Experimental Evaluation of Object Detection Algorithms for UAV Tracking

by

Bingsheng Wei

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

University of Alberta

Abstract

Object detection is an image processing technology to detection different classes of objects using computer vision, i.e. putting bounding boxes over objects from a camera video feed. A landmark detection method was the Viola-Jones Algorithm introduced in 2001. The object classifier in this algorithm is called the Cascade Classifier, and was used to detect human faces in real-time. However, the Cascade Classifier is poor at detecting objects which have different features or poses than on those it was trained, or when the object is presented with a complex background. In order to achieve a more robust and accurate detection result than Cascade Classifier, deep learning techniques are used as an object detection solution for real-time aerial robotics detection. Two state-of-the-art deep learning frameworks, Darknet and TensorFlow, were implemented in Robot Operating System (ROS) and tested in experiment. Computational costs and reliability among various classifiers in the two deep learning frameworks were tested and compared. The experiments were conducted on a commercially available Parrot AR.Drone 2.0 Unmanned Aerial Vehicle (UAV) flying over various backgrounds. Experimental results show that a real-time, accurate and consistent UAV detection can be achieved by using deep learning techniques, and the results can be extended to the more challenging case of UAV tracking.

Preface

This thesis is an original work by Bingsheng Wei. No part of this thesis has been previously published.

Acknowledgements

It has been a pleasant experience to stay at Dr. Martin Barczyk's mechatronics lab for two years.

I wish to thank Dr. Barczyk for his supervision and support. It was a great help to this thesis.

Adam Casey helped to optimize the communication between Parrot ARDrone 2.0 and lab computers.

Pranoy Panda helped to train Darknet APIs and optimized Re3 for tracking Parrot ARDrone 2.0.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

Human being has the ability to identify objects in the three-dimensional (3D) world. The human brain can effortlessly differentiate between various kinds of an object, e.g. different species of cats or dogs. In order to mimic human perception of objects, the study of computer vision can be traced back to 1966 when an undergraduate student at Massachusetts Institute of Technology was asked to get computer to know what it saw from a camera [1]. Nowadays, computer vision has applications such as simultaneous localization and mapping, surveillance, object detection, etc.

One significant achievement in object detection was the Cascade Classifier introduced by Paul Viola and Michael Jones in 2001 to detect human faces [2]. However, the Cascade Classifier could not identify the same face unless detection was conducted in the same fashion as training. For instance, detection would fail if a face presented to the classifier was rotated or presented over a complex background.

Inspired by the human brain's structure, the deep learning neural network was applied to object detection [3]. One big achievement of object detection using neural networks was the AlexNet winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 [4]. Deep learning object detection systems work by extracting features from images perceived by a computer, then use algorithms which train a model, which is then used to detect the object in future images. Deep learning detection is different from Cascade

Classifier detection. Not only can it detect objects, it can also distinguish between different types of the same object. For instance, it can determine the size and species of a dog. Deep learning training and detection require huge amounts of computational power. A Graphics Processing Unit (GPU) solves this issue by enabling computing on thousands of CUDA cores in parallel, reducing learning time and enabling object detection on real-time video streams.

For the purposes of UAV tracking and following, the quality of UAV detection is crucial as the onboard control system needs information in the form of bounding boxes to determine the location of a target drone. In Pablo Martinez Rodriguez's work [5], the Cascade Classifier is used for detecting a target. While this algorithm does the job of detecting drones, it is extremely unstable in realistic conditions. The system can only detect the target UAV over a pure white background and in the absence of noise or disturbances.

This thesis proposes the implementation of deep learning neural networks to perform UAV detection and tracking. This research will be a foundation of a UAV tracking project, serving as the computer vision input to a control system performing following to a target UAV. The TensorFlow and Darknet machine learning frameworks were employed with convolutional networks SSD MobileNet v1, SSD Inception v2, Faster RCNN Inception v2, YOLO v2 and Tiny YOLO to perform UAV detection and demonstrate improved performance over the Cascade Classifier, while being efficient enough to provide real-time detection performance.

1.2 Thesis Research Gap, Challenge and Objective

The research gap of this thesis is that the cascade classifier can barely detect a target UAV over a complex background. This is a limitation on the detection consistency. No reliable detection system was achieved for the UAV detection project in the lab.

The challenge of this thesis is to make TensorFlow and Darknet API run in

the Robot Operating System (ROS) environment and quantify the accuracy of detection result. Mean average precision (mAP) is used as a performance metric for deep learning object detection systems, such as Faster RCNN and YOLO [6] [7]. However, as discussed in Section 2.4.2, mAP only reflects the rate of correctly detected object in a set of data, it does not quantify the difference between a detected bounding box and the ground-truth (the exact size of the bounding box). In order to serve as a foundation of a UAV tracking project, the location of a target UAV need to be known to send information to a control system so that a tracking UAV can be moved to follow the target correctly. The difference between the detected object location derived from its bounding box and its real-world location need to be compared to quantify the accuracy of a detection. In this thesis, the algorithm to estimate relative positions of a target and a tracking UAV from bounding box is derived to quantify accuracy, and convolutional networks, namely SSD MobileNet v1, SSD Inception v2, Faster RCNN Inception v2, YOLO v2 and Tiny YOLO, are benchmarked and compared.

Objectives of this thesis are listed below:

- Apply deep learning neural network to UAV detection project.
- Make detection systems work in ROS (Robot Operating System) and communicate with a Parrot ARDrone 2.0.
- Compare the performance of SSD MobileNet v1, SSD Inception v2, Faster RCNN Inception v2, YOLO v2 and Tiny YOLO taking efficiency, accuracy and consistency into consideration.
- Test the impact of camera calibration of Parrot ARDrone 2.0's onboard camera on the detection accuracy.
- Suggest an object detection system that is fast enough for real-time detection and capable of detecting target UAV over a complex background.
- Estimate relative 3D positions of the target and observing UAV from 2D bounding box information.

1.3 Outline of Thesis

This Chapter provides an overview of the thesis and motivation for the research. A statement of contributions is given at the end of this Chapter.

Chapter 2 covers the theoretical background of the work, surveys the state-of-the-art technology in deep learning, and covers the required background tools from computer vision.

Chapter 3 explains the tools used for the UAV detection project including the indoor motion capture system and its calibration, computing hardware, UAV, camera models and calibration, and software tools such as ROS, OpenCV, CUDA and Matlab.

Chapter 4 presents detailed experimental evaluations of the learning and detection performance of both TensorFlow and Darknet applied to UAV detection. Position and depth estimation errors relative to ground-truth from the motion capture system are quantified and analyzed.

Chapter 5 summarizes and discusses the experimental results. A recommendation is made regarding the choices of deep learning framework and convolutional network. Limitations of the present work and suggestions for future research are provided.

1.3.1 Statement of Contributions

The research contributions of this thesis are:

- Deploy two state-of-the-art deep learning frameworks, TensorFlow and Darknet, for real-time UAV detection within ROS. A wrapper of TensorFlow in ROS is developed as detailed in Appendix A.1. This code makes TensorFlow work in ROS and retrieve information from robotic hardware, such as image feed from UAV's onboard camera.

- A 3D location estimation algorithm is derived in this thesis using images from a monocular camera and bounding box information so that the accuracy of an object detection system can be quantified.

- Detailed benchmarking and performance comparison of detection results are conducted under various convolutional networks, namely SSD MobileNet v1, SSD Inception v2, Faster RCNN Inception v2, YOLO v2 and Tiny YOLO to select a best overall performer in ROS environment.

- Codes developed for bounding box information retrieval and 3D location estimation are detailed in Appendix A.2 and A.3. This automates the location estimation process and can be used as a direct input to a control system for UAV tracking.

- Code developed for training data extraction of Darknet is detailed in Appendix A.4. This is code is important for monitoring the training process of Darknet.

Chapter 2

Theoretical Background

2.1 Overview

Through a combination of an offline training stage and an online detection stage, Cascade Classifier was able to achieve real-time object detection in a video stream, specifically facial recognition [2]. However, the Cascade Classifier is poor at detecting objects in different conditions from the training dataset. For example, if the Cascade Classifier is trained with pictures of a drone heading left over a white background, it has difficulties detecting the same drone when it is heading right, or if flying over non-white backgrounds.

Thanks to the rapid growth in Artificial Intelligence (AI), the method of deep learning has brought in more powerful techniques for object detection such as artificial neural networks. Various deep learning frameworks are now publicly available including TensorFlow, Caffe, Keras, Darknet, etc. Good support and frequent updates from Google made TensorFlow the most popular deep learning framework in 2018 [8].

The object detection APIs within Darknet and TensorFlow are only officially supported under Linux, Windows or Mac OS. In order to interface the object detection system with the Parrot ARDrone 2.0 UAV used in our lab, the APIs were modified and implemented to run within ROS. The resulting output information such as bounding boxes, runtime speed, detection probabilities and reliability metrics were logged to the ground station computer.

In this chapter, the theoretical background of this thesis is covered. It starts with a brief introduction of deep learning. Then, the Artificial Neural Network

and its extensions, namely Convolutional Neural Networks, Faster Region-based Convolutional Neural Network (Faster RCNN), Single Shot MultiBox Detector and You Only Look Once (YOLO) are detailed. Different schemes to process data, for instance MobileNet and Inception are explained. Performance metrics for these methods are introduced. Finally, the equations for estimating 3D positions from 2D images are derived, followed by the distortion model and calibration procedure used for monocular cameras.

2.2 Machine Learning and Deep Learning

The idea of "machine learning" was first introduced in 1959 at IBM by Arthur Samuel describing the process of a machine learning from sets of data, a very early stage of development in Artificial Intelligence (AI). In 1997, a more formal and widely accepted definition of machine learning was given by Tomas Mitchell: Machine learning is a computer program improves its performance over a task given experience to learn from [9] [10].

A machine learning example following the definition by Tomas Mitchell is Google's AlphaGO. AlphaGO uses Monte Carlo tree search (MCTS) method to determine its next move in the classic board game from its training experience using artificial neural networks [11].

Deep learning was proposed for machine learning in 1986 by R. Dechter, with applications such as object detection, natural language processing, advertising, etc [12] [13]. It has the following definition:

- Deep learning uses numerous non-linear processing layers to extract and transform features, analyze and classify patterns in a supervised or unsupervised manner [14].
- A "deep architecture" is described by a hierarchical structure which uses lower-level features and concepts to represent higher-level ones [14].
- Deep learning uses the idea of artificial neural networks. The same lower-level features could contribute to the construction of many higher-level features [14].

There are three main deep learning architectures: generative deep architecture, discriminative deep architecture and hybrid deep architecture. Generative deep architecture is normally interpreted by graphical models [15]. There are three typical generative deep architecture models: deep belief network based on sigmoid belief network proposed by G.E. Hinton in 2006 [16], deep autoencoder and deep Boltzmann machine [17].

Discriminative deep architectures are usually divided into three approaches: Linear classifiers, logistic regression and support vector machine. The difference between generative and discriminative architectures is that the former uses joint probability while the latter uses conditional probability [18]. Given a training dataset of input x_{input} and corresponding output y_{output}, a linear classifier is used to represent the behavior of the dataset as [19],

$$f(x_{\text{input}}; w) = \arg\max_{y_{\text{output}}}[w^T \phi(x_{\text{input}}, y_{\text{output}})]$$

where w is a weight vector of object class, $\phi(x_{\text{input}}, y_{\text{output}})$ is the joint feature vector, $w^T \phi(x_{\text{input}}, y_{\text{output}})$ generates a score for likelihood of input x_{input} to be the class y_{output}, and $\arg\max_{y_{\text{output}}}$ chooses the class with the highest score. Empirical risk is introduced to optimize the training performance [19],

$$R_D(w) = \frac{1}{n} \sum_i l^{0/1}(x_{\text{input}_i}, y_{\text{output}_i}, c(x_{\text{input}_i}; w))$$

where $l^{0/1}(x_{\text{input}_i}, y_{\text{output}_i}, c(x_{\text{input}_i}; w))$ is a 0/1 loss function evaluating the likelihood of score vector $c(x_{\text{input}_i}; w)$ with training data, 0 for positive and 1 for negative, $c(x_{\text{input}_i}; w)$ is a vector of scores $w^T \phi(x_{\text{input}}, y_{\text{output}})$, n is the number of input and corresponding output pair, i is the i^{th} data. By minimizing the empirical risk, training performance can be optimized.

Logistic regression is an alternative to the 0/1 loss function when determining R_D in a probabilistic way, defining conditional probability distribution $p(y_{\text{output}}|x_{\text{input}}; w)$ [19],

$$p(y_{\text{output}}|x_{\text{input}};w) = \frac{1}{Z(x_{\text{input}};w)}\exp(w^T\phi(x_{\text{input}},y_{\text{output}}))$$
$$Z(x_{\text{input}};w) = \sum_{y_{\text{output}}} \exp(w^T\phi(x_{\text{input}},y_{\text{output}}))$$

where $Z(x_{\text{input}};w)$ is a partition function.

Another alternative to 0/1 loss function is "hinge-loss" measuring difference the maximum confidence in the classifier that the confidence in the correct class. "hinge-loss" is defined as [19],

$$\begin{aligned} l^{hinge}(x_{\text{input}_i},y_{\text{output}_i},c(x_{\text{input}_i},y_{\text{output}_i};w)) = & max_{y_{\text{output}_i}}(w^T\phi(x_{\text{input}_i},y_{\text{output}_i}) \\ & + l^{0/1}c(x_{\text{input}_i},y_{\text{output}_i};w)) \\ & - w^T\phi(x_{\text{input}_i},y_{\text{output}_i}) \end{aligned}$$

Due to the fact that hinge-loss function is continuous, however not differentiable, a constrained optimization problem could be studied as an equivalent substitute to minimize loss and weight vector w can be determined as [19],

$$w = \sum_i \alpha_i(y_{\text{output}_i})(\phi(x_{\text{input}_i},y_{\text{output}_i}) - \phi(x_{\text{input}_i},y_{\text{output}_i})$$

where α is a non-zero variable used for solving the constrained optimization problem.

The method using "hinge-loss" function is called a support vector machine [20]. Finally, hybrid deep architectures are a hybrid of generative and discriminative deep architectures [21].

2.3 Deep Learning Frameworks

2.3.1 Overview

In this Section, deep learning frameworks implemented for UAV detection in this thesis are introduced. Various detection models are available: Single Shot MultiBox Detection (SSD), Faster Region-based Convolutional Neural Network (Faster RCNN), Mask Region-based Convolutional Neural Network (Mask RCNN) for TensorFlow [22] and You Only Look Once (YOLO), Tiny

YOLO for Darknet [23]. Different from all the other detection models, Mask RCNN has an extra mask output over a class label and bounding box. This slows down the detection speed, it only ran at 5 fps on Nvidia Tesla P100 GPU with 16 GB of HBM2 memory, 3584 CUDA cores and a maximum memory bandwidth of 732 GB/s [24] [25]. GPU used in the lab is a Nvidia GTX 1080 Ti with the same amount of CUDA core, while less memory and bandwidth than Tesla P100. Hence, the running speed on lab computer is expected to be less than 5 fps [26]. This is not acceptable for real-time detection tasks and thus not tested in this thesis.

2.3.2 TensorFlow

TensorFlow is an open source software library for machine learning developed by Google based on DistBelief, a closed-source machine learning framework [27]. "Tensor" stands for multi-dimensional data and "Flow" stands for the manner of processing this data. TensorFlow computation is represented by the data flow graph containing nodes representing computations and the directed edges into and out of nodes are tensors as visualized in Figure 2.1. TensorFlow supports programming languages including Python, C++, Java, Haskell, GO and Rust.

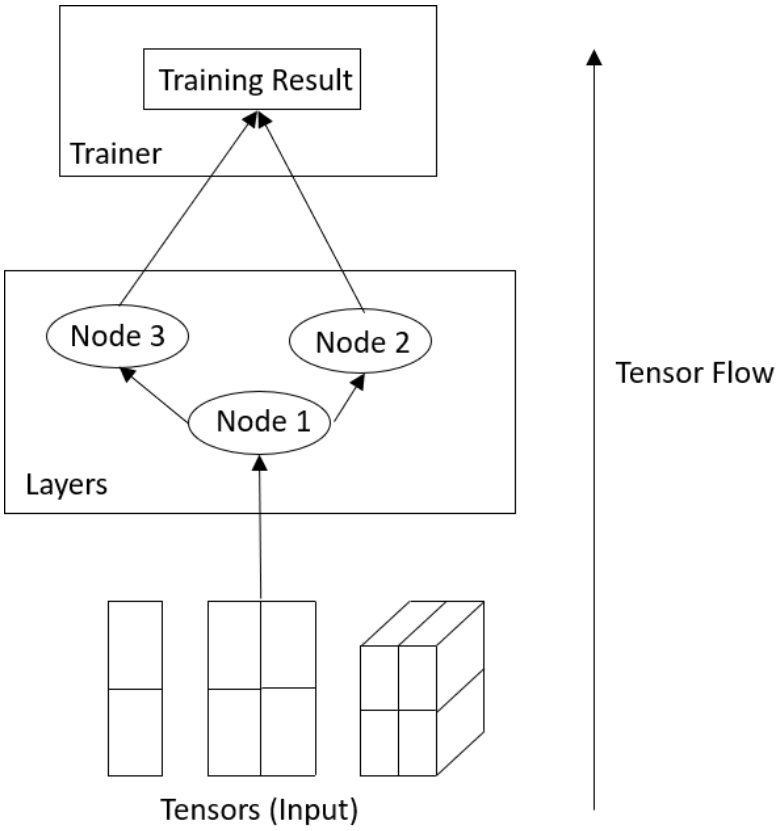

Figure 2.1: TensorFlow Graph

There are various modules in TensorFlow:

- High Level API is used to build and train deep learning models, write TensorFlow code, create pipelines to bring data into TensorFlow [28].
- Estimators save training checkpoints, process data of different types [28].
- Accelerators optimizes speed of TensorFlow by dividing tasks between CPU and GPU hardware [28].
- Low Level APIs handles computation tasks, saves and restores variables and models [28].
- Embeddings introduces embedding layers into TensorFlow [28].
- TensorBoard is a tool to visualize deep learning processes [28].
- Debugging is used for debugging TensorFlow code [28].

Various publicly available detection models are used in this thesis, namely SSD MobileNet v1, SSD inception v2 and Faster RCNN inception v2, which will be trained, tested and benchmarked for our purposes.

2.3.3 Darknet

Darknet is an open source machine learning framework written in C language and CUDA [23]. Compared to TensorFlow, Darknet has the following advantages:

- Darknet is faster than TensorFlow in specific tasks, for instance, object detection as shown in Section 4.3.1. Running speed becomes important when running without a high-end GPU.
- A ROS package written in C++ is publicly available on Github, which gives better compatibility with ROS.

Object detection models You Only Look Once (YOLO) and Tiny YOLO are trained, tested and benchmarked within Darknet and ROS environment.

2.4 Artificial Neural Network

The idea of artificial neural networks (ANN) comes from the brain's biological neural network. Neurons are input/output elements arranged in layers and their interconnection forms a neural network [29].

Three types of ANNs are deep feed-forward networks, convolution neural networks and recurrent networks. Deep feed-forward networks take data input into an input layer, proceed through multiple hidden layers, and give the results on an output layer. Convolution neural networks is an extension of deep feed-forward networks. CNN are detailed in Section 2.4.1. Unlike deep feed-forward networks, recurrent networks use feedback from output of layers when processing among neurons [30].

A traditional neural network is Multilayer Perceptron (MLP) that uses activation functions between neurons, where a perceptron is the simplest form of a neural network model [29] and activation functions are defined below.

An example of a typical MLP is illustrated in Figure 2.2, inputs are weighted through layers and gives corresponding outputs.

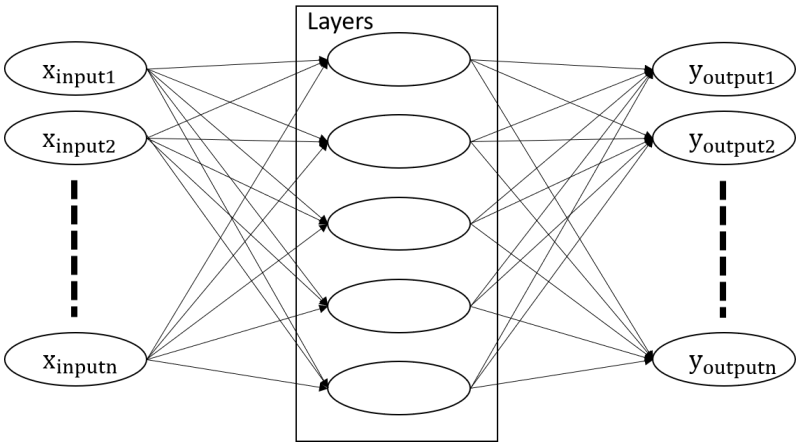

Figure 2.2: A typical MLP Structure

Output from the layers can be described by [29],

$$y_{\text{output}_i}(t+1) = \varphi(\sum_{j=1}^{n} w_{ij} x_{\text{input}_j}(t))$$

where $\varphi(\sum_{j=1}^{n} w_{ij} x_{\text{input}_j}(t))$ is called the activation function, which maps the sum of weighted inputs to the output. The input to a neuron can be computed from the output y_{output_i} of another neuron by the propagation function [31],

$$k_{\text{input}_j}(t) = \sum_{i} y_{\text{output}_i}(t) w_{ij}$$

2.4.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are one type of artificial neural network which specifically assume the inputs are images [32]. The first deep learning convolution neural network was Lenet proposed by Yann LeCun in 1998 for character recognition [33]. Figure 2.3 illustrates the architecture of Lenet-5, which has 7 trainable layers.

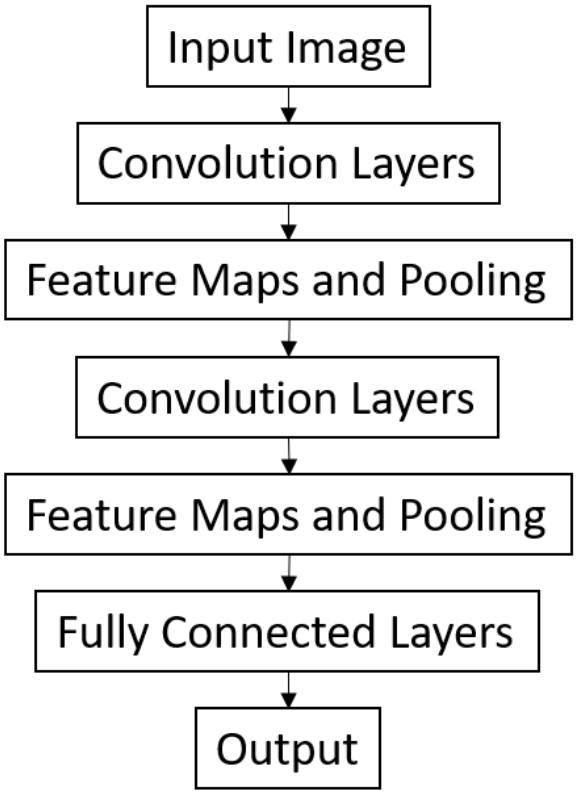

Figure 2.3: Lenet-5 Architecture

Based on the architecture of Lenet-5, it can be seen that there are three components in convolutional neural networks:

- The Convolutional Layer is the most important part of a CNN. It applies filters to input data with height M, width N and depth K $(M * N * K)$ to output features from the data [32].

- The Pooling Layer is placed between convolutional layers. The purpose of the pooling layer is to reduce the resolution of the output from a convolutional layer so that computing process are eased. Typical pooling methods are max-pooling, min-pooling and average-pooling, described by [34],

$$\text{Pooling}_j = \max(f_i^{n*n} s_{\text{region}}(n, x_{\text{input}}))$$
$$\text{Pooling}_j = \min(f_i^{n*n} s_{\text{region}}(n, x_{\text{input}}))$$
$$\text{Pooling}_j = \text{avg}(f_i^{n*n} s_{\text{region}}(n, x_{\text{input}}))$$

  where s_{region} is regions on feature map, n is the number of regions.

- The Fully Connected Layer connects all activations and neurons in the previous layers to compute the outputs through activation functions [32]. It parallels the MLP discussed in Section 2.4.

CNNs can be trained to identify specific objects. A measurement of the performance of training is "loss". Loss describes the difference between the ground truth and what the detection predicts as detailed in Section 2.2. The lower the loss, the better the detection performance of the trained CNN.

2.4.2 Mean Average Precision (mAP)

The performance of an object detection system is measured by its Mean Average Precision(mAP). The calculation of mAP relies on the concepts of recall and precision. The subsections below describe the details of mAP.

