activation (Activation)      (None, 3)                 0

=================================================================
Total params: 5,281,579
Trainable params: 5,281,579
Non-trainable params: 0
_________________________________________________________________
```

Figure 27: The dense layers of the *YOLO-inspired* architecture used in CNN research.

Bibliography

1. Satya Mallick. Histogram of oriented gradients explained using opencv, Nov 2021.

2. Daniel T. Johnson, Scott L. Nykl, and John F. Raquet. Combining stereo vision and inertial navigation for automated aerial refueling. *Journal of Guidance, Control, and Dynamics*, 40(9):2250–2259, 2017.

3. Matt Piekenbrock, Jace Robinson, Lee Burchett, Scott Nykl, Brian Woolley, and Andrew Terzuoli. Automated aerial refueling: Parallelized 3D iterative closest point: Subject area: Guidance and control. *Proceedings of the IEEE National Aerospace Electronics Conference, NAECON*, 0(August):188–192, 2016.

4. Christopher Parsons, Zachary Paulson, Scott Nykl, William Dallman, Brian G. Woolley, and John Pecarina. Analysis of simulated imagery for real-time vision-based automated aerial refueling. *Journal of Aerospace Information Systems*, 16(3):77–93, 2019.

5. Zachary Paulson, Scott Nykl, John Pecarina, and Brian Woolley. Mitigating the effects of boom occlusion on automated aerial refueling through shadow volumes. *Journal of Defense Modeling and Simulation*, 16(2):175–189, 2019.

6. Andrew Lee, Will Dallmann, Scott Nykl, Clark Taylor, and Brett Borghetti. Long-range pose estimation for aerial refueling approaches using deep neural networks. *Journal of Aerospace Information Systems*, 17(11):634–646, 2020.

7. Ryan Raettig. Accelerating Point Set Registration for Automated Aerial Refueling. 2021.

8. Joshua Larson. Stereo Camera Calibrations With Optical Flow. 2021.

9. M. L. Fravolini, G. Campa, and M. R. Napolitano. Evaluation of machine vision algorithms for autonomous aerial refueling for unmanned aerial vehicles. *Journal of Aerospace Computing, Information and Communication*, 4(9):968–985, 2007.

10. Thattapon Surasak, Ito Takahiro, Cheng Hsuan Cheng, Chi En Wang, and Pao You Sheng. Histogram of oriented gradients for human detection in video. *Proceedings of 2018 5th International Conference on Business and Industrial Research: Smart Technology for Next Generation of Information, Engineering, Business and Social Science, ICBIR 2018*, pages 172–176, 2018.

11. Reza Ebrahimzadeh and Mahdi Jampour. Efficient Handwritten Digit Recognition based on Histogram of Oriented Gradients and SVM. *International Journal of Computer Applications*, 104(9):10–13, 2014.

12. Paul E. Rybski, Daniel Huber, Daniel D. Morris, and Regis Hoffman. Visual classification of coarse vehicle orientation using histogram of oriented gradients features. *IEEE Intelligent Vehicles Symposium, Proceedings*, (December):921–928, 2010.

13. Norbert Buch, James Orwell, and Sergio A. Velastin. 3D extended histogram of oriented gradients (3DHOG) for classification of road users in urban scenes. *British Machine Vision Conference, BMVC 2009 - Proceedings*, 2009.

14. Tiesheng Wang and Irene Y H Gu. Object tracking using incremental 2D-PCA learning and ML estimation. *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, (348):933–936, 2007.

15. E. Munoz, Y. Konishi, V. Murino, and A. Del Bue. Fast 6D pose estimation for texture-less objects from a single RGB image. *Proceedings - IEEE International Conference on Robotics and Automation*, 2016-June:5623–5630, 2016.

16. Christopher Parsons and Scott Nykl. Real-time automated aerial refueling using stereo vision. In *International Symposium on Visual Computing*, pages 605–615. Springer, 2016.

17. Scott Nykl, Chad Mourning, Mitchell Leitch, David Chelberg, Teresa Franklin, and Chang Liu. An overview of the STEAMiE Educational game engine. *Proceedings - Frontiers in Education Conference, FIE*, pages 21–25, 2008.

18. Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Ha. Gradient-Based Learning Applied to Document Recognition Yann. *Proceedings of the IEEE*, (November):1–46, 1998.

19. Cong Zhang, Xiaobin Xu, Yuhui Shi, Yimin Deng, Cong Li, and Haibin Duan. Binocular Pose Estimation for UAV Autonomous Aerial Refueling via Brain Storm Optimization. *2019 IEEE Congress on Evolutionary Computation, CEC 2019 - Proceedings*, pages 254–261, 2019.

20. Wanqing Zhao, Shaobo Zhang, Ziyu Guan, Wei Zhao, Jinye Peng, and Jianping Fan. Learning Deep Network for Detecting 3D Object Keypoints and 6D Poses. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 14122–14130, 2020.

21. Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. DPOD: 6D pose object detector and refiner. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob:1941–1950, 2019.

22. Alexander Krull, Eric Brachmann, Frank Michel, Michael Ying Yang, Stefan Gumhold, and Carsten Rother. Learning analysis-by-synthesis for 6d pose estimation in RGB-D images. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:954–962, 2015.

23. Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. *arXiv*, 2017.

24. Yannick Bukschat and Marcus Vetter. EfficientPose: An efficient, accurate and scalable end-to-end 6D multi object pose estimation approach. 2020.

25. Mingxing Tan, Ruoming Pang, and Quoc V. Le. EfficientDet: Scalable and efficient object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 10778–10787, 2020.

26. Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-Time Seamless Single Shot 6D Object Pose Prediction. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 292–301, 2018.