2.4.2.1 Intersection over Union (IoU)

Intersection over Union (IoU) measures the quality of the bounding box of a detected object. An example of a detected Parrot ARDrone 2.0 bounding box is shown in Figure 2.4.

Figure 2.4: Ground Truth and Bounding Box

The blue rectangle is the exact boundary of the object, usually known as the ground truth in object detection systems. The red rectangle is the bounding box obtained from detection. IoU is then calculated as [35],

$$IoU = \frac{\text{Ground Truth Area} \bigcap \text{Bounding Box Area}}{\text{Ground Truth Area} \bigcup \text{Bounding Box Area}}$$

In this thesis, bounding boxes with an IoU of 0.5 or more are considered to be a positive detection. This is also the default threshold used by TensorFlow object detection APIs.

2.4.2.2 Recall and Precision

Recall describes the rate of detecting targets. Precision describes the accuracy of these positive detections. Figure 2.5 illustrates the classification for items being detected in a dataset .

Detected Correct (DC)	Detected Incorrect (DI)
Undetected Correct (UC)	Undetected Incorrect (UI)

Figure 2.5: Classification of Items in a Detection

Recall and precision can be calculated as follows [36],

$$\text{Recall} = \frac{\text{DC}}{\text{DC} + \text{UC}}$$
$$\text{Precision} = \frac{\text{DC}}{\text{DC} + \text{DI}}$$

The higher the values of recall and precision, the more accurate the detection.

2.4.2.3 Average Precision (AP)

Both recall and precision need to be considered when measuring the accuracy of a detection system. Average precision (AP) is the area under the precision-

recall curve. AP is calculated as [36],

$$\text{AP} = \sum_{i=1}^{n} p(i)\Delta r(i)$$

where $p(i)$ is the precision at each detection, $\Delta r(i)$ is the recall difference between detection $i-1$ and i.

Mean average precision (mAP) takes an average over different detection sets and thus measures the overall accuracy of the object detection system. The calculation of mAP is shown below [37],

$$\text{mAP} = \frac{1}{n} \sum_{i=1}^{n} AP(i)$$

where n is the number of detection sets, and $AP(i)$ is the average precision at each set.

2.4.3 Faster Region-based Convolutional Neural Network (Faster RCNN)

Regions with CNN features (RCNN) was proposed by Girshick in 2013 [38]. According to Girshick (2013), the structure of RCNN can be illustrated by Figure 2.6. RCNN first extracts region proposals (around 2000) from the input image, extracts a feature vector from each region using a CNN, then scores each feature using its corresponding SVM [38]. It outperformed other approaches on the PASCAL VOC 2010 image dataset with its 50.2 mAP. However, the speed of RCNN is still not usable for real-time video processing since it requires around 10 seconds to process an image [38].

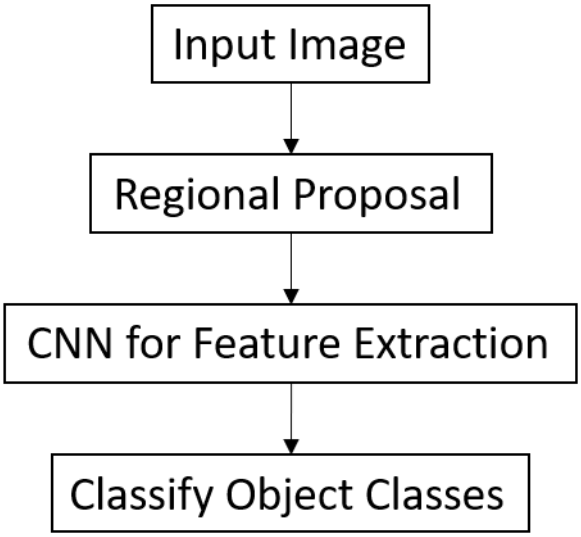

Figure 2.6: RCNN Structure

Fast RCNN is an extension of RCNN developed by the first author of RCNN in 2015 [39]. According to Girshick (2015), instead of running each region proposal through a CNN, the input image is run through several convolutional and max pooling layers to produce a convolutional feature map. Then, for each region proposal a pooling layer extracts a feature from the feature map, meaning feature extraction is done only once. Each feature is passed to a sequence of fully connected layers which yield classification probability and bounding box estimates using softmax layers for objects. Fast RCNN achieved 68.4 mAP on Pascal VOC 2012 dataset while RCNN only achieves 62.4 mAP. Image processing speed for Fast RCNN was 146 times faster than RCNN [39].

Faster RCNN was proposed by Ren and colleagues in 2015 [7] with the goal of optimizing the speed of Fast RCNN further. According to Ren (2015), Faster RCNN works by employing region proposal networks (RPN), a CNN which produces region proposals, to replace selective search algorithm in Fast RCNN. In this way the region proposal step can be carried out in around 10 milliseconds, allowing real-time object detection for the overall pipeline. On COCO dataset, Faster CNN achieved 41.5 mAP while Fast RCNN achieved 38.6 mAP. Running speed of Faster RCNN was about 10 times faster than Fast RCNN [7].

An overall evolution process from RCNN to Faster RCNN is listed in Table 2.1 [7] [38] [39].

Table 2.1: Summary of Evolution from RCNN to Faster RCNN

Detection Model	**Feature**
RCNN	Selective Search (SS) algorithm for region proposal; SVM for classification; CNN for feature extraction; Bounding-box Regression to minimize loss
Fast RCNN	SS algorithm for region proposal; Softmax layer for classification; CNN for feature extraction; Multi-task loss function to minimize loss
Faster RCNN	RPN for region proposal; Softmax layer for classification; CNN for feature extraction; Multi-task loss function to minimize loss

2.4.4 You Only Look Once (YOLO)

You Only Look Once was introduced by Redmon and colleagues in 2015 [40]. According to Redmon (2015), YOLO employs an end to end single neural network to reframe classification problem into regression problem that predicts bounding boxes and their associated probabilities in one evaluation pass to avoid complex pipeline [40].

Figure 2.7 illustrates the YOLO detection model. An image is split into an $S * S$ grid, and combinations of grids predicts bounding boxes, confidences and probabilities [40]. By setting a threshold, different combinations of grids with probability greater than the threshold will be presented.

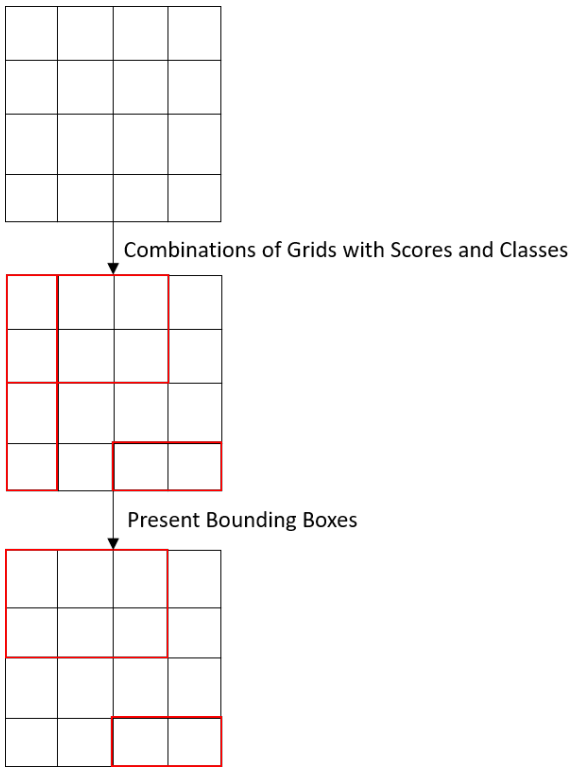

Figure 2.7: YOLO Detection Model

2.4.4.1 Single Shot MultiBox Detector (SSD)

Single Shot MultiBox Detector was proposed by Liu et al. in 2015 claiming to be both faster and more accurate than Faster RCNN and YOLO [41]. According to Liu (2015), SSD employs a base feed-forward convolutional network to produce an initial collection of bounding boxes and their associated detection probabilities, followed by a set of convolutional feature layers which progressively decrease in size and allow predictions at multiple scales. On the VOC 2007 dataset, SSD achieved 59 FPS and a mAP of 74.3 while Faster RCNN got 7 FPS, 73.2 mAP and YOLO got 45 FPS, 63.4 mAP [41].

2.4.4.2 You Only Look Once v2 (YOLO v2)

YOLO v2 is a newer version of YOLO which increases detection accuracy as well as efficiency proposed by Redmon in 2016 [6]. According to proposed by J. Redmon (2016), YOLO v2 employs batch normalization, training the classification network with higher-resolution images ($448 * 448$ vs $224 * 224$ on

YOLO), using anchor boxes to predict bounding boxes, finding good priors in the training dataset by using k-means clustering on the bounding boxes, predicting box location coordinates relative to the grid cells, concatenating low resolution $(13 * 13)$ and high resolution $(26 * 26)$ features for the detector, and training the network with a range of input image dimensions to make it capable of predicting across a range of input resolutions. Most of these features contribute to an improvement in mAP scores except for the anchor box, which increases recall. Overall, YOLO v2 is both faster and more accurate than YOLO. On VOC 2007 dataset, YOLO v2 achieved running speed as fast as 91 fps and mAP as high as 78.6, depending on image resolution [6].

2.4.5 MobileNet

MobileNet is a small and fast CNN architecture proposed by Howard et al. in 2017 [42]. It is optimized for low-power mobile and embedded vision applications. A standard convolution both filters and combines in one step. In MobileNet, this is replaced by depthwise separable convolutions, consisting of a separate layer for filtering and a separate layer for combining as illustrate in Figure 2.8. This dramatically reduces both computation and model size. Standard convolutions have the computational cost of $D_K * D_K * M * N * D_F * D_F$ where M, N are the number of input and output channels and $D_K * D_K$, $D_F * D_F$ are the kernel and feature map dimensions, respectively. Meanwhile depthwise separable convolutions have a computational cost of only $D_K * D_K * M * D_F * D_F + M * N * D_F * D_F$, or a fraction $1/N + 1/D_K^2$ of the standard convolution cost. According to Howard (2017), depthwise separable convolutions requires 8 times less computational power than a regular full convolution, with only 1.1% lost in accuracy [42].

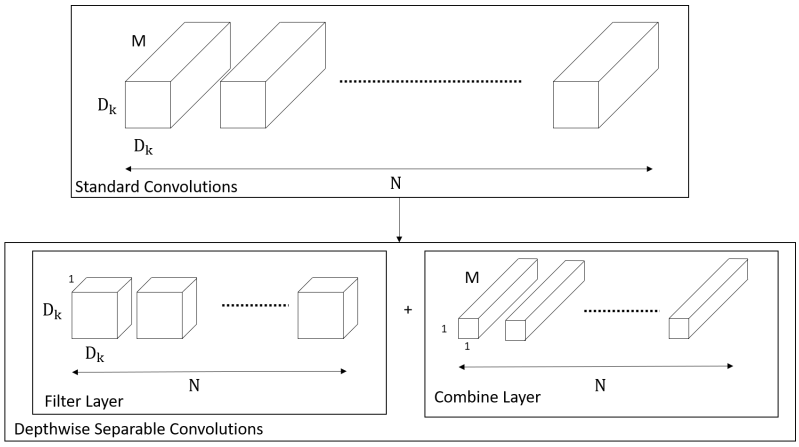

Figure 2.8: Depthwise Separable Convolutions

2.4.6 Inception

Inception v1 is a detection architecture proposed by Szegedy et al. (also from Google) in 2015, built on top of the Network-in-Network CNN introduced by Lin et al. in 2013 [43] [44]. The original design, Inception v1, consists of two models: "Naive" model and dimension reduced version model. An illustration of dimension reduced model over "Naive" model is shown in Figure 2.9.

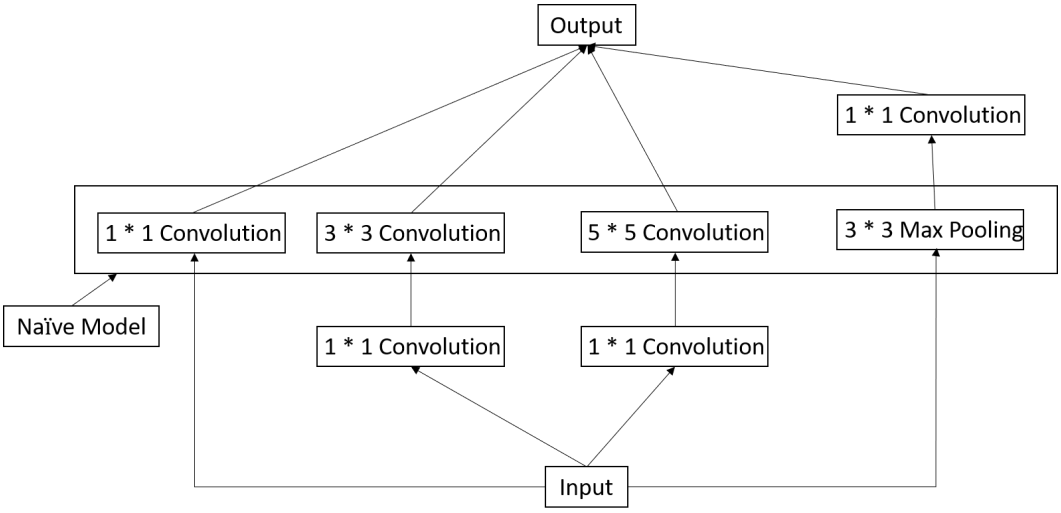

Figure 2.9: Structure of Inception v1 Dimension Reduced Model

As shown in Figure 2.9, the "Naive" model was introduced with input data with 3 different sized filters and a max pooling layer. However, this was inefficient and required an excessive amount of computation power. Extra 1 *

1 convolutions were thus added before the 3 * 3, 5 * 5 convolutions and after the max pooling layer. This reduces the dimension of the input to enhance efficiency [43].

In order to improve detection accuracy and speed, Inception v2 was introduced by the first author of Inception v1 in 2016 [45]. The architecture consists of a series of convolution steps, followed by Inception modules and a pooling filter bank leading to classification. The overall architecture is 42 layers deep and provided superior performance relative to other architectures achieving 21.2%, top-1 and 5.6% top-5 error for single crop evaluation on the ILSVRC 2012 dataset [45].

2.5 Distance Estimation from a Monocular Camera

Cameras like the Kinect have the ability to directly measure 3D images [46]. However, the Parrot ARDrone's front camera is a 720P monocular camera which can only capture 2D images. In order to track a UAV, its 3D location must be provided to the onboard tracking control system. Thus, we developed an algorithm to estimate relative positions between the target and the tracking drone from the onboard monocular video.

In order to verify the accuracy of estimated depth, the estimated results need to be compared against a ground truth. In this thesis, this information will be provided by the Vicon motion capture system discussed in Section 3.2.1. The comparison results will be provided in Section 4.3.2.

The Parrot's onboard camera is a monocular camera with resolution of 1280 * 720, capturing video at 30 fps with a 92-degree diagonal wide-angle lens [47]. The camera is modeled as a pinhole camera to estimate the relative horizontal, vertical and depth distance of the detected object to the optical center of the camera.

Figure 2.10 illustrates the 3D coordinate system defined for estimating the relative location between the target UAV and the observing UAV in the onboard camera video frame. Notice that the physical size of the object,

i.e. the height and width of the UAV need to be known for the algorithm in this section to work. If the size of the object changes, the relative position estimated would be changed as well.

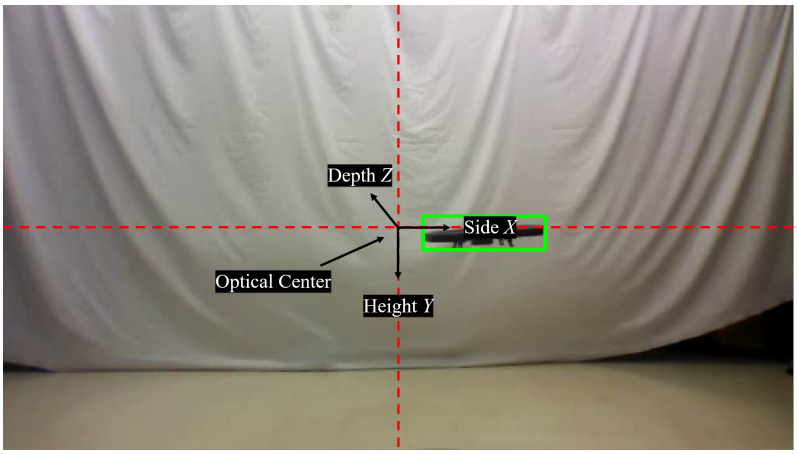

Figure 2.10: 3D Axis Illustration in Camera Video Frame

Figure 2.11 gives a schematic view of the object projection into the 2D image plane of a pinhole camera, assuming a ray from the object passes through the center of the lens and the object is always the same size and shape.

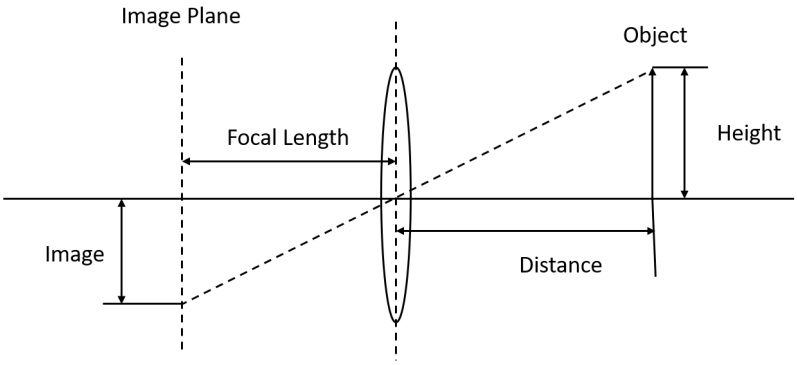

Figure 2.11: Image Formation of a Pinhole Camera in 2D Plane

The distance of the object from the camera lens can be calculated as

$$\text{Distance} = \text{Height} * \frac{\text{Focal Length}}{\text{Image}}$$

where distance, height and focal length are all expressed in SI Units.

Now extend this equation to 3D coordinate with the defined directions shown in Figure 2.10,

$$\begin{aligned}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} &= \begin{bmatrix} f\frac{X}{Z} \\ f\frac{Y}{Z} \\ 1 \end{bmatrix} = \frac{f}{Z}\begin{bmatrix} X \\ Y \\ \frac{Z}{f} \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \\ &= \frac{1}{Z}\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}\end{aligned} \tag{2.1}$$

where $(X,\ Y,\ Z)$ are the 3D coordinates of the object projected to $(x,\ y)$ on 2D image frame, f is the focal length. Notice that all the parameters in equation 2.1 are in SI units, i.e. meters, centimeters or millimeters. In order to relate the pixel length in a digital camera with SI units, transformation is done by,

$$\begin{aligned} x^{'} &= s_x x + c_x \\ y^{'} &= s_y y + c_y \end{aligned} \tag{2.2}$$

where s_x and s_y are in pixel/m scaling x and y in SI units to pixels, $(c_x,\ c_y)$ is the coordinate of the optical axis on the image, $x^{'}$ and $y^{'}$ are the coordinates of the object projected on the image. Combining equation 2.2 and equation 2.1,

$$\begin{bmatrix} x^{'} \\ y^{'} \\ 1 \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} s_x f & 0 & c_x \\ 0 & s_y f & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.3}$$

where the intrinsic camera matrix K is [48],

$$K = \begin{bmatrix} s_x f & 0 & c_x \\ 0 & s_y f & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where f_x and f_y are focal lengths in x and y direction in units of pixels. The intrinsic camera matrix K is upper-triangular and all of its diagonal entries are non-zero, thus K has an inverse. Projection matrix P is defined as,

$$P = \begin{bmatrix} s_x f & 0 & c_x \\ 0 & s_y f & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Relative position of the object and the camera lens can be estimated by the following assumptions:

- Object detection system provide precise bounding boxes around the target.
- The physical dimension of the object is known. (In this thesis, the dimension of the Parrot ARDrone 2.0 is measured to be 50.56 cm * 50.56 cm * 12.59 cm (width * depth * height).)
- The target's visible size, i.e. width and height, does not change much during the detection.

The third assumption requires the yaw, pitch, roll angles of the target relative to the observation target to be close to zero. For instance, the dimensions of Parrot ARDrone 2.0 are 50.65 cm in width and 12.59 cm in height from measurement using a metric ruler. However, if the drone yaws by 45°, its width perceived by the object detection system is equivalent to $\sqrt{50.65^2 + 50.65^2} = 71.63$ cm. One approach to avoid this assumption is to train the object detection systems to classify different orientations of the target UAV and dynamically assign a visible width and height.

Equation 2.3 gives the relationship between the image coordinate $(x^{'}, y^{'})$ and 3D coordinate (X, Y, Z),

$$\begin{aligned} x^{'} &= f_x \frac{X}{Z} + c_x \\ y^{'} &= f_y \frac{Y}{Z} + c_y \end{aligned} \tag{2.4}$$

Bounding box width $w^{'}$ and height $h^{'}$ in pixels are calculated from the detection result of the object in the digital camera as shown in Figure 2.12. Information of bounding boxes are given in pixels as $x^{'}_{min}$, $y^{'}_{min}$, $x^{'}_{max}$, $y^{'}_{max}$.

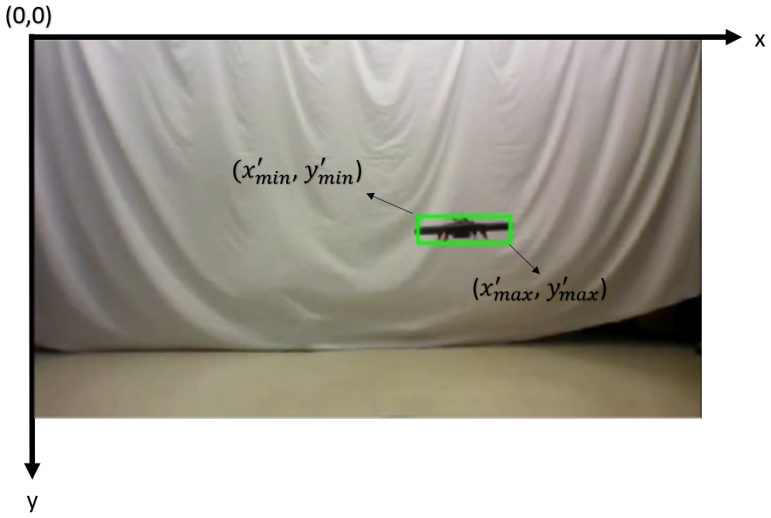

Figure 2.12: Bounding Box Given by Object Detection Systems

Bounding box width $w^{'}$ and height $h^{'}$ can be calculated as,

$$\begin{aligned} w^{'} &= x^{'}_{max} - x^{'}_{min} \\ &= (f_x \frac{X_{max}}{Z} + c_x) - (f_x \frac{X_{min}}{Z} + c_x) \\ &= f_x \frac{X_{max} - X_{min}}{Z} \\ &= f_x \frac{W}{Z_w} \\ h^{'} &= y^{'}_{max} - y^{'}_{min} \\ &= (f_y \frac{y_{max}}{Z} + c_y) - (f_y \frac{y_{min}}{Z} + c_y) \\ &= f_y \frac{Y_{max} - Y_{min}}{Z} \\ &= f_y \frac{H}{Z_h} \end{aligned} \tag{2.5}$$

where $(c^{'}_{min}, y^{'}_{min})$ is the top left coordinate of the bounding box, (X_{min}, Y_{min}) is the corresponding top left coordinate in 3D, $(x^{'}_{max}, y^{'}_{max})$ is the bottom right coordinate of the bounding box,(X_{max}, Y_{max}) is the corresponding bottom right coordinate in 3D, W is the width of the object and H is the height of the object, Z_w and Z_h are the perpendicular distance between the object and the camera lens estimated from width and height of the bounding box. From equation 2.5, perpendicular distance can be calculated by taking average of Z_w and Z_h,

$$Z = \frac{Z_w + Z_h}{2} = \frac{\frac{f_x W}{w'} + \frac{f_y H}{h'}}{2} \tag{2.6}$$

From equation 2.4, the 3D location (X, Y) of projected 2D point ($x^{'}$, $y^{'}$) can be obtained as,

$$\begin{aligned} X &= \frac{x^{'} - c_x}{f_x} Z \\ Y &= \frac{y^{'} - c_y}{f_y} Z \end{aligned} \tag{2.7}$$

The midpoint of bounding box on the image frame cane be calculated as,

$$\begin{aligned} x^{'}_{\text{mid}} &= \frac{x^{'}_{max} + x^{'}_{min}}{2} \\ y^{'}_{\text{mid}} &= \frac{x^{'}_{max} + x^{'}_{min}}{2} \end{aligned} \tag{2.8}$$

Combine equation 2.8 and equation 2.7, relative distance between the middle of the object and the camera lens in x and y direction can be calculated,

$$\begin{aligned} X_{\text{mid}} &= \frac{x^{'}_{\text{mid}} - c_x}{f_x} Z \\ Y_{\text{mid}} &= \frac{y^{'}_{\text{mid}} - c_y}{f_y} Z \end{aligned} \tag{2.9}$$

Relative position between the object and the camera lens can be obtained as (X_{mid}, Y_{mid}, Z). Notice that Z, the perpendicular distance to the camera, would not be correctly estimated if the third assumption, that the target's visible size does not change much during the detection, is violated.

2.6 Camera Calibration

In Section 2.5, equation 2.3 gives the mapping from 3D coordinates (X, Y, Z) in SI units with respect to camera lens to 2D coordinates on the image frame ($x^{'}$, $y^{'}$) in pixels based on the assumption that no distortion is on the image frame. However, as shown in Figure 3.16, the image frame from the onboard has observable distortions, i.e. curving the edges of the closet. In order to calibrate this distortion, the OpenCV built-in plumb-bob distortion

model, also known as the Brown-Conrady model [49] is used. The calibration process is conducted in ROS as detailed in Section 3.2.3.1.

Plumb-bob model calibrates the distorted image frame coordinate (x_d, y_d) to rectified image frame coordinate (x, y) by [48],

$$\begin{aligned} r^2 &= x_d^2 + y_d^2 \\ x &= x_d(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x_d y_d + p_2(r^2 + 2x_d^2) \\ y &= y_d(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y_d^2) + 2p_2 x_d y_d \end{aligned} \tag{2.10}$$

where k_1, k_2, k_3 are radial distortion coefficients caused by wide angle lens, p_1 and p_2 are tangential distortion coefficients caused by the lens plane not being perfectly parallel to the imaging sensor plane. Notice that (x_d, y_d) and (x, y) are in SI units.

Radial distortion coefficient, tangential distortion coefficients, intrinsic camera matrix K and projection matrix P can be obtained by performing camera calibration process as detailed in Section 3.2.3.1. Notice that the projection matrix P in the .yaml file shown in Section 3.2.3.1 is used to project 3D coordinates to calibrated 2D coordinates on the image frame [50].

2.7 Uncertainty Measurement

Uncertainty measurement is performance to evaluate the accuracy of measured distances by Vicon camera system, camera calibration matrices and length measurements using a ruler. Two types of errors are considered for uncertainty analysis: random and bias errors. Random error is the difference between the measured values of a repeated measurement. Bias error is the offset between the actual value and measured value [51]. In this thesis, bias error is caused by observation of metrics on a ruler when measuring the dimension of the Parrot ARDrone 2.0's indoor hull. Random error occurs in Vicon camera system measurements and camera calibrations.

From Section 4.6.0.1 and Section 4.6.0.2 discussing uncertainties in Vicon camera system and camera calibration, it can be seen that the distribution of

the samples are very close to normal distribution. The uncertainty for normal distribution can be calculated as [52] [53],

$$\begin{aligned} U &= z^* \frac{\sigma}{\sqrt{N}} \\ \sigma &= \sqrt{\frac{\sum_{i=1}^{N}(x_i - \overline{x})^2}{N-1}} \end{aligned} \tag{2.11}$$

where z^* is critical value defining the boundaries of the acceptance region [54]. z^* value depends on the confidence interval [55]. σ is standard deviation of measured dataset, N is the number of measurements, x_i is each measurement and $\overline{x}$ is the average of the measurements [56].

Student's t-distribution is used instead of normal distribution when the sample size is small [57], i.e. if the sample size is less than 30. If the sample size is large, t-distribution is similar to normal distribution [58]. The uncertainty calculation is similar to equation 2.11,

$$\begin{aligned} U &= t \frac{\sigma}{\sqrt{N}} \\ \sigma &= \sqrt{\frac{\sum_{i=1}^{N}(x_i - \overline{x})^2}{N-1}} \end{aligned} \tag{2.12}$$

where t is critical value for Student's t-distribution. It depends on the degree of freedom and confidence interval [59].

In order to calculate the uncertainties in side, height and depth distance estimation, Monte Carlo method is used. Monte Carlo method introduces the idea of random sampling to investigate uncertainty [60]. In equation 2.6 and equation 2.9, it can be obtained that the uncertainties comes from camera and projection matrices and measurement on height and width of the Parrot ARDrone 2.0. 1000 random samples are simulated on measurements of height and width of the UAV as well as f_x, f_y, c_x and c_y in camera matrices and projection matrices. One detection result of bounding box is used to calculated uncertainty. The Matlab code for Monte Carlo Simulation is shown in Appendix A.5.

Chapter 3

Experimental Tools

3.1 Overview

In this chapter, the hardware and software involved in the research are discussed. The Vicon Vero camera system is used to measure the true relative distance between the target drone and the tracking drone, both being the Parrot AR.Drone 2.0. A Nvidia GTX 1080 Ti GPU is used for acceleration of deep learning frameworks. The Robot Operating System (ROS) used to interface between the drones and ground computer is introduced. The Tracker 3 software processes the data from the Vicon Vero cameras. Other software including AR.FreeFlight 2.0 to control drone flight, TensorFlow and Darknet APIs to perform UAV detection and Matlab to process data are also discussed.

3.2 Hardware Tools

3.2.1 Vicon Motion Capture System

3.2.1.1 Layout of Vicon System Setup

Vicon is a motion capture system originally developed by Oxford Metrics in Oxford, England. Vicon has two types of cameras, namely the Vero and Vantage models [61] [62]. The Vantage is a high-end model which can be used outdoors, while the Vero is more affordable and only usable indoors. Due to the fact that the experiments conducted in this thesis are all in a lab environment, Vero cameras were used. The accuracy of 0.003 cm uncertainty in measurements (Vicon measurement uncertainty analysis is detailed in Sec-

tion 4.6.0.1), low latency of 3.6 ms [63] and high frame rates of 330 fps [63] of Vero camera make it sufficient for capturing the motion of Unmanned Aerial Vehicles (UAV) in this thesis.

For the purpose of our project, the Vero cameras are mounted on lab walls as shown in Figure 3.1.

Figure 3.1: Picture of Vicon Cameras Fixed to Wall

Fixing the cameras on the wall provides stable tracking capability for experimental tests. Also, this saves time since the cameras do not need to be re-calibrated after setup. However, camera calibration is still periodically required to maintain optimal performance. The full set of cameras in the lab are illustrated in Figure 3.2. This setup defines the area that an object can be detected.

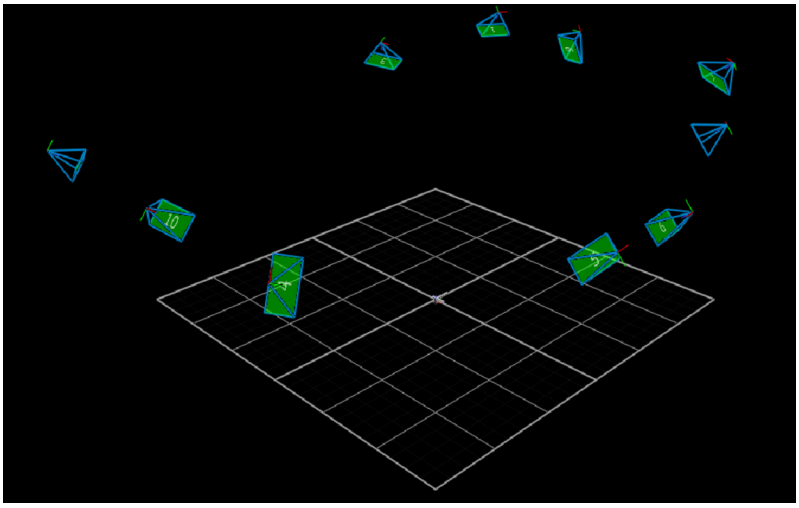

Figure 3.2: Vicon Camera Layout in the Lab

The green icons on the screen are the locations of the camera in the lab. The black and white grids are the valid capture area of the camera setup. Information from the Vicon cameras is transferred to a lab computer through individual Ethernet cables connected to a central gigabit Ethernet router.

3.2.1.2 Vicon System Vero Cameras and Markers

Vicon cameras need to be calibrated after they are fixed in place and anytime one of them is moved in order to provide accurate positioning performance. The wand used for calibration is shown in Figure 3.3. The LED lights on the wand in the figure are detected by the Vicon cameras as the wand is swung around the capture volume of the lab.

Figure 3.3: Vicon System Active Calibration Wand

Vicon Vero v2.2 cameras are employed in our lab. Videos are captured with 2.2 MP (2048 * 1088) resolution at 330 frames per second (FPS). This gives good and smooth motion capture performance. The cameras are equipped with a 6.5 - 15.5 mm varifocal lens, 44.1 * 23.6°(Horizontal * Vertical) for minimum standard field of view (FOV) and 98.1 * 50.1°for minimum wide FOV. Camera latency of 3.6 ms provides very low time delay in the system [63]. A picture of the Vero camera is shown in Figure 3.4. There are three rings on the Vicon Vero camera for adjusting zoom, aperture and focus. A good optical adjustment of the Vero camera results in the passive optical markers being recognized by the system.

Figure 3.4: Vicon System Vero Camera

Passive reflective markers are illuminated by infrared (IR) strobes on the Vicon cameras and recognized by the Vicon system. By attaching a set of markers to an object, its set of detected markers can be defined as a rigid body in the Vicon tracker software. Examples of UAVs with markers and their representation within the Tracker software are shown in Figures 3.5 and 3.6. The two UAVs have different combinations of markers to identify themselves as a tracking and target UAV.

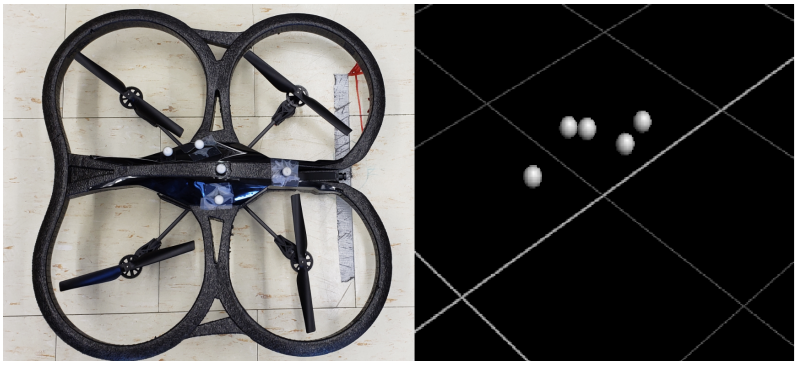

Figure 3.5: Black UAV with Marker Attached and its Representation in Vicon Tracker

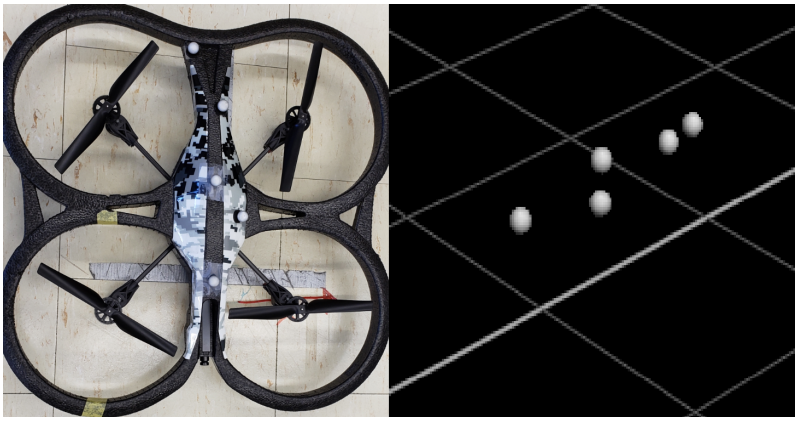

Figure 3.6: White UAV with Marker Attached and its Representation in Vicon Tracker

3.2.1.3 Vicon Vero Camera Calibration

The Vicon motion capture system is capable of estimating optical marker positions with an uncertainty of 0.003 cm in lab environment with the setup as shown in Figure 3.2. Discussion on Vicon system measurement uncertainty is detailed in Section 4.6.0.1. This allows it to be used as a ground truth in experimental UAV testing.

In order to provide this level of accuracy, the system needs to be calibrated every time the cameras are moved or disturbed. The wand shown in Figure 3.3 with active LED lights is placed on a floor landmark illustrated in Figure 3.7. For consistency, this landmark is used as the universal origin for all data collection experiments. The T-section of the landmark can guide the wand to

the same place when setting the origin. The calibration is performed within the Vicon Tracker software.

Figure 3.7: Landmark for Vicon Vero System

Lack of calibration is manifested in a number of ways. Compared with Figure 3.8, it could be seen in Figure 3.9 that the wand is not fully seen by the Vero cameras. There are also stray reflections which show up as white dots as seen in Figure 3.10. Both these factors will degrade the performance of the position estimates. The world frame is also set randomly as shown in Figure 3.11.

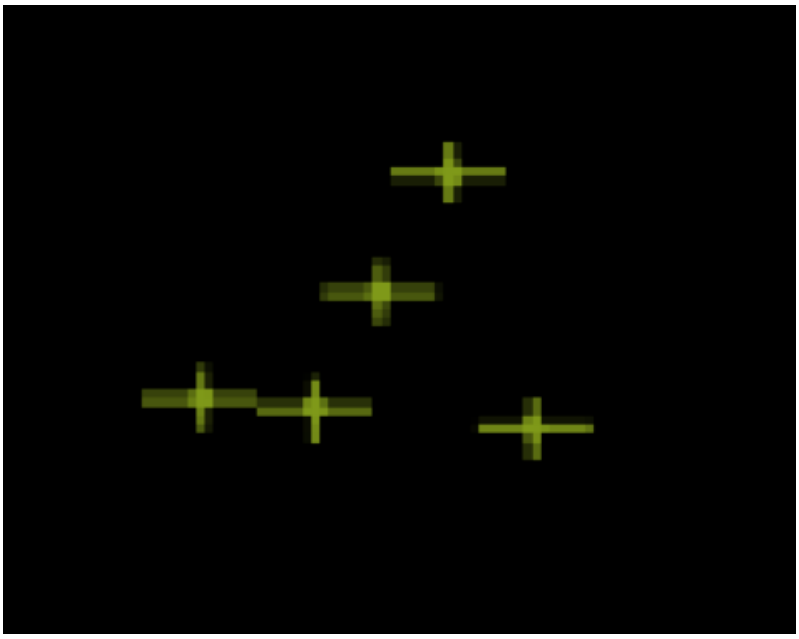

Figure 3.8: Wand Markers after Adjusting Vero Cameras

Figure 3.9: Poorly Detected Wand

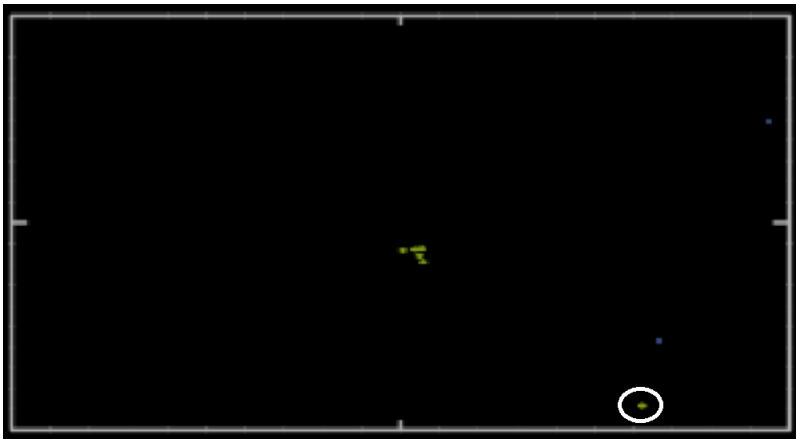

Figure 3.10: Stray Reflections in View of Cameras

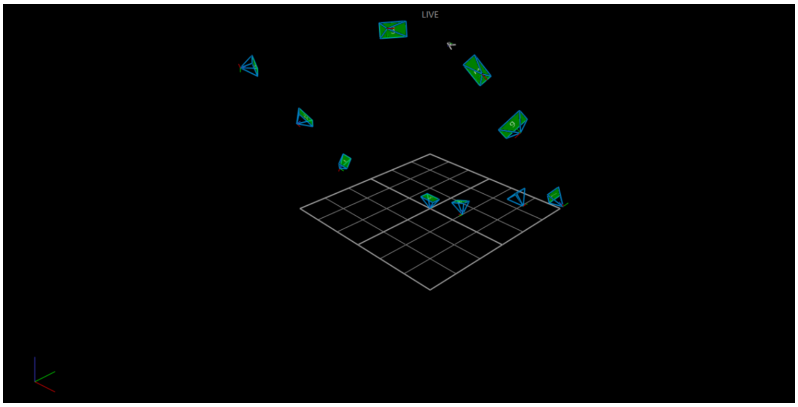

Figure 3.11: 3D View Before Defining Origin

To calibrate the camera system, Vicon Vero cameras need to be adjusted to recognize the wand properly. As introduced in Section 3.2.1.2, zoom, aperture and focus can be adjusted by twisting the rings on the camera. The camera 3D view of the wand from a well-adjusted camera are shown in Figure 3.12, all the markers are recognized correctly.

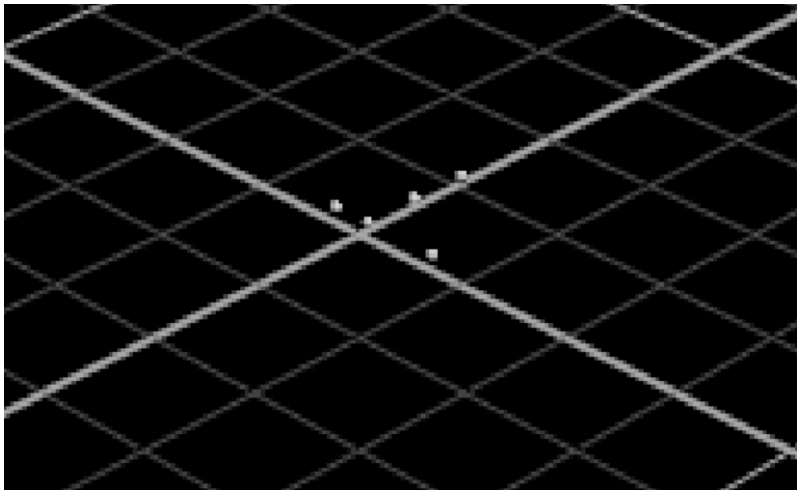

Figure 3.12: Wand is 3D View from Well-adjusted Camera

Reflections which appear as white dots on the ground can be eliminated by adding a "mask" to the camera view. This manually eliminates any white reflections seen by the camera when no optical markers are present. The masking process needs to be performed when the wand is out of the view of camera system. Otherwise, the wand's marker positions will be incorrectly masked out by the Tracker software.

Once the static calibration is done, the wand needs to be manually moved around the capture volume in order to calibrate the cameras' triangulation of markers. During this stage, the Vero camera indicator LEDs are red in colour. Once the calibration is complete these lights turn purple. Figure 3.13 shows the UI of Tracker3 for the wand calibration phase. Trajectory of the calibration wand is represented in the camera views and error of each camera during calibration process is displayed on the left bottom corner.

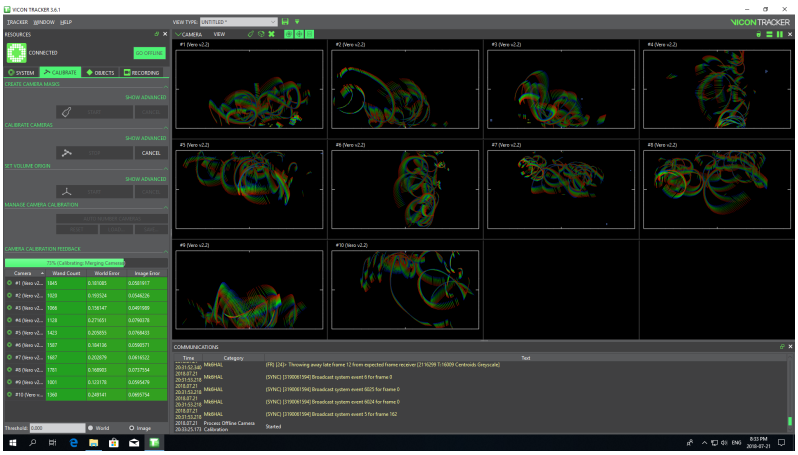

Figure 3.13: Vicon Vero Camera Calibration Process

Once a sufficient number of data points are captured in the wand calibration process, Tracker can perform the calibration of the motion capture system. A typical layout of this process is shown in Figure 3.14, the process bar will reach 100 % to indicate the completion of calibration. The green background in the error section shows that the motion capture system will deliver satisfactory positioning performance. Table 3.1 lists the numbers obtained following a calibration, indicating the errors achieved by each camera. Due to the fact that each camera receives different poses of the wand during calibration process as shown in Figure 3.13, the errors are not the same for every camera. For the purpose of this thesis, world errors around 0.2 mm are considered sufficiently good for running UAV experiments.

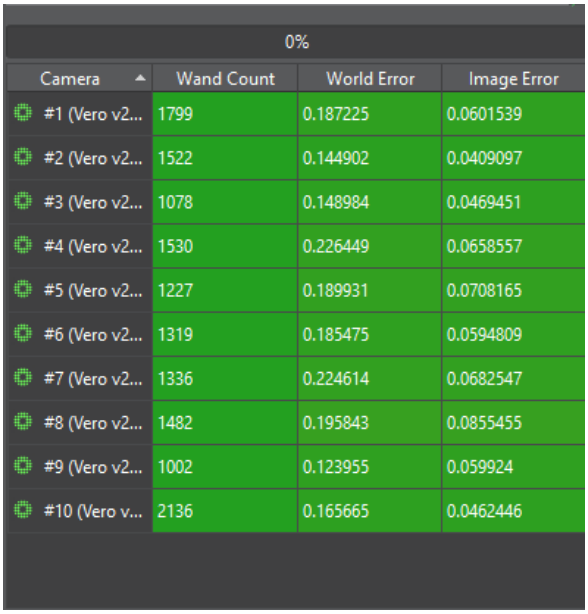

Figure 3.14: Camera View after Calibration

Table 3.1: Camera Calibration Feedback

Camera	**Wand Count**	**World Error (mm)**	**Image Error (px)**
1 (Vero v2.2)	1799	0.187225	0.0601539
2 (Vero v2.2)	1522	0.144902	0.0409097
3 (Vero v2.2)	1078	0.148984	0.0469451
4 (Vero v2.2)	1530	0.226449	0.0658557
5 (Vero v2.2)	1227	0.189931	0.0708165
6 (Vero v2.2)	1319	0.185475	0.0564809
7 (Vero v2.2)	1336	0.224614	0.0682547
8 (Vero v2.2)	1482	0.195843	0.0855455
9 (Vero v2.2)	1002	0.123955	0.059924
10 (Vero v2.2)	2136	0.165665	0.0462446

The world frame needs to be placed such that a consistent coordinate system is used for each set of experiments. The calibration wand is thus placed on the landmark on the floor shown in Figure 3.7. Tracker then places the origin of the world frame at this location.

3.2.2 Graphics Processing Unit (GPU)

There are two pieces of hardware in a computer for processing information, namely the Central Processing Unit (CPU) and Graphics Processing Unit (GPU). The CPU is designed for general computations. It is optimized for fast complex calculations due to its cores working at high frequencies, i.e. up to 5.0 GHz for an Intel I9 processor [64]. However, deep learning requires massive data processing in parallel [65]. A GPU is better at handling this due to its large amount of computing cores allowing parallelization of the calculations. Typically, a consumer class CPU has 8 to 12 computing cores, while a GPU can have upwards of 3000 computing cores [26] [64]. As a result, deep learning on a GPU can be accomplished well over 50 times faster than on a CPU [12].