27. Zekun Luo, Xia Wu, Qingquan Zou, and Xiao Xiao. Object detection based on binocular vision with convolutional neural network. *ACM International Conference Proceeding Series*, pages 60–65, 2018.

28. Daniel Scharstein, Heiko Hirschmüller, York Kitajima, Greg Krathwohl, Nera Nešić, X. Wang, and Porter Westling. High-resolution stereo datasets with subpixel-accurate ground truth. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8753(1):31–42, 2014.

29. Jure Žbontar and Yann Lecun. Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17:1–32, 2016.

30. Nuno R. Goncalves and Andrew E. Welchman. "What Not" Detectors Help the Brain See in Depth. *Current Biology*, 27(10):1403–1412.e8, 2017.

31. Andrew Lee. Object Detection with Deep Learning to Accelerate Pose Estimation for Automated Aerial Refueling. *Theses and Dissertations*, 2020.

32. Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December:779–788, 2016.

33. Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

34. Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):677–691, 2017.

35. MarcAurelio Ranzato, Arthur Szlam, Joan Bruna, Michael Mathieu, Ronan Collobert, and Sumit Chopra. Video (language) modeling: a baseline for generative models of natural videos. pages 1–15, 2014.

36. Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in Atari games. *Advances in Neural Information Processing Systems*, 2015-Janua:2863–2871, 2015.

37. Eduardo Parrilla, Damián Ginestar, José Luis Hueso, Jaime Riera, and Juan Ramón Torregrosa. Handling occlusion in optical flow algorithms for object tracking. *Computers and Mathematics with Applications*, 56(3):733–742, 2008.

38. Eduardo Parrilla, Jaime Riera, Juan R. Torregrosa, and José L. Hueso. Handling occlusion in object tracking in stereoscopic video sequences. *Mathematical and Computer Modelling*, 50(5-6):823–830, 2009.

39. Maryam Babaee, Zimu Li, and Gerhard Rigoll. Occlusion Handling in Tracking Multiple People Using RNN. *Proceedings - International Conference on Image Processing, ICIP*, pages 2715–2719, 2018.

40. Peter W. Battaglia, De Haas MJG, and Moolenaar SW. Interaction Networks for Learning about Objects, Relations and Physics. 2003.

41. Nicholas Watters, Andrea Tacchetti, Theophane Weber, Razvan Pascanu, Peter Battaglia, and Daniel Zoran. Visual Interaction Networks. pages 1–14, 2017.

42. Roozbeh Mottaghi, Mohammad Rastegari, Abhinav Gupta, and Ali Farhadi. "What happens if..." learning to predict the effect of forces in images. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9908 LNCS:269–285, 2016.

43. Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Jain structural rnn deep learning cvpr 2016 paper. *Cvpr*, pages 5308–5317, 2016.

44. Sebastien Ehrhardt, Aron Monszpart, Niloy J. Mitra, and Andrea Vedaldi. Learning A Physical Long-term Predictor. 2017.

Acronyms

AAR automated aerial refueling. iv, 1, 2, 3, 5, 6, 7, 24, 37, 63, 64, 65, 66, 1

CNN convolutional neural network. viii, ix, x, xi, xii, 2, 3, 4, 7, 23, 24, 25, 26, 27, 28, 29, 30, 35, 36, 37, 38, 40, 41, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 67, 68, 69, 70, 71, 72, 73

HOG histogram of oriented gradients. viii, 9, 10, 11, 12, 15, 16, 19

ICP iterative closest point. 2, 5, 6, 7, 8, 10, 35, 42

LSTM long short-term memory. 39, 48, 56, 59

PCA principal component analysis. 9, 12, 13, 15, 16, 18, 19, 20

PnP Perspective-n-Point. 25, 26

RANSAC random sample consensus. 25

RBF radial basis function. 12, 15

ReLU rectified linear unit. 32, 33

RMSE root mean squared error. viii, xi, xii, 11, 13, 15, 20, 31, 34, 35, 37, 38, 53, 54, 55, 56, 59

RNN recurrent neural network. ix, xi, xii, 2, 3, 4, 38, 39, 40, 41, 42, 43, 45, 46, 47, 48, 50, 51, 52, 53, 54, 55, 56, 59, 60, 61, 62, 63

SVR support vector regressor. xi, 12, 15, 16, 19, 20

UAV unmanned aerial vehicle. 1, 25

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24–03–2022	Master's Thesis	Sept 2020 — Mar 2022

4. TITLE AND SUBTITLE

Deep Learning Techniques To Estimate 3D Position In Stereoscopic Imagery

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Nicholson, Jonathan I., 2d Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-22-M-050

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Current AAR efforts utilize machine vision algorithms to estimate the pose of a receiver aircraft. However, these algorithms are dependent on several conditions such as the availability of precise 3D aircraft models; the accuracy of the pipeline significantly degrades in the absence of high-quality information given beforehand. We propose a deep learning architecture that estimates the 3D position of an object based on stereoscopic imagery. We investigate the use of both machine learning techniques and neural networks to directly regress the 3D position of the receiver aircraft. We present a new position estimation framework that is based on the differences between two stereoscopic images without relying on the stereo block matching algorithm. We analyze the speed and accuracy of its predictions and demonstrate the effectiveness of the architecture in mitigating various visual occlusions.

15. SUBJECT TERMS

Aerial Refueling, Machine Learning, Deep Learning, Stereoscopic Imagery

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Col W. Blair Watkinson II, AFRL/RYZ
U	U	U	UU	94	19b. TELEPHONE NUMBER *(include area code)* (937) 713-8020; Warren.Watkinson@us.af.mil