This thesis employs two types of deep learning frameworks, Darknet and TensorFlow. The speeds of Darknet and TensorFlow depend heavily on the hardware they run on as GPUs can provide more computation cores than CPUs. To achieve real-time detection, a performance of at least 10 FPS is required. A GTX 1080 Ti CUDA-based graphics card built by Aorus is used to run Darknet and TensorFlow on the lab computer. This card has 3584 Nvidia CUDA cores with boost clock speed of up to 1746 MHz, 11 GB GDDR5X memory with 352-bit interface width and 494 GB/s bandwidth [26]. The technical specification of a GTX 1080 Ti may have minor differences depending on the manufacturer. A laptop-class Nvidia GTX 1060 GPU is used for data collection tasks, i.e. extracting bounding box information. It has 1280 Nvidia CUDA cores with boost clock speed up to 1708 MHz, 6 GB GDDR5 memory with 192-bit interface width and 192 GB/s bandwidth [66].

3.2.3 Parrot ARDrone 2.0

Parrot ARDrone 2.0 is a lightweight UAV drone designed for both indoor and outdoor flights[67]. It has a 720P (1280*720) onboard front camera with 92°diagonal angle capturing video at 30 fps [47]. The camera transmits the video feed via a Wi-Fi connection to a tablet, smart phone or computer. A

1500 mAh Lithium-Polymer (LiPo) rechargeable battery provides around 12 minutes of flying time. Parrot ARDrone 2.0 comes with an ARM Cortex A8 1 GHz 32-bit processor running Linux and a video Digital Signal Processor (DSP) running at 800 MHz [5]. An ultrasonic altimeter is located at the bottom of ARDrone 2.0 monitoring distance between the drone and any object beneath it using an emission frequency of 40 kHZ [67]. Other sensors include a 3-axis gyroscope, 3-axis accelerometer, 3-axis magnetometer and a barometer. Each propeller is driven by a brushless 14.5 W in-runner motors running at 28500RPM [47]. LED lights are located under each propeller to indicate the status of the drone; detailed information on their meanings are listed in Table 3.2 [67].

Table 3.2: AR.Drone 2.0 LED Light Indications

LED Light Condition	**Indication**
4 LED lights are all red	Power is connected but there is a problem with the drone
4 LED lights are all green	Drone is ready for takeoff
2 LED lights are green, other two are red	Used to distinguish front and back while flying
LED lights flash one after another	Motors initiating

Hulls with or without guard rings are attached to the drone before flight for indoor and outdoor use respectively. The AR.FreeFlight 2.0 application to pilot and monitor the Parrot ARDrone 2.0 is available on both iOS and Android. The set of ARDrone 2.0 parts is shown in Figure 3.15. The hull with markers is attached to the UAV so that it can be recognized by Vicon camera system.

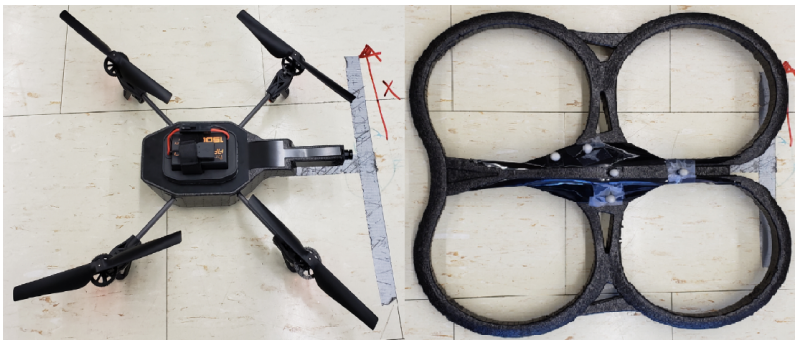

Figure 3.15: Parrot ARDrone 2.0 and Indoor Hull

3.2.3.1 Parrot ARDrone 2.0 Onboard Camera Calibration

The Parrot ARDrone 2.0 is equipped with an onboard wide-angle monocular camera causing barrel distortion as shown in Figure 3.16. The closet in the background is presented with curve on its edges and the metal rod on the right hand side is also curved. In reality, all these edges should be straight.

Figure 3.16: Unrectified Camera View

To rectify this distortion, the camera needs to be calibrated. A ROS package based on OpenCV's camera calibration module provides a tool to do this. A 5 * 7 chessboard with 0.032 * 0.032 m squares is printed for camera calibration. The ROS package identifies the chessboard in the camera frames, and provides visual feedback in the form of markers as shown in Figure 3.17, the connection points are marked as data points and perceived for calibration. By manually moving and tilting the chessboard, data points are collected by

the ROS package. Once all the shown indicators, namely x, y, size and skew, turn green, the system has sufficient data points to calculate the calibration parameters of the camera. The number of data points collected by ROS package for calibration is not limited. However, excessive number of data points may cause the package to crash.

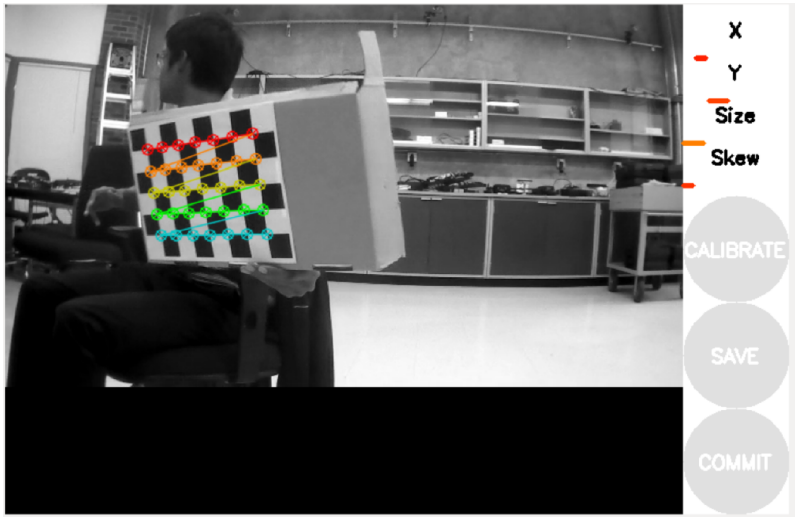

Figure 3.17: Marker Visual Feedback on Chessboard

Following the calibration process, a rectified view of the scene in Figure 3.16 is shown in Figure 3.18. The metal rod on the right hand side is straight after calibration, the bottom of the closet is straight, however, the top right part of the image is still distorted. The calibration process is a one-time process, meaning that after a calibration is done, ROS will load the same calibration file to calibration the camera for further operations.

Figure 3.18: Rectified Camera View

A side by side comparison of an image before and after calibration can be seen in Figure 3.19. It can be seen that the calibration gives good rectification in vertical, however, it still has distortions in horizontal, especially on the top right part of the image.

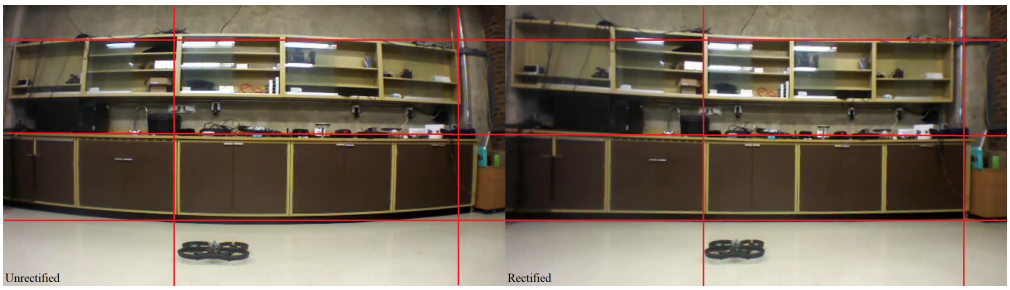

Figure 3.19: Comparison of an Image before and after Calibration

The reason that the calibration is not perfect in all the directions of the images is the limitation of the hardware. The ROS driver for Parrot ARDrone 2.0 limits the resolution to 640 * 360 instead of native 720P in order to achieve low latency in video transportation over Wi-Fi. This low resolution results in poor precision of calibration. The size option in Figure 3.17 never reaches full scale because the camera cannot identify the checkerboard when it is placed 2 meters away from the lens. Also, the size of the squares on the checkerboard is 3.2 cm * 3.2 cm, however, camera calibration package only acceptable integers, i.e. we have to input 3 cm * 3 cm instead of 3.2 cm * 3.2 cm. All these factors affect the output of the calibration.

Although the calibration is not perfect, it still gives good calibration in most part of the image, i.e. in Figure 3.18, only the top right corner of the image has pincushion distortions. Most importantly, it requires little computational power and gives desirable speed for real-time tasks when working with deep learning networks which requires intense computational power.

The camera calibration parameters can be saved to a .yaml file and used in subsequent experiments. The calibration file containing camera matrix, distortion coefficients, rectification matrix and projection matrix used for this thesis is shown below. The file needs to be loaded every time the system restarts.

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix: K
  rows: 3
  cols: 3
  data: [568.0748474282261, 0, 330.5770829093369, 0,
     ↪ 568.7139402113011, 193.6023640202218, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients: D
  rows: 1
  cols: 5
  data: [-0.5521437396858673, 0.2841312811070593,
     ↪ -0.0050346665442643, 0.006753347352338298, 0]
rectification_matrix: R
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix: P
  rows: 3
  cols: 4
  data: [449.2703857421875, 0, 344.3868504957354, 0, 0,
     ↪ 534.8646850585938, 194.3035218322311, 0, 0, 0, 1, 0]
```

The pipeline of ROS camera calibration package is detailed as following [50]. ROS transforms the distorted raw video through the inverse of camera matrix K, changing the units of the coordinates on the images from pixels to SI units. The scaled coordinates are run through plumb-bob distortion model with distortion coefficients in distortion matrix D to rectify the distorted co-

ordinates. Rectification matrix is only applied for stereo cameras, it does not do anything here as the onboard camera is a monocular camera. Finally, projection matrix P is used to change the calibrated coordinates from SI units to pixels.

3.3 Software

In this Section, software used in this thesis is described.

3.3.1 Robot Operating System (ROS)

The Robot Operating System (ROS) is an open-source, meta-operating system designed for robotics built on top of Linux [68]. It supports robots such as jackal unmanned ground vehicle, Parrot ARDrone 2.0 used in this thesis, etc. ROS is designed to run on Ubuntu Linux, however it can also run on top of other Debian-based distributions and potentially on Apple OS X as well [69]. However, running in OS X is still an experimental feature. Three important structures in ROS are nodes, topics and messages. Nodes are executable modules which realize specific functions. Messages are classes of data sent between nodes. Different types of messages from a single node are organized into groups called topics [68]. Topics are published by a node and can be subscribed to by another node if desired. A topic can be subscribed to by multiple nodes at the same time.

One advantage of ROS is that it supports various programming languages including C++ and Python. This allows porting codebases such as OpenCV into ROS where they are run through a software wrapper. An example of custom Python wrapper used is can be found in Appendix A.1.

For this thesis, ROS Kinetic is used running on top of Ubuntu 16.04. ROS support for Parrot ARDrone 2.0 was obtained from the ardrone_autonomy ROS driver available on GitHub, which supports this ROS distribution.

3.3.2 CUDA

CUDA is a programming platform for using GPUs developed by Nvidia [70]. Certain applications, for example, TensorFlow and Darket, can run over 50 times faster on GPUs supported by CUDA than on a CPU or other GPUs without CUDA support [12]. The main advantage of CUDA is that it automatically handles the back-end details of parallelizing computations and combining the results [65]. The thousands of cores on a GPU versus a traditional CPU with 8 cores makes it much more efficient for computations such as deep learning. Figure 3.20 illustrates CUDA work flow chart. It can be obtained that the CUDA core works in parallel to compute information.

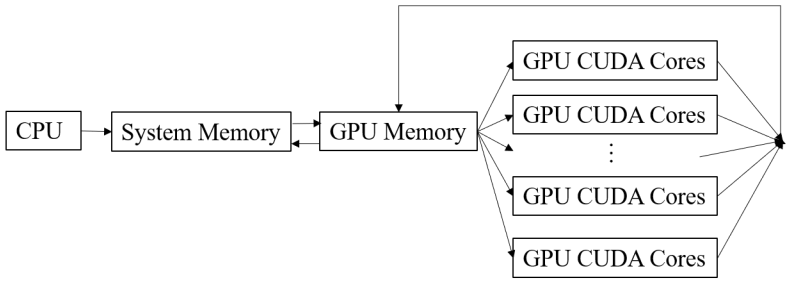

Figure 3.20: CUDA Work Flow Chart

3.3.3 Tracker

Tracker is a software developed by Vicon for tracking objects using optical markers captured by Vicon's cameras. The software can process data with a latency of 1.5 ms at more than 500 fps [71]. Camera calibration and rigid body object tracking is performed by this software as discussed in Section 3.2.1.2.

As shown in Figure 3.21, Tracker can display the position of a logged rigid body from the Vero cameras for visualization convenience, the list of logged rigid bodies are listed in the right.

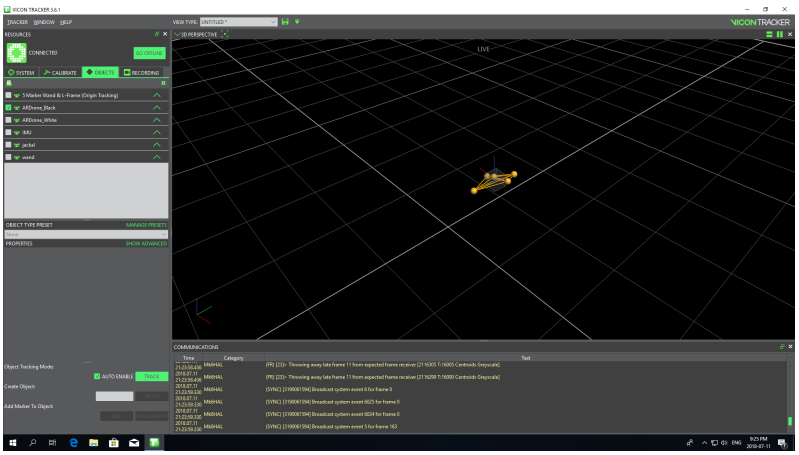

Figure 3.21: Position of a Rigid-body (UAV) in Tracker

3.3.4 AR.FreeFlight 2.0

AR.FreeFlight 2.0 is an application available on iOS and Android. This application provides various functions including piloting, recording the video from the onboard camera view, and setting the flight parameters such as speed limit, rotation limit, tilt angle limit, etc. This application is used to pilot the target UAV to do various flying patterns.

3.3.5 Vicon Bridge

Tracker introduced in Section 3.3.3 provides real-time tracking information about defined rigid bodies within view of the cameras. However, the current version of Tracker as of the day this thesis is written only runs on Windows 10 x64. Experiments in this project required to record pose information about the tracking UAV and target UAV from Tracker while running object detection UAV control modules on a Linux computer with ROS. This would require at least two people working at the same time to perform an experiment. A second problem would be the difficulty of matching the time indices between the two computers.

Vicon Bridge is a ROS package which allows streaming Tracker-produced data as a ROS topic. This solves the issue of synchronizing the time stamps of the Vicon information with the object detection modules, since both employ the same ROS timing clock. This also provides the ability to perform real-

time pose data processing in a ROS environment. For instance, Vicon Bridge can output the relative position between target UAV and tracking UAV, while Tracker can only output their individual poses.

3.3.6 OpenCV

Open Source Computer Vision (OpenCV) is a library of functions mainly used for real-time computer vision applications [72]. OpenCV is a free cross-platform library fist developed by Intel in 1999. It is now used in various deep learning frameworks including Caffe, TensorFlow, Torch and Darknet. The older Cascade Classifier algorithm is also supported in OpenCV. However, supports for APIs may not always be stable across different distributions of operation system. For instance, cascade classifier is not stable when testing object detection in Ubuntu 16.04 and ROS Kinetic with default OpenCV 3 library.

3.3.7 Matlab

Matlab is a numerical computing environment which provides a large number of mathematical functions and data visualization tools. It also provides a numerical simulation environment and a wide variety of field-specific toolboxes. There are object detection APIs available within Matlab which could potentially work for object detection task in this thesis [73]. However, they are not usually at the cutting edge, and slower than implementation of other programming languages such as C++ [74]. Thus, only the Linux-based TensorFlow and Darknet frameworks are used for object detection. Matlab is mainly used to implement the offline algorithms for depth estimation introduced in Section 2.5, accuracy calculation, Darknet object detection system training visualization and error calculation.

3.4 Experiment Summary

In this section, a summary of the experimental setup is presented in Figure 3.22. Parrot ARDrone 2.0 is used as both target and detect UAV. From

its onboard camera, the location of a target drone can be presented as a bounding box given by object detection systems. The detecting Parrot ARDrone 2.0 is connect to a lab computer for video processing, camera feed is presented as a topic in ROS and subscribed by object detection systems. By using the equations introduced in Section 2.5 and Section 2.6 for Plumb-bob camera calibration model built in OpenCV, the relative position of the two UAV can be calculated. The target UAV is piloted by AR.FreeFlight 2.0 on an Android tablet. Object detection systems are running on a GTX 1080 Ti GPU. Vicon camera system gives ground-truth location information of the two UAVs via Tracker and publishes the positions as a ROS topic using Vicon Bridge. Vicon Bridge gives real-time relative positions of the two UAVs as comparisons against detection results.

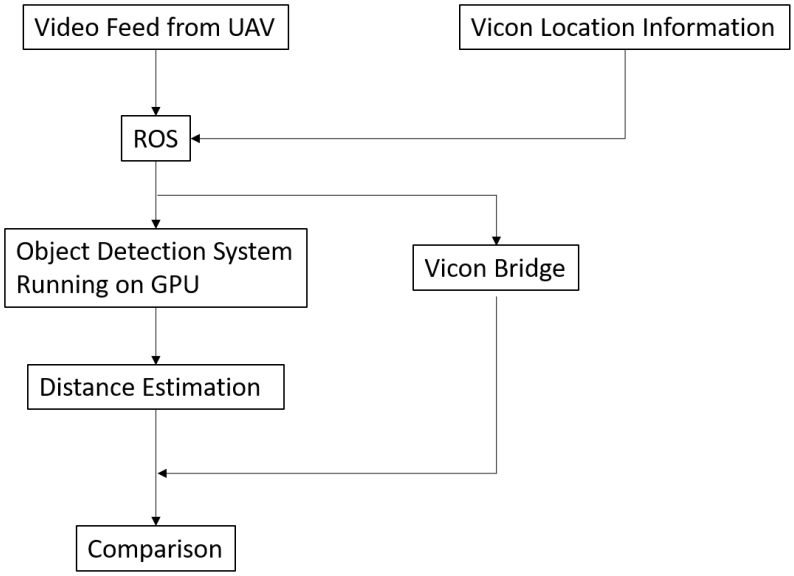

Figure 3.22: Experimental Setup Summary

Chapter 4

Experimental Results

4.1 Overview

Using the drone's onboard monocular video camera, 2D bounding boxes for a detected drone can be obtained in a 2D plane, allowing direct tracking of horizontal and vertical motion of the target. However, motion in the depth (into the plane) axis cannot be measured directly. In order to track the detected target drone in 3D, an algorithm was derived in Section 2.5 to transform 2D bounding box information in pixels into 3D positions in SI units.

In this Chapter, the performances of the studied object detection APIs, namely YOLO v2, tiny YOLO, SSD MobileNet v1, SSD Inception v2 and Faster RCNN Inception v2, are compared for efficiency, accuracy and consistency. Pose data from Vicon Vero camera system are used as the ground truth for the various object detection APIs. A trial drone flight captured by the Parrot ARDrone's onboard camera is recorded to a ground computer while Vicon bridge interface logs the poses of both UAVs. The recorded videos are then fed through the various object detection systems, and their detection results are compared against the Vicon logged data to assess their accuracy and consistency. Efficiency is tested by comparing the individual object detection systems' training time and running speed. In order to have a fair comparison, all training is conducted with the same set of images and the detection APIs are run on the same hardware platform, whose specifications are listed in Table 4.1.

Table 4.1: Lab Computer Specifications

Computer Component	**Specification**
CPU	Intel I7-8700K @ 3.70 GHz
RAM	32 GB DDR4-2666MHz
GPU	Nvidia GTX 1080Ti
Storage	2TB HDD 7200 rpm

4.2 Object Detection API Training

4.2.1 Overview

In this Section, the training efficiency of each object detection API is tested and compared. In order to make the comparison fair, all of the object detection APIs are trained on the same set of 1750 images and on the same computer whose specifications were listed in Table 4.1. The images were taken from videos recorded by a tracking UAV hovering and the target drone controlled to fly in different poses in the lab. Location of the target UAV is labelled manually as the ground-truth for training purpose. However, the batch size configuration for each API is customized to maximize its training efficiency. Batch size is a setting that controls the size of the data set being processed at each training step and affects the overall efficiency of training process [75]. Low batch size results in overly long training times, while setting the batch size too high leads to system crashes due to excessive resource demands.

The TensorFlow object detection APIs, namely SSD MobileNet v1, SSD Inception v2 and Faster RCNN Inception v2 come with convolutional weights pre-trained on the COCO (Common Objects in Context) dataset [76], a large (328k) set of images of common objects together with classification, localization and segmentation information of each. However, despite the pre-trained weights, the APIs were found to be poor at detecting the Parrot ARDrone 2.0. For this reason, further training of the COCO-derived weights was required. Note that training of the TensorFlow object detection APIs from scratch is possible, this would require an enormous amount of computation time.

The Darknet framework object detection APIs, namely YOLO v2 and tiny YOLO were trained in VOC format due to the fact that only pre-trained

weights on VOC dataset are available for YOLO v2 and Tiny YOLO by the time they were trained for experiments. However, it could not detect anything when YOLO v2 trained from a pre-trained weight caused by overfitting. Tiny YOLO did not exhibit this problem and worked fine training from a pre-trained weight. For fairness of comparison with YOLO v2 and TensorFlow APIs, a fully customized dataset is used for training. YOLO v2 and Tiny YOLO are significantly faster compared to the TensorFlow-based APIs on both training and detection as shown in this Section and Section 4.3. The efficiency of YOLO v2 and tiny YOLO make it possible to train from scratch. Annotations and training files are treated the same way for VOC dataset, which are interchangeable with COCO dataset and would give the same detection result if trained using COCO's format from scratch as well. The same dataset (1750 images) was used to train YOLO v2 and Tiny YOLO as that for SSD MobileNet v1, SSD Inception v2 and Faster RCNN Inception v2 to provide a consistent comparison.

4.2.2 TensorFlow APIs Training

The training process of SSD MobileNet v1 was finished in 37 hours and 40 minutes. Tensorboard was used to monitor the training process. Training went through 200k steps with a batch size of 42. Figure 4.1 illustrates the curve of loss function of the training process.

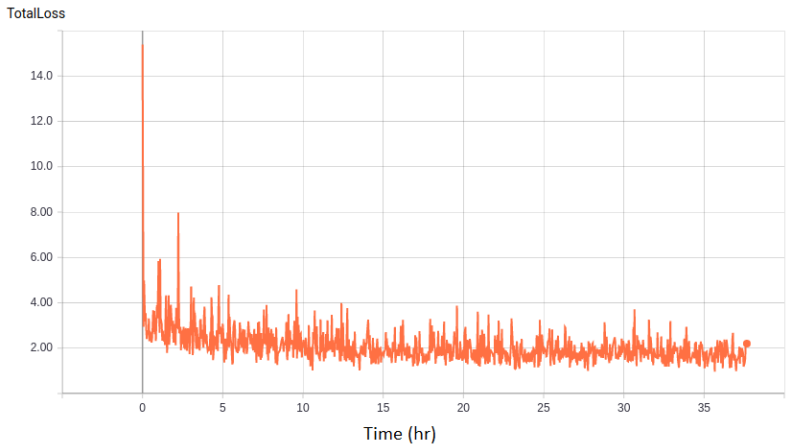

Figure 4.1: SSD MobileNet v1 Total Loss

Loss function is detailed in Section 2.2. The loss curve is spiky which makes it difficult to read the trend. Curve smoothing was conducted using a built-in feature of Tensorboard to add a low pass filter, resulting in the curve shown in Figure 4.2.

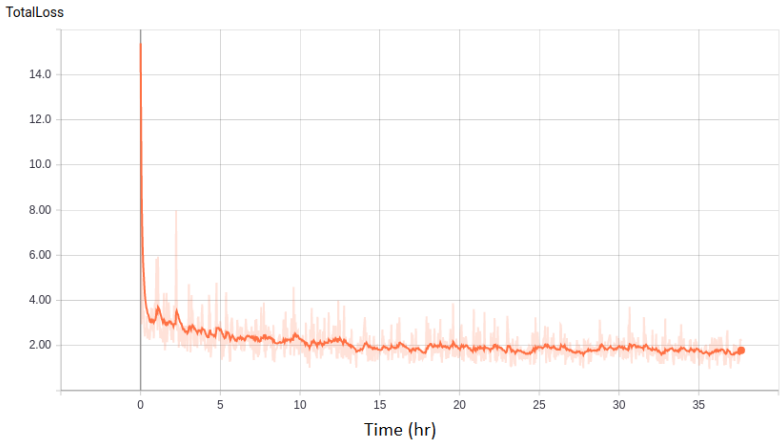

Figure 4.2: SSD MobileNet v1 Total Loss after Filtering

It can be seen that the total loss stabilizes and converges to 2.00. Table 4.2 lists the training parameters for SSD MobileNet v1, as well as SSD Inception v2 and Faster RCNN Inception v2. The loss curves of the latter two are shown in Appendix B.1.

It can be seen from the plots that the training processes completed with a converged loss curve using the default 200k step setting. Among the converged total loss values, Faster RCNN Inception v2 had the lowest at 0.05. Thus Faster RCNN Inception v2 can be expected to have the best detection performance among the three TensorFlow object detection APIs. Detailed results are listed in Table 4.2.

Table 4.2: TensorFlow Object Detection API Training Settings

TensorFlow Object Detection API	**Steps**	**Batch Size**	**Training Time (hr)**	**Converged Total Loss**
SSD MobileNet v1	200k	42	37.67	2.00
SSD Inception v2	200k	24	21.43	2.00
Faster RCNN Inception v2	200k	1	5.6	0.05

4.2.3 Darknet APIs Training

The training process of YOLO v2 finished in 14 hours and 30 minutes. The loss curve is shown in Figure 4.3.

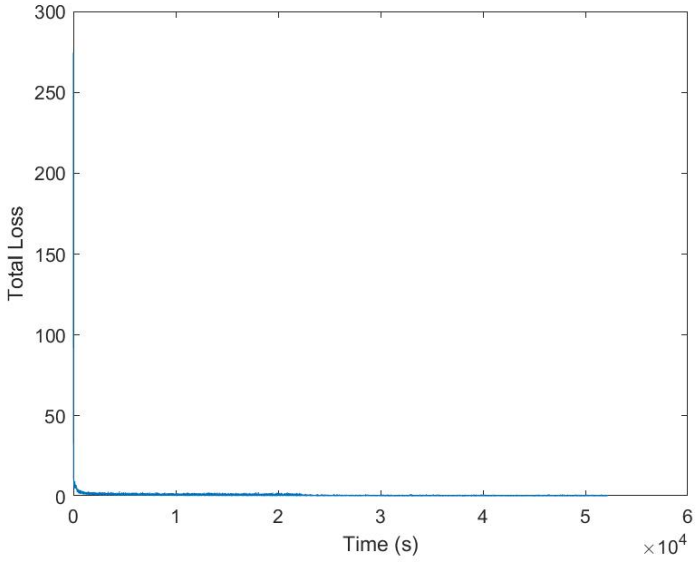

Figure 4.3: YOLO v2 Total Loss

Figure 4.3 illustrates the overall trend of the loss curve of YOLO v2 training. It can be seen that the total loss converges to a certain value, meaning the training is successful. However, due to the initial loss value of 274.6, it is hard to identify the converged loss value. In order to visualize the trend of the trend better, the figure is zoomed and smoothed as shown in Figure 4.4.

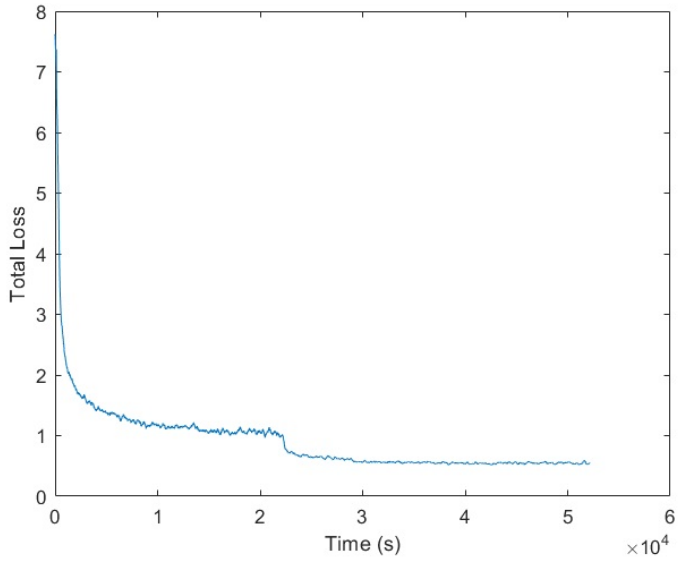

Figure 4.4: Zoomed and Smoothed YOLO v2 Total Loss

Figure 4.4 shows a zoomed-in and smoothed version of the loss curve. It can be seen that the total loss curve converges to 0.55. Table 4.3 lists the training parameters of both YOLO v2 and Tiny YOLO. The value of batch size divided by subdivision is the actual information feeding rate. Table 4.4 gives the actual batch size and subdivision settings used. Loss curve as well as zoomed and smoothed of Tiny YOLO can be obtained in Appendix B.1.

Table 4.3: Darknet Object Detection API Training Settings

Darknet Object Detection API	**Steps**	**Batch Size/-Subdivision**	**Training Time (hr)**	**Converged Total Loss**
YOLO v2	45000	8	14.5	0.55
Tiny YOLO	40200	16	9.33	0.93

Table 4.4: Batch Size and Subdivision Settings of Darknet API Training

Darknet Object Detection API	**Batch Size**	**Subdivision**
YOLO v2	64	8
Tiny YOLO	64	4

4.3 Object Detection Experimental Results

In this Section, the performances of each object detection system are evaluated. Three factors are considered to be the determinants when evaluating the performance of object detection systems: running speed, accuracy and consistency.

Running speed evaluates the efficiency and feasibility of an object detection system. An important part of UAV tracking is to acquire real-time target position information from the detection system. To qualify the requirement of "real-time", at least 10 fps detection should be achieved. Notice that all the detection tests are run on a full-tower computer with a GTX 1080 Ti GPU. The running speed could vary dramatically with a different model of GPU, for instance, YOLO v2 runs 71 fps on the GTX 1080 Ti used in lab while only runs around 25 fps on a GTX 1060. Lower frame rates could be expected when running on a smaller form factor and lower-power GPU such as the Nvidia Jetson TX2 with only 256 CUDA cores [77].

Accuracy is evaluated by taking the root mean square (RMS) error between the detected location given by the object detection system and the actual location from the Vicon motion-capture system. This is the most important part of the performance evaluation process. Accuracy is tied to the detection algorithm of an object detection API and cannot be easily optimized by upgrading hardware or changing the code parameters.

Consistency is measured by the capture rate of a target by the object detection system. It is inevitable for a detection system to lose the target over a complex background from time to time. Consistency indicates the performance of an object detection system in a complex working environment. To compensate for loss of feedback from the detection system when the target is lost, a tracker such as Re3 or a Kalman Filter could be implemented. These two methods are beyond the scope of this thesis and will only be briefly introduced. Throughout this Section, the Average Precision metric introduced in Section 2.4.2 is used to evaluate the consistency of the object detection systems.

Tests of the object detection systems include the differences between the object detector's estimated position and the Vicon motion capture system in the side(x), height(y) and depth (z) axes. Both rectified and raw video feeds as illustrated in Section 3.2.3.1 from the onboard camera are processed by the Darknet object detection APIs in order to assess the impact of camera calibration. Due to the poor running speed of TensorFlow APIs in ROS, only raw video feeds are processed to save computational power. Further tests and optimizations of TensorFlow-based APIs are expected in the future.

4.3.1 Running Speed

Running speeds of different object detection systems are tested on a lab computer with the specifications given in Table 4.1. Running speeds in both "native" Linux and in ROS are compared. From the testing, all TensorFlow and Darknet APIs run faster on Linux than in ROS. This is expected as ROS is an operating system running on top of Linux. As shown in Table 4.5, Tiny YOLO and YOLO v2 are qualified for real-time tasks in both Linux and ROS. TensorFlow-based APIs SSD MobileNet v1, SSD Inception v2 and Faster RCNN v2 are only qualified for real-time performance in Linux. Notice that the ROS wrapper for TensorFlow is self-developed in Python as shown in Appendix A.1. This leads to the limitation on the speed of TensorFlow in ROS.

Table 4.5: Object Detection Systems Speed Comparison

Object Detection API	**Speed in Linux (fps)**	**Speed in ROS (fps)**
SSD MobileNet v1	188.35	2.72
SSD Inception v2	107.32	1.98
Faster RCNN v2	21.25	1.97
YOLO v2	71.11	67.30
Tiny YOLO	140.51	73.80

4.3.2 Accuracy

In this Section, detection results of a target UAV as shown in Figure 3.6 from SSD MobileNet v1, SSD Inception v2, Faster RCNN v2, YOLO v2 and Tiny

YOLO are compared against Vicon motion capture system data in the side (x), height (y) and depth (z) directions. Root mean square error (RMS Error) is used to evaluate the performance of the detection results.

Eight trials are conducted to collect data. A white curtain is used to provide a simple background for the object detection systems to capture the target UAV as the best case scenario as shown in Figure 4.5. The unrectified camera view is shown on the left and the rectified camera view is shown on the right. There is very little noise in the background and good detection results can be expected.

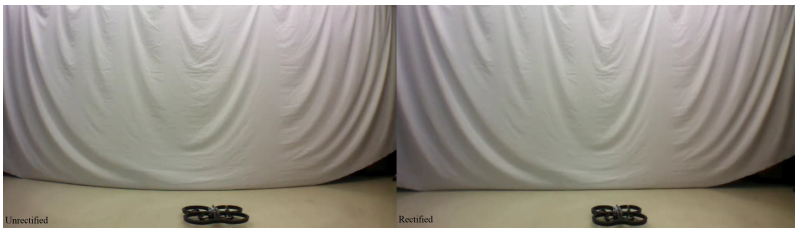

Figure 4.5: White Curtain Setup

In order to get more realistic testing conditions, a more complicated background is also used for detection tasks as shown in Figure 4.6. Both the unrectified and rectified camera view is shown in Figure 4.6.

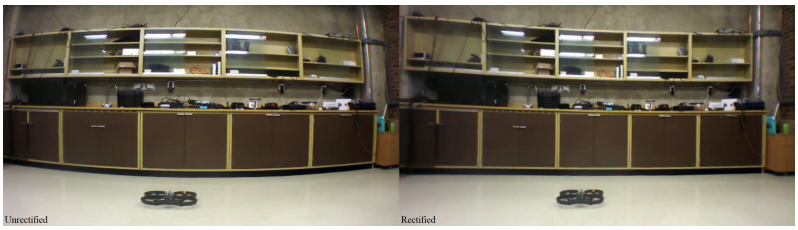

Figure 4.6: Complex Background Setup

There are similar patterns to the UAV presented in the background such as an indoor UAV hull sitting on the middle part of the bench and the rectangular sized shapes can also confues the detection system with a real target UAV.

TensorFlow APIs are trained with unrectified images recorded from the onboard camera. This is the image dataset used for training comparison in Section 4.2. Notice that the CNNs used for TensorFlow and Darknet APIs

could detect target drone in rectified images as well despite the fact that it is trained with unrectified images. Both rectified and unrectified (raw) video streams are processed by the object detection systems to test the impact on accuracy from camera calibration. YOLO v2 and Tiny YOLO are trained with both unrectified (the same dataset as TensorFlow) and rectified images to provide more comparisons in experimental testing. Due to the fact that YOLO v2 and Tiny YOLO qualifies the speed for real-time detection, they are more likely to be used for the extension of the UAV tracking project. The comparison between detection result of unrectified and rectified training dataset can quantify the impact of a rectified training dataset. Notice that all the relative distance estimations from the detection results are calculated with the three assumptions discussed in Section 2.5. Validation of the assumptions is discussed in Section 4.5.

A systematic error analysis is used when conducting experiments. First, two trials of simple movements in side and height directions are operated, as this is a 2D movement which can be directly observed in the onboard camera view. Next, two trials of back and forth movements in the depth direction are operated. This is more difficult for detection since the onboard monocular camera cannot give position information in the depth direction. Depth estimation solely relies on the algorithm derived in Section 2.5. Next, two trials of UAV rotations are conducted. Due to the shape of the Parrot ARDrone 2.0, rotation of the vehicle causes the bounding box to change size. These two trials of rotation movement are thus intended to test detection reliability when the target changes yaw angle. Finally, two trials of complex flight patterns are conducted. These two operations are intended to replicate realistic flight scenarios. Detailed experimental results are listed in Appendix B.2.

4.3.2.1 Offset in Vicon Camera System

In Vicon camera system, the two UAV are logged as rigid bodies as shown in Figure 3.21. Relative distances are measured from the center of one rigid body representing a UAV to the center of the other one as shown in Figure 4.7.

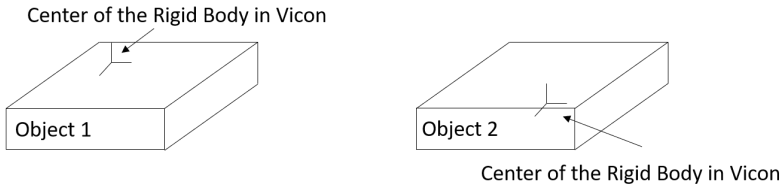

Figure 4.7: Offset in the Vicon Camera System

However, the relative distances calculated from the bounding box using the equations in Section 2.5 are calculating the distance between the surface of the UAV facing the camera and the center of the lens. The rigid body can represent the relative movement of the two UAVs, however, due to the difference between the structure of the logged rigid and physical UAV, there are offsets in side, depth and height directions. Vicon bridge allows to align the center of the rigid bodies to the physical center of the UAVs in order to eliminates the offset in the side and height direction, however, the offset in the depth direction cannot be eliminated due to the physical structure of the UAV.

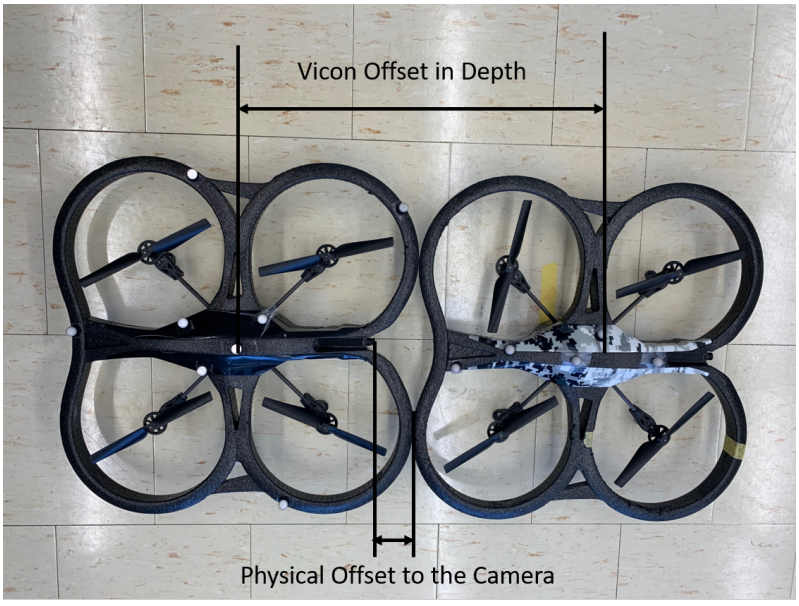

Figure 4.8: Zero Distance in Distance Estimation

Figure 4.8 illustrates a 0 distance between the two UAV for depth distance estimation from bounding box calculation. There are two offsets occur: the

offset from Vicon after aligning the center of rigid bodies in Vicon bridge and the physical offset from the camera to the front surface of the UAV. The calculation of the offset from Vicon depth and depth estimation using bounding box is:

$$\text{Depth}_{\text{offset}} = \text{Vicon}_{\text{offset}} - \text{Physical}_{\text{offset}}$$

$\text{Vicon}_{\text{offset}}$ is 50.56 cm and $\text{Physical}_{\text{offset}}$ is measure to be 6.19 cm. Offset in depth can be calculated as 44.37 cm. The distance estimation of translations in depth direction (Z direction) before and after calibrating for the offset from one set of the Faster RCNN Inception v2 detection results is shown in Figure 4.9 and Figure 4.10.

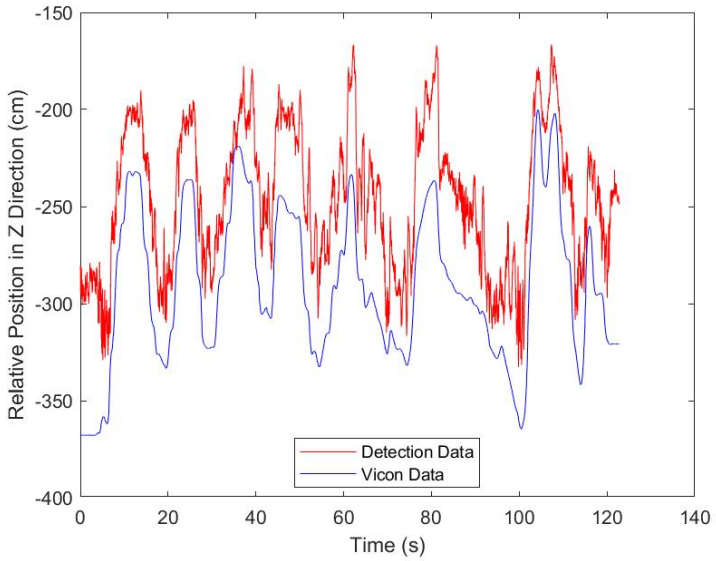

Figure 4.9: Distance Estimation in Depth before Offset Calibration

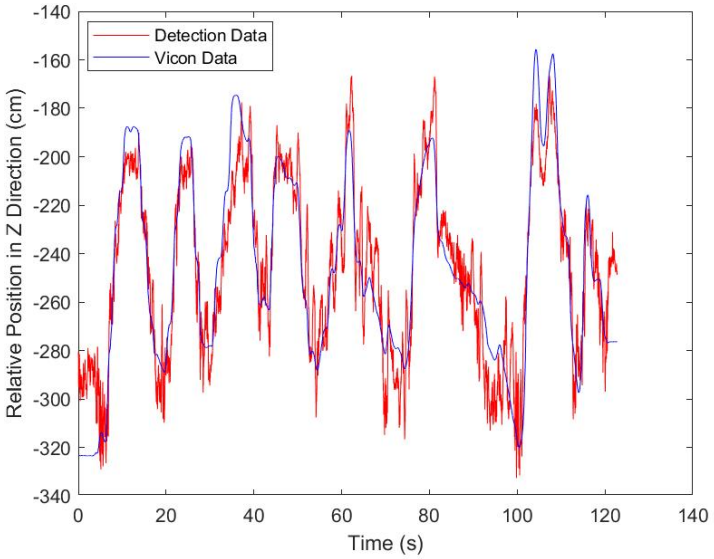

Figure 4.10: Distance Estimation in Depth after Offset Calibration

Comparing Figure 4.9 and Figure 4.10, it can be obtained that the offset in depth distance estimation is eliminated by offset calibration.

4.3.2.2 Root Mean Square Error (RMS Error)

Root mean square error (RMS Error) is used as an accuracy metric in this thesis. The expressions to calculate RMS Error is shown below [78],

$$\text{Error}_p = \text{Detection Data}_p - \text{Vicon Data}_p$$

$$\text{Root Mean Square Error} = \sqrt{\frac{1}{n}\sum_{p=1}^{n}(\text{Error}_p)^2}$$

where detection data is calculated from the estimated bounding box information together with the equations derived in Section 2.5, Vicon data is used as "true" value, n is the number of data points in each trial of the detection tests.

The errors in the first trials of detection tests, namely pure translation in the side, height and depth directions with white curtain background, trained by unrectified images and detect in unrectified camera views are plotted fully to visualize the accuracy and consistency of the object detection systems. All

the other errors are summarized by a single RMS value in the tables and listed in Appendix B.2.

4.3.2.3 Camera Calibration Results

In Appendix B.2, RMS errors between distance estimations from bounding boxes of object detection systems, namely SSD MobileNet v1, SSD Inception v2, Faster RCNN Inception v2, YOLO v2 and Tiny YOLO, and Vicon system data in side, height and depth directions are listed. In this Section, discussions on the impact of camera calibration on both camera video and training dataset are detailed.

Despite the fact that TensorFlow APIs are trained on unrectified image dataset, object detection system can detect target UAV in both rectified and unrectified camera videos. The percentage difference between detection results on rectified and unrectified videos are listed in Table 4.6. With positive percentages means that more error is presented in rectified videos.

Table 4.6: Difference of TensorFlow Detection Results on Rectified and Unrectified Videos

Detection System	**Difference in Side x (%)**	**Difference in Height y (%)**	**Difference in Depth z (%)**	**Average Difference(%)**
SSD MobileNet v1	6.21	18.42	72.11	36.30
SSD Inception v2	3.85	25.74	73.59	35.70
Faster RCNN Inception v2	127	14.95	56.08	68.24

It can be obtained from Table 4.6 that all of the detection systems performs worse on rectified videos. This is the results of insufficient training on rectified images. Faster RCNN Inception v2 has the most difference in the detection results.

Darknet APIs are trained with both rectified and unrectified images. YOLO v2 is able to detect target UAV in both rectified and unrectified camera videos with unrectified or rectified training dataset. The difference on detection results with various setup is listed in Table 4.7. With positive percentages means that more error is presented in the later compare setup. For instance, training

rectified/ video unrectified vs training rectified/ video rectified giving 15.09% indicates that the former setup has less error.

Table 4.7: Difference of YOLO v2 Detection Results with Different Setup

Setup	**Difference in Side x (%)**	**Difference in Height y (%)**	**Difference in Depth z (%)**	**Average Difference(%)**
Training Unrectified/ Video Unrectified vs Training Unrectified/ Video Rectified v1	3.92	18.83	40.45	24.8
Training Rectified/ Video Unrectified vs Training Rectified/ Video Rectified	16.51	28.68	4.61	15.09
Training Unrectified/ Video Unrectified vs Training Rectified/ Video Rectified	80.63	94.38	1.8	28.37

It can be obtained from Table 4.7 that detection on rectified videos are worse than the detection results on unrectified videos with both training rectified and unrectified. It is reasonable that detection result is better with training unrectified/ video unrectified setup than training unrectified/ video rectified setup. This is because the object detection system is more familiar with unrectified images. Ideally, training rectified/ video rectified setup should be better than training rectified/ video unrectified setup as the object detection system is more familiar with rectified images. Training rectified/ video rectified setup is worse than training unrectified/ video unrectified indicates that camera calibration process introduces error in detection results. Poor detection results of flights patterns with complex background contributes the most to the huge increase in error. For instance, with white background, the error difference between training unrectified/ video unrectified and training rectified/ video rectified is 1.9 % decrease in side, 18.05 % increase in height and 3.06 % increase in depth. With complex background, error in side increases by 163.13 %, error in height increases by 170.71 % and the error in depth increases by 0.54 %. The reason for camera calibration gives worse accu-

racy is that the projection matrix used to calculate the relative position is not accurate. The uncertainties in camera and projection matrices are detailed in Section 4.6.0.2.

Unlike YOLO v2, Tiny YOLO cannot detect anything with unrectified training dataset. The reason can be insufficient training data or overfitting. With rectified training dataset, Tiny YOLO is able to detect target UAV in both rectified and unrectified camera videos. Compared to detection on unrectified camera videos, detection on rectified camera videos is 5.01 % worse in distance estimation on side x axis, 9.66 % worse on height y axis and 1.5 % better on depth z axis. Overall, the average of the distance estimations on x, y and z axes are 1.53 % worse with rectified camera videos. The reason for the training rectified/ video rectified setup to be worse is the same as YOLO v2 and detailed in Section 4.6.0.2.

4.3.2.4 Accuracy of Object Detection Systems

In this Section, accuracy of each object detection system is compared. Due to the fact that all the tested object detection systems are trained with unrectified image dataset, the training unrectified/ video unrectified setup is used to compare the accuracy of SSD MobileNet v1, SSD Inception v2, Faster RCNN Inception v2 and YOLO v2. Tiny YOLO does not detect anything with this setup, thus YOLO v2 and Tiny YOLO are compared using training rectified/ video rectified setup as both of the detection systems are trained with rectified image dataset as well as unrectified image dataset.

Test flights that the target drone is moving in side and height direction, i.e. pure translations on x and y axis, are conducted in order to assess detection accuracy on simple movements. The average RMS errors in x, y and z directions are listed in the tables below. Table 4.8 and Table 4.9 lists the RMS errors when the flight is conducted with white background and Table 4.10 and Table 4.11 lists the RMS errors when the flight is conducted with complex background.

Table 4.8: Average RMS Errors of Side and Height Translation Flights with White Background, Training Unrectified/ Video Unrectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
SSD MobileNet v1	12.08	9.84	37.21	19.71
SSD Inception v2	11.43	8.00	23.57	14.33
Faster RCNN Inception v2	11.33	7.76	19.96	13.02
YOLO v2	12.37	6.48	19.35	12.73

It can be obtained that Faster RCNN Inception v2 has the lowest error in x direction, YOLO v2 has the lowest error in y and z direction. Overall, YOLO v2 has the lowest average error. This makes YOLO v2 to be the most accurate one in the 4 systems for the two test flights.

Table 4.9: Average RMS Errors of Side and Height Translation Flights with White Background, Training Rectified/ Video Rectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
YOLO v2	11.84	7.06	20.82	13.24
Tiny YOLO	13.18	5.64	32.19	17.00

Comparing YOLO v2 and Tiny YOLO, it can be obtained that Tiny YOLO has more side RMS error, depth RMS error and less height RMS error. Overall, YOLO v2 has less average RMS error.

Table 4.10: Average RMS Errors of Side and Height Translation Flights with Complex Background, Training Unrectified/ Video Unrectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
SSD MobileNet v1	11.69	15.93	18.88	15.50
SSD Inception v2	10.62	14.70	8.58	11.30
Faster RCNN Inception v2	16.81	9.14	32.41	19.45
YOLO v2	15.01	11.39	40.19	22.20

Table 4.11: Average RMS Errors of Side and Height Translation Flights with Complex Background, Training Rectified/ Video Rectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
YOLO v2	47.72	19.82	37.53	35.02
Tiny YOLO	14.96	10.15	44.15	23.09

From Table 4.10 and Table 4.11, it can be obtained that all these object detection systems gives similar detection results when the target UAV is flying with complex background. SSD MobileNet v1 and SSD Inception v2 has lower RMS errors compared to the flights with background due to their better accuracy in depth estimation. Faster RCNN Inception v2, YOLO v2 and Tiny YOLO have more error with complex background.

Test flights of a target flying over a white and complex background in depth z direction are conducted and processed by object detection systems. This flight pattern is used to test the robustness of the distance estimation. Unlike flights on side and height axis (only the locations of the bounding box change), when flying on depth axis the only change is the size of bounding boxes. The RMS errors are listed in the tables below.

Table 4.12: Average RMS Errors of Depth Translation Flights with White Background, Training Unrectified/ Video Unrectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
SSD MobileNet v1	12.86	6.86	30.44	16.72
SSD Inception v2	12.52	4.78	21.02	12.77
Faster RCNN Inception v2	11.61	5.42	17.23	11.42
YOLO v2	12.45	4.28	17.25	11.33

Table 4.13: Average RMS Errors of Depth Translation Flights with White Background, Training Rectified/ Video Rectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
YOLO v2	12.04	4.95	18.89	11.96
Tiny YOLO	11.80	3.75	20.27	11.94

From Table 4.12 and Table 4.13, it can be obtained that YOLO v2 is more accurate than any of the TensorFlow systems and have similar RMS error with Tiny YOLO when the UAV is flying over a white background. Also, it can be obtained that the average RMS error is less than pure translations on side and height axis. However, the difference in RMS errors of the object detection systems between the two flight patterns is 2.31 cm on average. This is not a substantial difference considering that the relative distances between the two UAVs during all test fights are usually more than 2 meters.

Table 4.14: Average RMS Errors of Depth Translation Flights with Complex Background, Training Unrectified/ Video Unrectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
SSD MobileNet v1	7.65	16.18	22.70	15.51
SSD Inception v2	7.50	13.10	17.15	12.58
Faster RCNN Inception v2	11.33	7.21	25.69	14.75
YOLO v2	15.17	10.20	37.67	21.01

Table 4.15: Average RMS Errors of Depth Translation Flights with Complex Background, Training Rectified/ Video Rectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
YOLO v2	39.73	24.05	41.10	34.96
Tiny YOLO	15.04	12.23	59.65	29.98

With complex background, SSD MobileNet v1 and SSD Inception 2 has improvement on side and depth distance estimation, but has more error in height estimation compared to flight with white background. All the other systems are less accurate with complex background. SSD Inception v2 has the least average RMS error. It can also be obtained that SSD Mobilnet v1, SSD Inception v2 and YOLO v2 have similar average RMS error with pure translations on side and height direction (0.01 cm difference on average for the three detection systems). Thus, the distance estimations on side, height and depth directions are robust because the detection results have similar RMS errors when the UAV is doing pure translations on side and height direction or depth direction.

Test flights with a target UAV doing pure rotation around y axis (yaw) is conducted to investigate the impact of yaw angle change (visible size to the camera change) to the detection results. The rotation angle has a maximum of 360 °. Detection results are listed in the tables below.

Table 4.16: Average RMS Errors of Pure Rotation Flights with White Background, Training Unrectified/ Video Unrectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
SSD MobileNet v1	16.38	7.02	31.43	18.28
SSD Inception v2	16.49	5.21	20.28	14.00
Faster RCNN Inception v2	15.31	4.92	19.13	13.12
YOLO v2	15.57	4.31	18.87	12.92

Table 4.17: Average RMS Errors of Pure Rotation Flights with White Background, Training Rectified/ Video Rectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
YOLO v2	15.88	4.70	16.91	12.50
Tiny YOLO	17.15	4.08	26.44	15.89

From Table 4.16 and Table 4.17, it can be obtained that YOLO v2 has the least RMS error on average. Notice that comparing to the test flights that the target is doing pure translations in side, depth and height direction, doing rotation does not have a substantial increase in error. Detailed comparison is detailed in Section 4.5.

Table 4.18: Average RMS Errors of Pure Rotation Flights with Complex Background, Training Unrectified/ Video Unrectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
SSD MobileNet v1	14.62	14.96	15.22	14.93
SSD Inception v2	13.70	11.96	19.45	15.04
Faster RCNN Inception v2	18.90	7.01	33.84	19.91
YOLO v2	20.84	9.35	37.28	22.49

Table 4.19: Average RMS Errors of Pure Rotation Flights with Complex Background, Training Rectified/ Video Rectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
YOLO v2	34.52	11.80	37.80	28.04
Tiny YOLO	20.74	9.20	39.02	22.99

When rotation flights are conducted over a complex background, SSD MobileNet v1 has less average RMS error while the rest of the systems has an increase of error. The comparison between the errors in pure rotation and pure translations are detailed in Section 4.5.

In order to mimic realistic flight patterns, complex flight motions are conducted. The flights include translations as well as rotations.

Table 4.20: Average RMS Errors of Complex Flights with White Background, Training Unrectified/ Video Unrectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
SSD MobileNet v1	14.69	5.67	28.91	16.42
SSD Inception v2	13.78	5.12	21.68	13.53
Faster RCNN Inception v2	13.76	5.94	18.35	12.68
YOLO v2	14.79	3.92	15.00	11.24

Table 4.21: Average RMS Errors of Complex Flights with White Background, Training Rectified/ Video Rectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
YOLO v2	14.46	5.43	14.96	11.62
Tiny YOLO	14.74	3.42	19.32	12.49

From Table 4.21 and Table 4.21, it can be obtained that with white background, YOLO v2 has the least average error. Compared to pure translations

with white curtain, SSD MobileNet v1 has a 9.23 % decrease in average RMS error, SSD Inception v2 has a 0.16 % increase in average RMS error, Faster RCNN Inception v2 has a 4.25 % increase in average RMS error, YOLO v2 has a 6.92 % decrease in average RMS error and Tiny YOLO has a 10.95 % decrease in average RMS error.

Compared to pure rotations with white curtain, SSD MobileNet v1 has a 10.14 % decrease in average RMS error, SSD Inception v2 has a 3.34 % decrease in average RMS error, Faster RCNN Inception v2 has a 3.33 % decrease in average RMS error, YOLO v2 has a 10.03 % decrease in average RMS error, and Tiny YOLO has a 21.38 % decrease in average RMS error.

Table 4.22: Average RMS Errors of Complex Flights with Complex Background, Training Unrectified/ Video Unrectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
SSD MobileNet v1	13.62	16.51	18.92	16.35
SSD Inception v2	13.18	19.17	30.49	20.95
Faster RCNN Inception v2	17.68	7.75	34.25	19.89
YOLO v2	15.89	7.06	52.38	25.11

Table 4.23: Average RMS Errors of Complex Flights with Complex Background, Training Rectified/ Video Rectified Setup

Detection System	**Average x RMS Error (cm)**	**Average y RMS Error (cm)**	**Average z RMS Error (cm)**	**Average RMS Error (cm)**
YOLO v2	44.59	29.99	44.68	39.75
Tiny YOLO	17.42	10.42	50.31	26.05

From Table 4.22 and Table 4.23, it can be obtained that with complex background, SSD MobileNet v1 has the least average error. Compared to pure translations with complex background, SSD MobileNet v1 has a 5.44 % decrease in average RMS error, SSD Inception v2 has a 75.93 % increase in average RMS error, Faster RCNN Inception v2 has a 18.58 % increase in

average error, YOLO v2 has a 14.96 % increase in average RMS error, and Tiny YOLO has a 1.37 % increase in average RMS error.

Compared to pure rotations with complex background, SSD MobileNet v1 has a 9.48 % increase in average RMS error, SSD Inception v2 has a 39.32 % increase in average RMS error, Faster RCNN Inception v2 has a 0.12 % decrease in average RMS error, YOLO v2 has an 26.71 % increase in average RMS error, and Tiny YOLO has a 13.33 % increase in average RMS error.

As a summary of the detection results, all the five object detection systems are working for flights with and without white curtain. Given that the detection results are accurate (with white background), the difference of average RMS errors between complex fight patterns and simply flight patterns are not substantial. In most cases, depth estimation has the most error compared to the errors in side and height distance estimation. This is because the camera and projection matrix is not precise as discussed in Section 4.6.0.1. The quality of the bounding box is crucial to the accuracy of distance estimation as well.

Table 4.24: Average RMS Errors of Object Detection Systems with Training Unrectified/ Video Unrectified Setup

Detection System	**Average RMS Error With White Curtain (cm)**	**Average RMS Error Without White Curtain (cm)**
SSD MobileNet v1	17.78	15.57
SSD Inception v2	13.66	14.97
Faster RCNN Inception v2	12.56	18.50
YOLO v2	12.50	22.70

Table 4.25: Average RMS Errors of Object Detection Systems with Training Rectified/ Video Rectified Setup

Detection System	**Average RMS Error With White Curtain (cm)**	**Average RMS Error Without White Curtain (cm)**
YOLO v2	12.33	34.44
Tiny YOLO	14.33	25.27

From Table 4.24 and Table 4.25, it can be obtained that Faster RCNN Inception v2 and YOLO v2 has the lowest average RMS error when the background has little noise (with white background) and SSD Inception v2 has the lowest RMS error when the background is complex. Putting all the test results with training unrectified/ video unrectified setup together, SSD MobileNet v1 has an average RMS error of 16.68 cm, SSD Inception v2 has an average RMS error of 14.32 cm, Faster RCNN Inception v2 has an average RMS error of 15.53 cm and YOLO v2 has an average RMS error of 17.61 cm. With training rectified/ video rectified setup, YOLO v2 has an average RMS error of 23.39 cm and Tiny YOLO has an average RMS error of 19.08 cm. Hence, SSD Inception v2 is the most accurate detection system followed closely by Faster RCNN Inception v2.

Notice that, the average errors are noticeable (from 12.33 cm up to 34.44 cm) due to the fact that the bounding boxes may not always be accurate. Figure 4.11 illustrates the best case scenario that the detected bounding boxes are accurate and tight around the target. Figure 4.12 and Figure 4.13 illustrates two types of situations that the bounding boxes are not accurate. The red bounding box represents a perfect bounding box. The green bounding box represents the real detection result. In Figure 4.12, it can be obtained that the actual bounding boxes are loose. Figure 4.13 illustrates bounding boxes are confused by the background. The detection results do not have the target inside the bounding box.

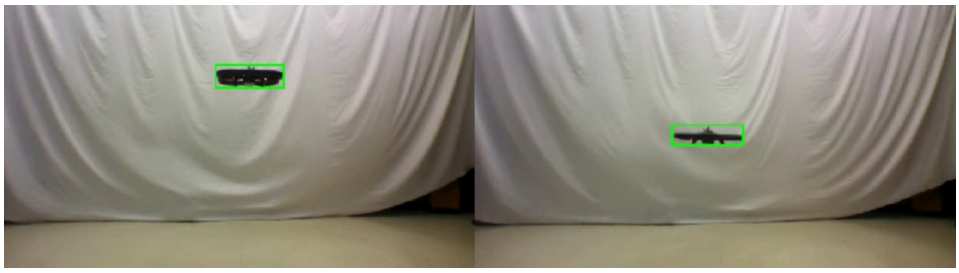

Figure 4.11: Examples of Accurate Bounding Box

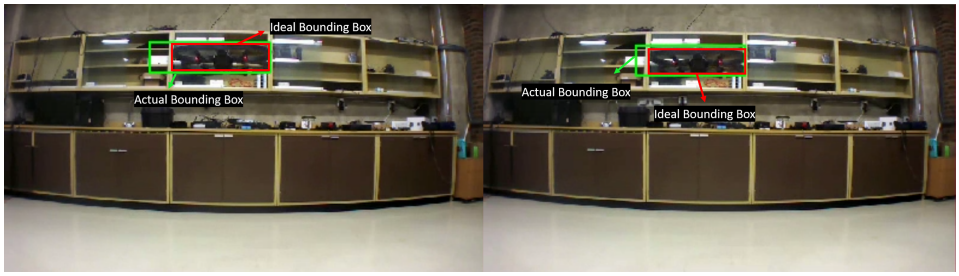

Figure 4.12: Examples of Loose Bounding Box

Figure 4.13: Examples of Wrong Bounding Box

Distribution graph of difference between detection results and ground-truth Vicon data can be used to represent the probability of the bounding box to be accurate. From Figure 4.14, it can be obtained that the probability distribution of the difference in detection results and ground-truth does not follow normal distribution. This implies that the bounding box is not always accurate. For instance, there are two peaks in the distribution on y axis (height direction), this implies that some detected bounding boxes does not have the target inside as illustrated in Figure 4.13. Distribution graphs for the rest of the test flights are shown in Appendix B.2.1.

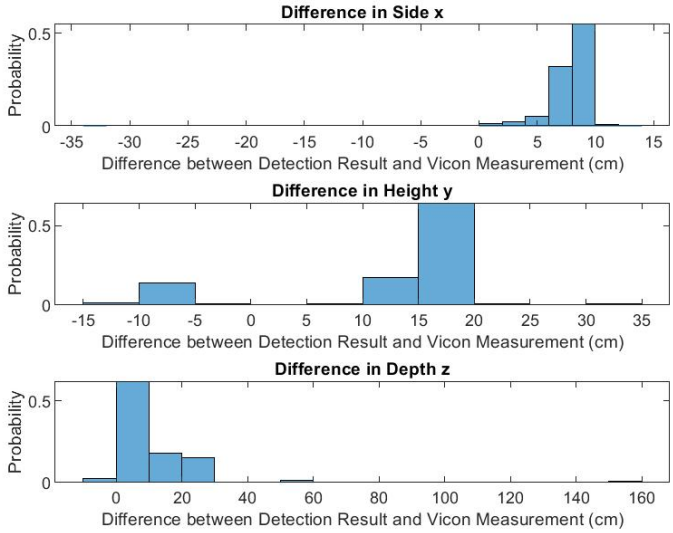

Figure 4.14: SSD MobileNet v1 Pure Translation in Side and Height Direction Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

4.3.3 Consistency Results Discussion

In this Section, consistency of the object detection systems in different experimental trials are evaluated. The average precision (AP) introduced in Section 2.4.2.3 is used as a metric of consistency. IoU threshold settings for all of the detection systems are set to 0.5. This is the default of threshold of SSD MobileNet v1, SSD Inception v2 and Faster RCNN v2. The default thresholds of YOLO v2 and Tiny YOLO are 0.2, and these were adjusted to 0.5 to make the comparison fair. mAP of the object detection systems are shown in Table 4.26. It can be obtained that Faster RCNN Inception v2 is the most consistent system having the highest mAP of 0.98.

Table 4.26: Overall mAP of Tested Object Detection Systems

Object Detection System	**mAP**
SSD MobileNet v1	0.71
SSD Inception v2	0.64
Faster RCNN Inception v2	0.98
YOLO v2	0.83
Tiny YOLO	0.38

4.4 Further Discussion

In this Section, overall performance of the object detection systems is evaluated considering efficiency, accuracy and consistency and a decision matrix is made to assess the object detection systems. Efficiency has the least weight (0.2) as it can be optimized by coding, i.e. develop a ROS package for TensorFlow in C language. Accuracy and consistency are related to the architecture of object detection systems and training dataset. They can be improved by training with better image datasets, i.e. more representative images. However, further training requires huge amount of computational power and the detection results may suffer from overfitting problem. Hence, accuracy (0.4) and consistency (0.4) have more weights as they are harder to improve comparing to efficiency.

Efficiency is scored based on running speed. The higher fps the better. In this thesis, only speed in ROS is compared for efficiency scoring because the real-time tasks are performed in ROS to communicate with observing UAV. Take 100 fps to be score 10, and thus SSD MobileNet v1 scores 0.27, SSD Inception v2 scores 0.20. Faster RCNN scores 0.20, YOLO v2 scores 6.73 and Tiny YOLO scores 7.38.

Accuracy is scored based on the average RMS error. The lower error the better. Training unrectified/ video unrectified setup are used to compare and Tiny YOLO is scaled with respect to YOLO v2 to be comparable in this setup. For instance, YOLO v2 has an average RMS error of 17.61 cm with training unrectified/ video unrectified setup and 23.39 cm with training rectified / video rectified setup. Tiny YOLO has an average RMS error of 19.08 cm with training rectified / video rectified setup and scaled to 19.08/23.39*17.61 = 14.37 cm error with training unrectified/ video unrectified setup. Define that 0 cm error scores 10 and 30 cm error scores 0, SSD MobileNet v1 scores 4.44, SSD Inception v2 scores 5.23, Faster RCNN Incepion v2 scores 4.82, YOLO v2 scores 4.13 and Tiny YOLO scores 5.21.

Consistency is scored based on mAP values. The higher mAP the better. Take mAP of 1 to score 10 and 0 to score 0, SSD MobileNet v1 scores 7.06,

SSD Inception v2 scores 6.44, Faster RCNN Inception v2 scores 9.76, YOLO v2 scores 8.28 and Tiny YOLO scores 3.78.

Table 4.27: Decision Matrix of Selecting an Object Detection API

Object Detection API	**Efficiency (0.2)**	**Accuracy (0.4)**	**Consistency (0.4)**	**Score**
SSD MobileNet v1	0.27	4.44	7.06	4.65
SSD Inception v2	0.20	5.23	6.44	4.71
Faster RCNN v2	0.20	4.82	9.76	5.87
YOLO v2	6.73	4.13	8.28	6.31
Tiny YOLO	7.38	5.21	3.78	5.07

From decision matrix Table 4.27, it can be obtained that YOLO v2 is the best object detection system considering efficiency, accuracy and consistency. Thus, YOLO v2 is suggested in this thesis to be the object detection system used for UAV tracking project.

4.5 Assumption Assessment

Calculations of the relative distance using bounding box information are restricted by three assumptions discussed in Section 2.5. The first and second assumptions, assuming object detection system to be precise and the physical dimension to be known, are straight forward. In this section, the third assumption, the impact of visible size change of UAV is assessed. As discussed in Section 4.4, YOLO v2 has the best overall performance and suggested to be the object detection system used for control system in the future plan of UAV tracking project. Validation of the third assumption is made based on the detection results from YOLO v2 with training unrectified/ video unrectified setup flying over white background.

In Section 4.3.2.4, YOLO v2's RMS errors of the flights that target UAV is doing pure translations and pure rotations are listed. Comparing the RMS errors of pure translations and pure rotation, there is 25.48 % increase in side, 16.41 % decrease in height and 3.46 % decrease in depth. Overall, there is a 7.75 % increase in average RMS error. Yaw angle changes has the largest im-

pact on side and height distance estimation and does not affect depth distance estimation by much.

The change of roll and pitch angle is conducted by the Parrot ARDrone 2.0 to provide thrust so that it can do translation movements in side and depth direction. The larger the acceleration, the larger angle change in roll and pitch.

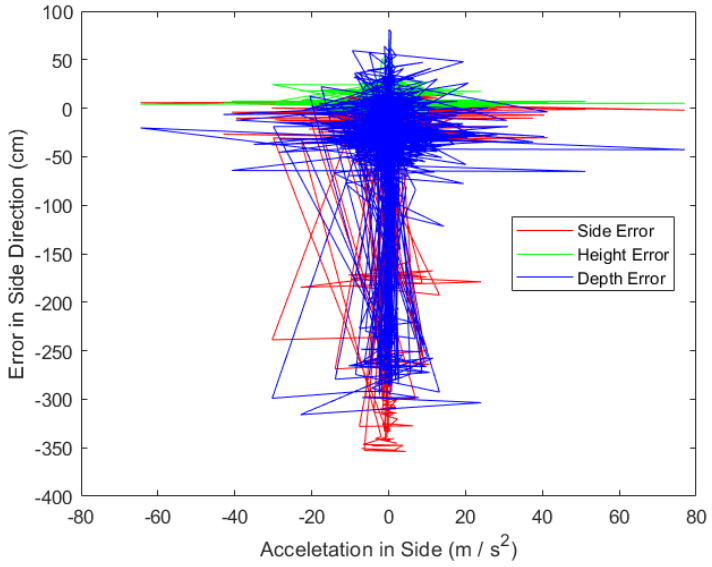

Figure 4.15: Error Change with Acceleration in Side

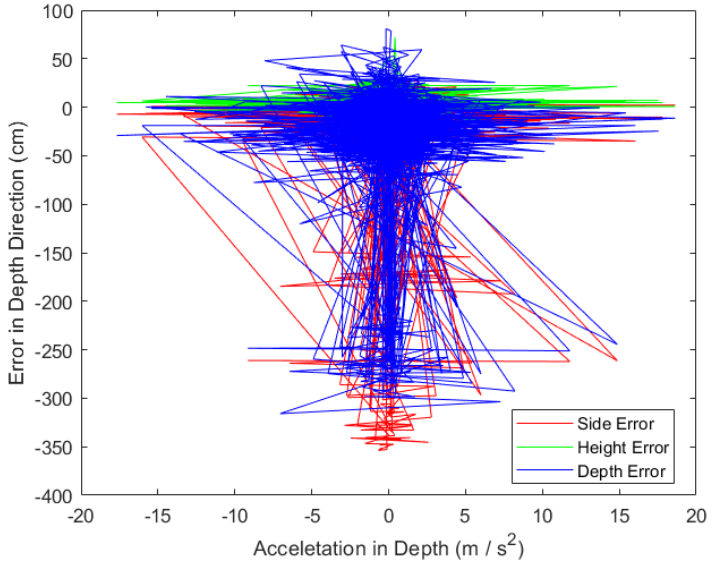

Figure 4.16: Error Change with Acceleration in Depth

Figure 4.15 and Figure 4.16 illustrates the error change with respect to acceleration change in side and depth direction. Although higher acceleration has larger errors, similar errors occur when the acceleration is low. Overall, the error caused by visible size change of the target UAV is acceptable. Hence, assumption 3 is reasonable.

4.6 Uncertainty Analysis

In this Section, uncertainties in the experiments are assessed. As discussed in Section 2.7, two sorts of uncertainties are measured, namely bias uncertain and random uncertainty. Tests for analyzing uncertainty in bounding box coordinate output is conducted by feeding the same image to object detection systems for 100 times. All the detection results are exactly the same, thus the uncertainty in bounding box coordinate output is 0. Bias uncertainty in UAV dimension measurement is 0.01 cm because the minimum scale on the ruler is 0.1 cm. Detailed calculation on uncertainty in Vicon system measurements, camera calibration and distance estimation are discussed in the sections below.

4.6.0.1 Uncertainty in Vicon Camera System

In this Section, the uncertainty in Vicon camera system measurements are investigated. The uncertainties are measured in three directions, side x, height y and depth z. Experiments are conducted by placing a UAV at a known location and then move to a known distance in one of the x, y, and z directions. Vicon data at the original location and final location is recorded to calculate the distance moved in Vicon measurements. 331 samples were taken for each of the experiments. Probability distribution of the measurements are illustrated in Figure 4.17, Figure 4.18 and Figure 4.19.

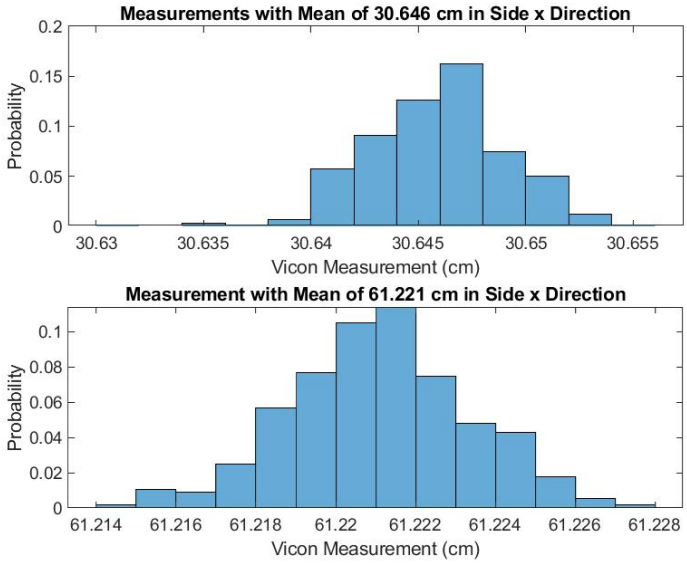

Figure 4.17: Measurement Probability Distribution in Side Direction

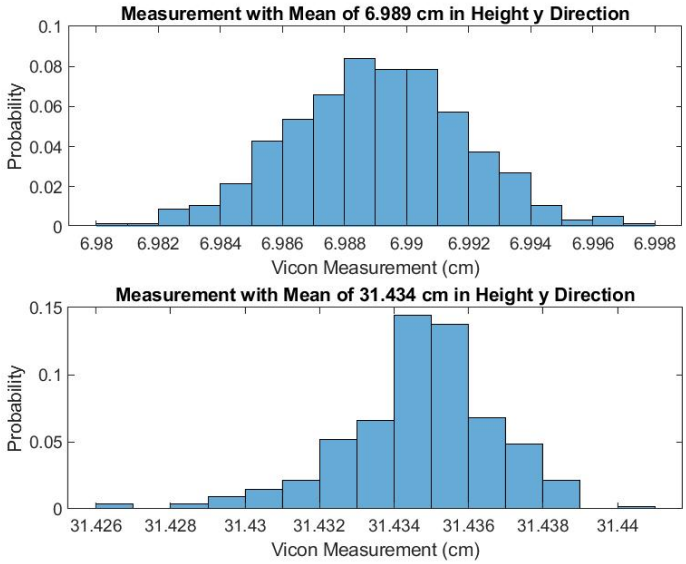

Figure 4.18: Measurement Probability Distribution in Height Direction

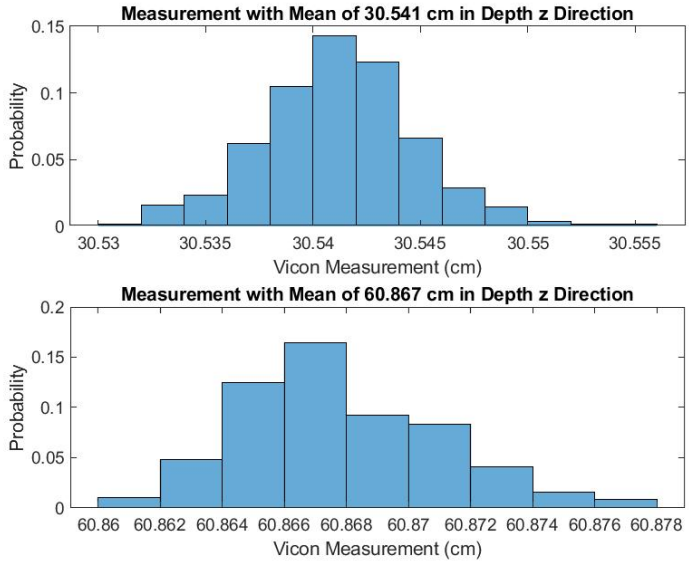

Figure 4.19: Measurement Probability Distribution in Depth Direction

From Figure 4.17, Figure 4.18 and Figure 4.19, it can be obtained that the probability distributions are close to normal distribution. The normality test parameters of the samples are listed in Table 4.28.

Table 4.28: Normality Test Vicon System Measurements

Mean of Test Distance	**Excess Kurtosis**	**Skewness**
30.646 cm in x	0.81	-0.27
61.221 cm in x	-0.06	0
6.989 cm in y	0.10	-0.04
31.434 cm in y	1.42	-0.70
30.541 cm in z	0.68	0.23
60.867 cm in z	-0.07	0.53

Kurtosis measures the peakedness and skewness measures the asymmetry of a distribution. In this thesis, excess kurtosis is used to measure peakedness. Excess kurtosis is simply kurtosis value subtract by 3. Ideally, for a perfect normal distribution, both excess kurtosis and skewness value should be 0 [79]. From Table 4.28, it can be obtained that the absolute values of excess kurtosis is less than 7 and absolute value of skewness is less than 2 in all the experiments. Since the samples size is larger than 300, samples in all the experiments

follows normal distribution [79]. $z*$ equals to 1.96 for 95% confidence interval for a normal distribution [55]. Using equation 2.11, the uncertainty with 95 % confidence interval can be calculated as:

Table 4.29: Uncertainty in Vicon System Measurements

Mean of Test Distance	**Uncertainty (cm)**
30.646 cm in x	0.004
61.221 cm in x	0.002
6.989 cm in y	0.004
31.434 cm in y	0.003
30.541 cm in z	0.003
60.867 cm in z	0.002

From Table 4.29 it can be obtained that the uncertainties in Vicon system measurement are minor and can be neglected. There is no bias uncertainty in Vicon measurement because all the data are generated from computer.

4.6.0.2 Uncertainty in Camera Calibration

As discussed in Section 3.2.3.1, there are two hardware limitations that restrict the performance of camera calibration: low camera resolution in ROS and inaccurate checkerboard size input. In order to quantify the uncertainty in camera calibration, 12 trials of camera calibration process is conducted and the camera calibration file which gives the best rectification on the image by eye observation is shown in Section 3.2.3.1. The other 11 calibration files can be seen in Appendix B.3. All the 12 calibration files give similar but different camera matrices and projection matrices. All these tests are provided with enough calibration data points, i.e. the x, y, size and skew are all green in Figure 3.17. Only random uncertainty is discussed for camera calibration. This is because there is no measurement reading by human, all calibration data is measured and calculated by computer. The probability distribution of the camera and projection matrices are plotted as shown in Figure 4.20 and Figure 4.21.

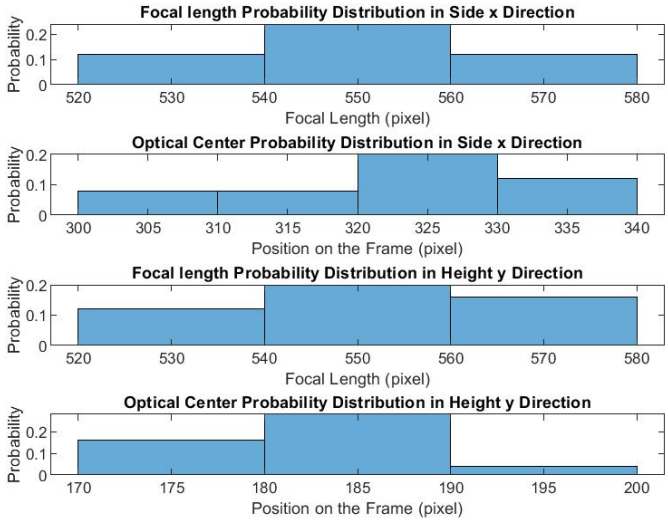

Figure 4.20: Camera Matrix Parameters Probability Distribution

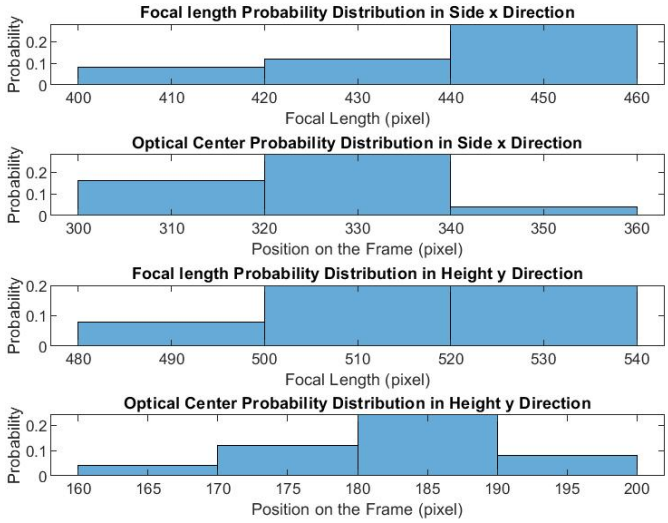

Figure 4.21: Projection Matrix Parameters Probability Distribution

Table 4.30 and Table 4.31 gives normality test parameters for the camera matrices and projection matrices in calibration files. Excess kurtosis, skewness and standard error (SE) are generated by excel.

Table 4.30: Normality Test for Parameters in Camera Matrices

Parameter in Camera Matrix	**Excess Kurtosis/ SE**	**Skewness/ SE**
f_x	-0.05	-0.21
c_x	-0.21	-0.25
f_y	-0.08	-0.21
c_y	0.51	-0.26

Table 4.31: Normality Test for Parameters in Projection Matrices

Parameter in Camera Matrix	**Excess Kurtosis/ SE**	**Skewness/ SE**
f_x	-0.08	-0.24
c_x	-0.11	-0.09
f_y	-0.01	-0.19
c_y	0.27	-0.28

The normality test is conducted by dividing excess kurtosis and skewness value by standard error. For small samples, i.e. sample amount less than 50, excess kurtosis and skewness value divided by standard error should not exceed 1.96 for a normal distribution with 95 % confidence interval [79]. From Table 4.30 and Table 4.31, it can be obtained that the absolute value of excess kurtosis and skewness value divided by standard error for parameters in both camera matrices and projection matrices are less than 1.96. Hence, the sample data are assumed to follow normal distribution.

Since there are only 12 sets of calibration data point, student's t-distribution is used to calculate uncertainty instead of normal distribution. For sample size of 12 and confidence interval of 95 % (two-sided), degree of freedom is 11 and t value is 2.201 [59]. Using equation 2.12, the uncertainties of parameters in camera and projection matrices are:

Table 4.32: Uncertainty in Parameters in Camera Matrices

Parameter in Camera Matrix	**Uncertainty (pixel)**
f_x	8
c_x	6
f_y	8
c_y	4

Table 4.33: Uncertainty in Parameters in Projection Matrices

Parameter in Projection Matrix	**Uncertainty (pixel)**
f_x	8
c_x	8
f_y	8
c_y	4

It can be obtained that the uncertainties in the camera matrices are 1.44 % for f_x, 1.81 % for c_x, 1.44 % for f_y and 2.06 % for c_y. The uncertainties in the projection matrices are 1.83 % for f_x, 2.47 % for c_x, 1.54 % for f_y and 2.33 % for c_y. Although the calibration matrices are not perfect, i.e. camera and projection matrices differ for every calibration, the uncertainty is not substantial.

4.6.0.3 Uncertainty in Distance Estimation from Bounding Box

As discussed in Section 2.7, the uncertainties in side, height and depth distance estimation calculated by Monte Carlo method. The Matlab code used for this simulation is shown in Appendix A.5. The normality of f_x, f_y, c_x and c_y in camera matrices and projection matrices are proved in Section 4.6.0.2 and thus 1000 random samples of f_x, f_y, c_x and c_y following normal distribution are taken. 1000 random simulated measurements of height and width are taken following normal distribution with standard deviation of 0.01 cm, mean of 12.59 cm for height and standard deviation of 0.01 cm, mean of 50.65 cm for width. One of the bounding boxes with coordinates ($x^{'}_{min}$, $y^{'}_{min}$, $x^{'}_{max}$, $y^{'}_{max}$) of (359, 99, 490, 127) from YOLO v2 is used for calculations. The simulations are conducted for detection on both unrectified and rectified videos. The normality test on the results are listed in Table 4.34.

Table 4.34: Normality Test for Distance Estimation from Bounding Box

Distance Direction/ Video Calibration	**Excess Kurtosis**	**Skewness**
Side/ Video Unrectified	-0.02	-0.09
Side/ Video Rectified	-0.10	0.07
Depth/ Video Unrectified	-0.20	0.04
Depth/ Video Rectified	0.09	0.06
Height/ Video Unrectified	-0.03	0.09
Height/ Video Rectified	0.01	-0.12

From Table 4.34, it can be obtained that the simulated data follows normal distribution because all of the absolute values of excess kurtosis is less than 7 and absolute value of skewness is less than 2 for 1000 samples [79]. Remember that $z*$ equals to 1.96 for 95% confidence interval for a normal distribution [55]. For 95 % confidence interval, using equation 2.11, the mean value of the random samples and uncertainties can be calculated:

Table 4.35: Uncertainty in Distance Estimation from Bounding Box

Distance Direction/ Video Calibration	**Mean (cm)**	**Uncertainty (cm)**
Side/ Video Unrectified	-46.5	0.2
Side/ Video Rectified	-46.4	0.4
Depth/ Video Unrectified	230.7	0.2
Depth/ Video Rectified	200.7	0.2
Height/ Video Unrectified	29.4	0.2
Height/ Video Rectified	27.7	0.2

From Table 4.35, it can be obtained that the uncertainties in distance estimations are not substantial.

Notice that the uncertainty is a function of $x^{'}_{min}$, $y^{'}_{min}$, $x^{'}_{max}$, $y^{'}_{max}$ and will change when the bounding box information changes. Due to the fact that $x^{'}_{min}$, $y^{'}_{min}$, $x^{'}_{max}$, $y^{'}_{max}$ are random numbers from object detection systems, it is tedious to quantify uncertainties for all the bounding boxes knowing that the uncertainties in distance estimations are minor.

Chapter 5

Conclusions and Future Work

5.1 Summary

This thesis has tested and benchmarked deep learning object detection systems in both Linux (Ubuntu 16.04) and ROS environment, namely SSD MobileNet v1, SSD Inception V2 and Faster RCNN Inception v2 for TensorFlow as well as YOLO v2 and Tiny TOLO for Darknet to propose a feasible solution to real-time UAV detection. The training process is done offline and detection is done by using the Parrot ARDrone 2.0's onboard camera.

Uncertainties involved in experiments and measurements are also discussed. Bias error in this thesis comes from the measurements on the dimensions of Parrot ARDrone 2.0. Due to the limitation of the camera resolution and the checkerboard quality, the camera calibration is not precise causing random error in calibration process. Another source of random error is the distance measurements by Vicon camera system.

Efficiency, accuracy and consistency of each detection system are compared, YOLO v2 turns out to be to best all-around system for real-time UAV tracking project.

5.2 Limitations of the Work

This thesis demonstrated a monocular camera-based object detection solution for UAV tracking purposes. However, there are still limitations:

- The use of the GTX 1080 Ti GPU constrains the mobility of the design,

i.e. at this stage, tests can only be run offboard the UAV in a lab environment. A potential substitute for the GTX 1080 Ti is the Nvidia Jetson TX2. This is a low-power single-board form factor GPU. Carrying a Jetson TX2 onboard the UAV could give it the capability to accomplish tracking tasks without the aid of a desktop-sized computer console.

- YOLO v2 turns out to be the object detection system with the best all-around performance. However, it comes with compromises in accuracy and consistency. Faster RCNN Inception v2 is better than YOLO v2 in accuracy and consistency, but its speed is unusable.

- Due to the time limit and resources available for this thesis, the quality of training is sub-optimal. The current training results work well enough for the purpose of this thesis, which was to select an object detection system for vison-based UAV tracking. However better performance can be expected if more images were used to train the object detection APIs.

- Parrot ARDrone's onboard camera can only stream videos at a resolution of 640 * 360. The video quality is not good enough for precise camera calibration and object Detection.

- The checkerboard used for calibration is not ideal. The dimensions of the blocks are not exactly 3 cm x 3 cm. Also, printing a checkerboard on a piece of a paper and attach it to a cardboard causes distortion on the presentation of the checkerboard to the camera.

5.3 Future Work

The following work could be performed in order to obtain a better UAV detection system:

- The computational power of the lab computer only allows TensorFlow-based APIs to be trained starting from pre-trained model weights. A fully customized training could be conducted if more powerful GPUs or a cloud computing service were used.

- Due to the complex background and movement, it is inevitable for the object detection system to periodically lose the target. To compensate these "blank frames", a tracker such as the Real-time Recurrent Regression Networks (Re3) could be used [80]. One of the merits of this tracker is that it offers the ability to lock in a target at the first frame of detection. This could potentially eliminate false target tracking by the system.

- A Kalman Filter could be used to reduce the computational power required. Bounding box locations in the next few frames can be predicted by a Kalman Filter so that object detection systems can have lower running speed.

- Mobile version of processing chips could be potentially used for object detection assuming the detection system was efficient enough and compatible with the drone's operating system. This would greatly improve the mobility of the UAV object tracking system.

- Using a lager sized checkerboard with better quality, i.e. printed on a plain hard surface, can also improve the calibration result.

- A distance sensor or a camera with higher resolution can be used to replace the onboard camera in order to improve the distance estimation on the depth direction.

Bibliography

[1] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[2] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

[3] Lichao Chen, Sudhir Singh, Thomas Kailath, and Vwani Roychowdhury. Brain-inspired automated visual object discovery and detection. *Proceedings of the National Academy of Sciences*, 116(1):96–105, 2019.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] Pablo Martinez Rodriguez. Vision-based algorithms for uav mimicking control system. Master of science, University of Alberta, 2017.

[6] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, July 2017.

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[8] Jeff Hale. Deep learning framework power scores 2018. https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a, 2018 (accessed February 20, 2019).

[9] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[10] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[11] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[12] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[13] Rina Dechter. Learning while searching in constraint-satisfaction-problems. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, pages 178–183. AAAI Press, 1986.

[14] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[15] Michael Irwin Jordan. *Learning in graphical models*, volume 89. Springer Science & Business Media, 1998.

[16] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[17] Jungang Xu, Hui Li, and Shilong Zhou. An overview of deep generative models. *IETE Technical Review*, 32(2):131–139, 2015.

[18] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Ad-*

vances in Neural Information Processing Systems, pages 841–848. MIT Press, 2002.

[19] Roland Memisevic. An introduction to structured discriminative learning. Technical report, Technical report, University of Toronto, Toronto, Canada, 2006.

[20] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of machine learning research*, 2(Dec):265–292, 2001.

[21] Li Deng. Three classes of deep learning architectures and their applications: a tutorial survey. *APSIPA transactions on signal and information processing*, 2012.

[22] TensorFlow. Tensorflow detection model zoo. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md`, 2018 (accessed December 12, 2018).

[23] Joseph Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.

[24] K. He, G. Gkioxari, P. Dollr, and R. Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, Oct 2017.

[25] Nvidia. *NVIDIA TESLA P100 GPU ACCELERACTOR*, 2016.

[26] Ryan Martin. *Gigabyte Aorus GTX 1080 Ti Xtreme Edition 11GB Review*, 2017.

[27] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[28] TensorFlow. Tensorflow guide. `https://www.tensorflow.org/guide/`, 2018 (accessed October 29, 2018).

[29] Farooq Azam. *Biologically inspired modular neural networks*. PhD thesis, Virginia Tech, 2000.

[30] Kumar Shridhar. *A Beginners Guide to Deep Learning*, 2017.

[31] Andreas Zell. *Simulation neuronaler netze*, volume 1. Addison-Wesley Bonn, 1994.

[32] Standford University. Cs231n: Convolutional neural networks for visual recognition, 2018 (accesed on December 12, 2018).

[33] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[34] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Proceedings of the 20th International Conference on Artificial Neural Networks: Part III*, ICANN'10, pages 92–101, Berlin, Heidelberg, 2010. Springer-Verlag.

[35] Md Atiqur Rahman and Yang Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In *International symposium on visual computing*, pages 234–244. Springer, 2016.

[36] Mu Zhu. Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo*, 2:30, 2004.

[37] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[38] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmen-

tation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[39] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[40] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, June 2016.

[41] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Proceedings of the European Conference on Computer Vision (ECCV) (2016)*, 2016. To appear.

[42] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[44] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. *arXiv e-prints*, page arXiv:1312.4400, December 2013.

[45] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, June 2016.

[46] Kourosh Khoshelham and Sander Oude Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012.

[47] Joel Johnson. *Parrot AR Drone 2.0 Review: Your Own Private Predator*, 2012.

[48] OpenCV Organization. Camera calibration and 3d reconstruction. `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`, 2019 (accessed February 25, 2019).

[49] D. C. Brown. Decentering Distortion of Lenses. *Photometric Engineering*, 32(3):444–462, 1966.

[50] docs.ros.org. sensor_msgs/camerainfo message. `http://docs.ros.org/api/sensor_msgs/html/msg/CameraInfo.html`, 2019 (accessed February 25, 2019).

[51] IEC BIPM, ILAC IFCC, IUPAC ISO, and OIML IUPAP. Evaluation of measurement dataan introduction to the guide to the expression of uncertainty in measurement and related documents. joint committee for guides in metrology, jcgm 104: 2009 (2009).

[52] George W Brown. Standard deviation, standard error: Which'standard'should we use? *American Journal of Diseases of Children*, 136(10):937–941, 1982.

[53] Yale University. Confidence intervals. `http://www.stat.yale.edu/Courses/1997-98/101/confint.htm`, 2019 (accessed February 12, 2019).

[54] A.J. Hughes and D.E. Grawoig. *Statistics, a foundation for analysis.* Business and Economics Series. Addison-Wesley Pub. Co., 1971.

[55] Consumer Dummies. How to use the z-table. `https://www.dummies.com/education/math/statistics/how-to-use-the-z-table/`, 2019 (accessed March 19, 2019).

[56] Deborah J Rumsey. *U Can: statistics for dummies*. John Wiley & Sons, 2015.

[57] Richard G Brereton. The t-distribution and its relationship to the normal distribution. *Journal of Chemometrics*, 29(9):481–483, 2015.

[58] David Pollard. Statistics 101106 lecture 7. `http://www.stat.yale.edu/~pollard/Courses/100.fall98/pollard/lecture7.pdf`, 1998 (accessed March 19, 2019).

[59] easycalculation.com. T table. `https://www.easycalculation.com/statistics/t-distribution-critical-value-table.php`, 2019 (accessed March 19, 2019).

[60] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[61] Vicon. Vantage cutting edge, flagship camera with intlligent feedback and resolution. `https://www.vicon.com/products/camera-systems/vantage`, 2018 (accessed December 12, 2018).

[62] Vicon. Vicon vero - offering the best resolution, speed and price on the market. `https://www.vicon.com/products/camera-systems/vero`, 2018 (accessed December 12, 2018).

[63] Vicon. Vicon vero family. `https://www.vicon.com/products/camera-systems/vero`, 2018 (accessed December 12, 2018).

[64] Intel. Intel core i9 processors. `https://www.intel.ca/content/www/ca/en/products/processors/core/i9-processors.html`, 2019 (accessed February 25, 2019).

[65] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[66] Nvidia. The power of geforce gtx 1060. https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1060/, 2019 (accessed February 25, 2019).

[67] Parrot. *Parrot ARDrone 2.0 User Guide*, 2012.

[68] Jason M.O'Kane. *A Gentle Introduction to ROS*. CreateSpace Independent Publishing Platform, 2013.

[69] ROS.org. Ros kinetic installation instructions. http://wiki.ros.org/kinetic/Installation, 2019 (accessed February 25, 2019).

[70] Nvidia. Cuda zone. https://developer.nvidia.com/cuda-zone, 2018 (accessed October 29, 2018).

[71] Vicon. Tracker fast, precise object tracking: even from a single camera. https://www.vicon.com/products/software/tracker, 2018 (accessed October 29, 2018).

[72] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. Real-time computer vision with opencv. *Communications of the ACM*, 55(6):61–69, 2012.

[73] Mathworks. Train a cascade object detector. https://www.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html, 2019 (accessed February 25, 2019).

[74] Ammar Sameer Anaz and Diyaa Mehadi Faris. Comparison between open cv and matlab performance in real time applications. *AL Rafdain Engineering Journal*, 23(4):183–190, 2015.

[75] Nabil Mikati. Dependence of lead time on batch size studied by a system dynamics model. *International Journal of Production Research*, 48(18):5523–5532, 2010.

[76] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco:

Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[77] Nvidia. Nvidia jetson systems. `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/`, 2019 (accessed February 25, 2019).

[78] T. Chai and R. R. Draxler. Root mean square error (rmse) or mean absolute error (mae)?- arguments against avoiding rmse in the literature. *Geoscientific Model Development*, 7(3):1247–1250, 2014.

[79] Hae-Young Kim. Statistical notes for clinical researchers: assessing normal distribution (2) using skewness and kurtosis. *Restorative dentistry & endodontics*, 38(1):52–54, 2013.

[80] D. Gordon, A. Farhadi, and D. Fox. Re^3: Re al-time recurrent regression networks for visual tracking of generic objects. *IEEE Robotics and Automation Letters*, 3(2):788–795, April 2018.

Appendices

Appendix A

Code Developed

A.1 ROS Wrapper for TensorFlow

In this section, a ROS wrapper developed for TensorFlow is presented. This code is written in Python and enables TensorFlow object detection API to run in ROS environment.

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

import rospy
...

class ObjectDetection():
       def __init__(self):
        rospy.init_node('object_detection')
        rospy.on_shutdown(self.shutdown)
        ...

        self._sub = rospy.Subscriber(...)
        self._pub = rospy.Publisher(...)
        ...

       def callback(...):
        ...

       def shutdown(self):
              rospy.loginfo("Stopping object detection...")
              rospy.sleep(1)

if __name__ == '__main__':
       try:
```

```
            ObjectDetection()
            rospy.spin()
        except rospy.ROSInterruptException:
            rospy.loginfo("ROS object detection has started."
                ↪ )
```

A.2 Bounding Box Information Retrieval for TensorFlow

This section includes the Python code for object detection API to retrieve bounding boxes and their corresponding time stamp, score and class. Bounding boxes are used for 3D location estimation, time stamp is used to calculate the running speed, score is store to verify the bounding box qualifies the threshold of 0.5 and class is always ardrone in this thesis as there is only one target to detection.

```
import copy
import csv
import time
...

#Get Video Path
global_path = '...'
paths = ['...']
video_names = ['...']
...

            #Collect Data
            output_boxes = []
            output_scores = []
            output_classes = []
            t_to_array = []
            temp_t = time.clock()
            t_to_array = np.append(t_to_array,temp_t)
            t_to_array = np.append(t_to_array,fps)
            for i,box in enumerate(np.squeeze(boxes)):
                if(np.squeeze(scores)[i] > 0.5): #threshhold for
                    ↪ box displayed default at 0.5
                    temp_box = copy.deepcopy(np.squeeze(boxes)[i])
                    temp_score = copy.deepcopy(np.squeeze(scores)[
                        ↪ i])
```

```
                temp_class = copy.deepcopy(np.squeeze(classes)
                    ↪ [i])
                #temp_t = time.clock()

                output_boxes = np.append(output_boxes,temp_box
                    ↪ )
                output_scores = np.append(output_scores,
                    ↪ temp_score)
                output_classes = np.append(output_classes,
                    ↪ temp_class)

            #Record Data
            location_path = "/home/bingshen/Desktop/Data/"+
                ↪ sub_name+"_time.csv"
            olocation = open(location_path,"ab")
            wlocation = csv.writer(olocation, delimiter=",")
            wlocation.writerow(t_to_array)
            olocation.close()
            location_path = "/home/bingshen/Desktop/Data/"+
                ↪ sub_name+"_boxlocation.csv"
            olocation = open(location_path,"ab")
            wlocation = csv.writer(olocation, delimiter=",")
            wlocation.writerow(output_boxes)
            olocation.close()
            score_path = "/home/bingshen/Desktop/Data/"+sub_name+
                ↪ "_boxscore.csv"
            oscore = open(score_path,"ab")
            wscore = csv.writer(oscore, delimiter=",")
            wscore.writerow(output_scores)
            oscore.close()
            class_path = "/home/bingshen/Desktop/Data/"+sub_name+
                ↪ "_boxclass.csv"
            oclass = open(class_path,"ab")
            wclass = csv.writer(oclass, delimiter=",")
            wclass.writerow(output_classes)
            oclass.close()
            ...
```

A.3 3D Location Estimation from Bounding Box

In this section, Matlab code for estimation of 3D relative position of a target drone and a tracking drone from bounding box information is presented. In this code, root mean square error between the estimation and ground-truth are calculated and visualized.

```
clear;
clc;

%measured coeffcients
Drone_height = 12.59;
Drone_width = 50.65;
f_xk = 568.0748474282261;
f_yk = 568.7139402113011;
c_xk = 330.5770829093369;
c_yk = 193.6023640202218;

f_xp = 449.2703857421875;
f_yp = 534.8646850585938;
c_xp = 344.3868504957354;
c_yp = 194.3035218322311;

frame_height = 360;
frame_width = 640;


test_trail = 0;
bg = 0;
cc = 0;
tc = 0;

basead = 'C:\Users\Miller\Desktop\Data\';
framework = {'TensorFlow';'Darknet'};
[mf,nf] = size(framework);
TFapi = {'SSD_MobileNet';'SSD_Inception';'Faster_RCNN_Inception'
    ↪ };
[mtf,ntf] = size(TFapi);
DKapi = {'YOLO';'Tiny_YOLO'};
[mDK,nDK] = size(DKapi);
background = {'With_White_Curtain';'Without_White_Curtain'};
[mbackground,nbackground] = size(background);
```

```
traincc = {'train_unrect';'train_rect'};
[mtraincc,ntraincc] = size(traincc);
videocc = {'unrect';'rect'};
[mvideo,nvideo] = size(videocc);
test = {'test_1_pure_translation_take_1';'
    ↪ test_1_pure_translation_take_2';'
    ↪ test_2_back_and_forth_take_1';'
    ↪ test_2_back_and_forth_take_2';'
    ↪ test_3_rotation_take_1_realistic_180';'
    ↪ test_3_rotation_take_2_exteme_with_360';'
    ↪ test_4_random_flight_take_1';'test_4_random_flight_take_2'
    ↪ };
[mtest,ntest] = size(test);
count_xls = 0;
for f = 1:mf
    if (strcmp(framework(f,:),'TensorFlow'))
        for tf = 1:mtf
            for tt = 1:mtest
                for bk = 1:mbackground
                    for vc = 1:mvideo
                        vicon_path = string(strcat(basead,'
                            ↪ Vicon_Data\',background(bk,:),'\',
                            ↪ test(tt,:),'\vicon_data.txt'));
                        vicon = csvread(vicon_path);

                        %initiate variables
                        height = 0;
                        width = 0;
                        depth_from_height = 0;
                        depth_from_width = 0;
                        depth_Z = 0;
                        Xmid = 0;
                        Ymid = 0;
                        side_X = 0;
                        height_Y = 0;

                        detect_path = string(strcat(basead,
                            ↪ framework(f,:),'\',TFapi(tf,:),'\',
                            ↪ background(bk,:),'\',test(tt,:),'_',
                            ↪ videocc(vc,:),'_','boxlocation.csv')
                            ↪ );
                        result_name = string(strcat(framework(f,:)
                            ↪ ,'_',TFapi(tf,:),'_',background(bk
                            ↪ ,:),'_',test(tt,:),'_',videocc(vc,:)
                            ↪ ));
```

```
detect = csvread(detect_path);
detect(isinf(detect))=0;
[m,n] = size(detect);

if (strcmp(videocc(vc,:),'unrect'))
    for i = 1:m %extract data
        if detect(i,1) == 0 && detect(i,2)
            ↪ == 0 && detect(i,3) == 0 &&
            ↪ detect(i,4) == 0
            side_X(i) = 0;
            depth_Z(i) = 0;
            height_Y(i) = 0;
        else
            height(i) = frame_height*(
                ↪ detect(i,3) - detect(i,1)
                ↪ );
            width(i) = frame_width*(detect(
                ↪ i,4) - detect(i,2));
            depth_from_height(i) =
                ↪ Drone_height * f_yk /
                ↪ height(i);
            depth_from_width(i) =
                ↪ Drone_width * f_xk /
                ↪ width(i);
            depth_Z(i) = (depth_from_height
                ↪ (i) + depth_from_width(i)
                ↪ ) / 2;
            Xmid(i) = frame_width*(detect(i
                ↪ ,4) + detect(i,2))/2;
            Ymid(i) = frame_height*(detect(
                ↪ i,3) + detect(i,1))/2;
            side_X(i) = -(Xmid(i) - c_xk) *
                ↪  depth_Z(i) / f_xk;
            height_Y(i) = -(Ymid(i) - c_yk)
                ↪  * depth_Z(i) / f_yk;
        end
    end
else
    for i = 1:m %extract data
        if detect(i,1) == 0 && detect(i,2)
            ↪ == 0 && detect(i,3) == 0 &&
            ↪ detect(i,4) == 0
            side_X(i) = 0;
            depth_Z(i) = 0;
            height_Y(i) = 0;
```

```
            else
                height(i) = frame_height*(
                    ↪ detect(i,3) - detect(i,1)
                    ↪ );
                width(i) = frame_width*(detect(
                    ↪ i,4) - detect(i,2));
                depth_from_height(i) =
                    ↪ Drone_height * f_yp /
                    ↪ height(i);
                depth_from_width(i) =
                    ↪ Drone_width * f_xp /
                    ↪ width(i);
                depth_Z(i) = (depth_from_height
                    ↪ (i) + depth_from_width(i)
                    ↪ ) / 2;
                Xmid(i) = frame_width*(detect(i
                    ↪ ,4) + detect(i,2))/2;
                Ymid(i) = frame_height*(detect(
                    ↪ i,3) + detect(i,1))/2;
                side_X(i) = -(Xmid(i) - c_xp) *
                    ↪  depth_Z(i) / f_xp;
                height_Y(i) = -(Ymid(i) - c_yp)
                    ↪  * depth_Z(i) / f_yp;
            end
        end
    end

    T = [side_X; depth_Z; height_Y];

    xx=0;%initialize matrix
    dx=0;
    dxx=0;
    vx=0;
    fx=0;

    Dx = [dx,dxx,fx];%intialize vector in x
        ↪ axis, side
    Vx = [xx,vx];

    count_x = 1;
    for i = 1:m %extract data
        dx(i) = T(1,i);
        dy(i) = T(3,i);
        dz(i) = T(2,i) * -1;
        if dx(i) == 0 && dy(i) == 0 && dz(i) ==
```

```
        ↪ 0
        count_x = count_x;
    else
        xx(count_x) = i * 1/30;
        vx(count_x) = vicon(i,1)*100;
        dxx(count_x) = dx(i);
        fx(count_x) = dxx(count_x) - vx(
            ↪ count_x);
        count_x = count_x + 1;
    end
end

Side = figure;set(Side, 'Visible', 'off');
    ↪ %plot and save
plot(xx,dxx,'-r',xx,vx,'-b');
xlabel('Time␣(s)'), ylabel('Relative␣
    ↪ Position␣in␣X␣direction␣(cm)');
legend('Detection␣Data','Vicon␣Data','
    ↪ location','best');
saveas(Side,strcat(result_name,'Side_X'),'
    ↪ jpg');

xy=0;%initalize vector in y axis, height
dy=0;
dyy=0;
vy=0;
fy=0;

Dz = [xy,dy,dyy,fy];%intialize vector
Vz = [xy,vy];

count_y = 1;
for i = 1:m %extract data
    dx(i) = T(1,i);
    dy(i) = T(3,i);
    dz(i) = T(2,i) * -1;
    if dx(i) == 0 && dy(i) == 0 && dz(i) ==
        ↪ 0
        count_y = count_y;
    else
        xy(count_y) = i * 1 / 30;
        vy(count_y) = vicon(i,3)*100;
        dyy(count_y) = dy(i);
        fy(count_y) = dyy(count_y) - vy(
            ↪ count_y);
```

```
        count_y = count_y + 1;
    end
end

Height = figure;set(Height, 'Visible', '
    ↪ off');%plot and save
plot(xy,dyy,'-r',xy,vy,'-b');
xlabel('Time␣(s)'), ylabel('Relative␣
    ↪ Position␣in␣Y␣direction␣(cm)');
legend('Detection␣Data','Vicon␣Data','
    ↪ location','best');
saveas(Height,strcat(result_name,'Height_Y
    ↪ '),'jpg');

xz=0;%initalize vector in z axis, depth
dz=0;
dzz=[];
vz=0;
fz=0;

Dz = [xz,dz,dzz,fz];%intialize vector
Vz = [xz,vz];

count_z = 1;
for i = 1:m %extract data
    dx(i) = T(1,i);
    dy(i) = T(3,i);
    dz(i) = T(2,i) * -1;
    if dx(i) == 0 && dy(i) == 0 && dz(i) ==
        ↪  0
        count_z = count_z;
    else
        xz(count_z) = i * 1 / 30;
        vz(count_z) = vicon(i,2)*100 +
            ↪ 44.37;
        dzz(count_z) = dz(i);
        fz(count_z) = dzz(count_z) - vz(
            ↪ count_z);
        count_z = count_z + 1;
    end
end

Depth = figure;set(Depth, 'Visible', 'off'
    ↪ );%plot and save
plot(xz,dzz,'-r',xz,vz,'-b');
```

```
                    xlabel('Time␣(s)'), ylabel('Relative␣
                        ↪ Position␣in␣Z␣Direction␣(cm)');
                    legend('Detection␣Data','Vicon␣Data','
                        ↪ location','best');
                    saveas(Depth,strcat(result_name,'Depth_Z')
                        ↪ ,'jpg');

                    Difference = figure;set(Difference, '
                        ↪ Visible', 'off');%plot and save
                    plot(xy,fx,'-r',xy,fy,'-b',xy,fz,'-g');
                    xlabel('Time␣(s)'), ylabel('Difference␣(cm
                        ↪ )');
                    legend('Difference␣in␣Side␣x','Difference␣
                        ↪ in␣Height␣y','Difference␣in␣Depth␣z'
                        ↪ ,'location','best');
                    saveas(Difference,strcat(result_name,'
                        ↪ Difference'),'jpg');

                    fx_mean = rms(fx);
                    fy_mean = rms(fy);
                    fz_mean = rms(fz);
                    ave_mean = (fx_mean+fy_mean+fz_mean)/3;
                    count_xls = count_xls + 1;
                    xlscell = strcat('A',num2str(count_xls));

                    T_excel = [strcat(framework(f,:),'_',TFapi
                        ↪ (tf,:),'_',test(tt,:),'_',background
                        ↪ (bk,:),'_',videocc(vc,:),'
                        ↪ _train_unrect'),fx_mean,fy_mean,
                        ↪ fz_mean, ave_mean];
                    xlswrite('TensorFlow.xlsx',T_excel,string(
                        ↪ TFapi(tf,:)),xlscell)
                end
            end
        end
    end
elseif (strcmp(framework(f,:),'Darknet'))
    for dk = 1:mDK
        for tt = 1:mtest
            for bk = 1:mbackground
                for tr = 1:mtraincc
                    for vc = 1:mvideo
                        if (strcmp(strcat(DKapi(dk,:),'\',
                            ↪ traincc(tr,:)),'Tiny_YOLO\
                            ↪ train_unrect'))
```

```
    continue
else
    vicon_path = string(strcat(basead,'
        ↪ Vicon_Data\',background(bk,:)
        ↪ ,'\',test(tt,:),'\vicon_data.
        ↪ txt'));
    vicon = csvread(vicon_path);
    detect_path = string(strcat(basead,
        ↪ framework(f,:),'\',DKapi(dk
        ↪ ,:),'\',traincc(tr,:),'\',
        ↪ background(bk,:),'\','bbox','
        ↪ \',test(tt,:),'_',videocc(vc
        ↪ ,:),'.csv'));
    result_name = string(strcat(
        ↪ framework(f,:),'_',DKapi(dk
        ↪ ,:),'_',traincc(tr,:),'_',
        ↪ background(bk,:),'_',test(tt
        ↪ ,:),'_',videocc(vc,:)));
    detect = csvread(detect_path);
end
%initiate variables
height = 0;
width = 0;
depth_from_height = 0;
depth_from_width = 0;
depth_Z = 0;
Xmid = 0;
Ymid = 0;
side_X = 0;
height_Y = 0;


detect(isinf(detect))=0;
[m,n] = size(detect);

if (strcmp(videocc(vc,:),'unrect'))
    for i = 1:m %extract data
        if detect(i,1) == 0 && detect(i
            ↪ ,2) == 0 && detect(i,3)
            ↪ == 0 && detect(i,4) == 0
            side_X(i) = 0;
            depth_Z(i) = 0;
            height_Y(i) = 0;
        else
            height(i) = detect(i,4) -
```

```
                    ↪ detect(i,2);
                width(i) = detect(i,3) -
                    ↪ detect(i,1);
                depth_from_height(i) =
                    ↪ Drone_height * f_yk /
                    ↪ height(i);
                depth_from_width(i) =
                    ↪ Drone_width * f_xk /
                    ↪ width(i);
                depth_Z(i) = (
                    ↪ depth_from_height(i) +
                    ↪  depth_from_width(i))
                    ↪ / 2;
                Xmid(i) = (detect(i,3) +
                    ↪ detect(i,1))/2;
                Ymid(i) = (detect(i,4) +
                    ↪ detect(i,2))/2;
                side_X(i) = -(Xmid(i) - c_xk
                    ↪ ) * depth_Z(i) / f_xk;
                height_Y(i) = -(Ymid(i) -
                    ↪ c_yk) * depth_Z(i) /
                    ↪ f_yk;
            end
        end
    else
        for i = 1:m %extract data
            if detect(i,1) == 0 && detect(i
                ↪ ,2) == 0 && detect(i,3)
                ↪ == 0 && detect(i,4) == 0
                side_X(i) = 0;
                depth_Z(i) = 0;
                height_Y(i) = 0;
            else
                height(i) = detect(i,4) -
                    ↪ detect(i,2);
                width(i) = detect(i,3) -
                    ↪ detect(i,1);
                depth_from_height(i) =
                    ↪ Drone_height * f_yp /
                    ↪ height(i);
                depth_from_width(i) =
                    ↪ Drone_width * f_xp /
                    ↪ width(i);
                depth_Z(i) = (
                    ↪ depth_from_height(i) +
```

```
                ↪  depth_from_width(i))
                ↪ / 2;
            Xmid(i) = (detect(i,3) +
                ↪ detect(i,1))/2;
            Ymid(i) = (detect(i,4) +
                ↪ detect(i,2))/2;
            side_X(i) = -(Xmid(i) - c_xp
                ↪ ) * depth_Z(i) / f_xp;
            height_Y(i) = -(Ymid(i) -
                ↪ c_yp) * depth_Z(i) /
                ↪ f_yp;
        end
    end
end

T = [side_X; depth_Z; height_Y];

xx=0;%initialize matrix
dx=0;
dxx=0;
vx=0;
fx=0;

Dx = [dx,dxx,fx];%intialize vector in
    ↪ x axis, side
Vx = [xx,vx];

count_x = 1;
for i = 1:m %extract data
    dx(i) = T(1,i);
    dy(i) = T(3,i);
    dz(i) = T(2,i) * -1;
    if dx(i) == 0 && dy(i) == 0 && dz(i
        ↪ ) == 0
        count_x = count_x;
    else
        xx(count_x) = i * 1/30;
        vx(count_x) = vicon(i,1)*100;
        dxx(count_x) = dx(i);
        fx(count_x) = dxx(count_x) - vx
            ↪ (count_x);
        count_x = count_x + 1;
    end
end
```

```
Side = figure;set(Side, 'Visible', 'off
    ↪ ');%plot and save
plot(xx,dxx,'-r',xx,vx,'-b');
xlabel('Time␣(s)'), ylabel('Relative␣
    ↪ Position␣in␣X␣direction␣(cm)');
legend('Detection␣Data','Vicon␣Data','
    ↪ location','best');
saveas(Side,strcat(result_name,'Side_X'
    ↪ ),'jpg');

xy=0;%initalize vector in y axis,
    ↪ height
dy=0;
dyy=0;
vy=0;
fy=0;

Dz = [xy,dy,dyy,fy];%intialize vector
Vz = [xy,vy];

count_y = 1;
for i = 1:m %extract data
    dx(i) = T(1,i);
    dy(i) = T(3,i);
    dz(i) = T(2,i) * -1;
    if dx(i) == 0 && dy(i) == 0 && dz(i
        ↪ ) == 0
        count_y = count_y;
    else
        xy(count_y) = i * 1 / 30;
        vy(count_y) = vicon(i,3)*100;
        dyy(count_y) = dy(i);
        fy(count_y) = dyy(count_y) - vy
            ↪ (count_y);
        count_y = count_y + 1;
    end
end

Height = figure;set(Height, 'Visible',
    ↪ 'off');%plot and save
plot(xy,dyy,'-r',xy,vy,'-b');
xlabel('Time␣(s)'), ylabel('Relative␣
    ↪ Position␣in␣Y␣direction␣(cm)');
legend('Detection␣Data','Vicon␣Data','
    ↪ location','best');
```

```
saveas(Height,strcat(result_name,'
    ↪ Height_Y'),'jpg');

xz=0;%initalize vector in z axis,
    ↪ depth
dz=0;
dzz=[];
vz=0;
fz=0;

Dz = [xz,dz,dzz,fz];%intialize vector
Vz = [xz,vz];

count_z = 1;
for i = 1:m %extract data
    dx(i) = T(1,i);
    dy(i) = T(3,i);
    dz(i) = T(2,i) * -1;
    if dx(i) == 0 && dy(i) == 0 && dz(i
        ↪ ) == 0
        count_z = count_z;
    else
        xz(count_z) = i * 1 / 30;
        vz(count_z) = vicon(i,2)*100 +
            ↪ 44.37;
        dzz(count_z) = dz(i);
        fz(count_z) = dzz(count_z) - vz
            ↪ (count_z);
        count_z = count_z + 1;
    end
end

Depth = figure;set(Depth, 'Visible', '
    ↪ off');%plot and save
plot(xz,dzz,'-r',xz,vz,'-b');
xlabel('Time␣(s)'), ylabel('Relative␣
    ↪ Position␣in␣Z␣Direction␣(cm)');
legend('Detection␣Data','Vicon␣Data','
    ↪ location','best');
saveas(Depth,strcat(result_name,'
    ↪ Depth_Z'),'jpg');

Difference = figure;set(Difference, '
    ↪ Visible', 'off');%plot and save
plot(xy,fx,'-r',xy,fy,'-b',xy,fz,'-g');
```

```
                        xlabel('Time (s)'), ylabel('Difference
                            ↪ (cm)');
                        legend('Difference in Side x','
                            ↪ Difference in Height y','
                            ↪ Difference in Depth z','location
                            ↪ ','best');
                        saveas(Difference,strcat(result_name,'
                            ↪ Difference'),'jpg');

                        fx_mean = rms(fx);
                        fy_mean = rms(fy);
                        fz_mean = rms(fz);
                        ave_mean = (fx_mean+fy_mean+fz_mean)/3;
                        count_xls = count_xls + 1;
                        xlscell = strcat('A',num2str(count_xls)
                            ↪ );

                        T_excel = [strcat(framework(f,:),'_',
                            ↪ DKapi(dk,:),'_',test(tt,:),'_',
                            ↪ background(bk,:),'_',videocc(vc
                            ↪ ,:),'_',traincc(tr,:)),fx_mean,
                            ↪ fy_mean,fz_mean, ave_mean];
                        xlswrite('Darknet.xlsx',T_excel,string(
                            ↪ DKapi(dk,:)),xlscell)
                    end
                end
            end
        end
    end
end
disp('finish!')
```

A.4 Training Data Extraction of Darknet

In section, Python code for extracting training data of Darknet is presented. Unlike TensorFlow, Darknet does not provide a tool to monitor training process. The code is developed to get useful information logged from training terminal.

```
# coding=utf-8
```

```
old = open('log_without_title.txt')
new = open('extracted_log__yolov2.txt', 'w')

for line in old:
    if 'Loaded' in line:
        continue
    if 'Region' in line:
        continue
    if 'seconds' in line:
        new.write(line)

old.close()
new.close()
```

A.5 Monte Carlo Method for Uncertainty Analysis

In this section, Matlab code for uncertainty analysis for distance estimation using Monte Carlo method is presented.

```
clear;
clc;

m_Drone_height = 12.59;
std_h = 0.01;
Drone_height = normrnd(m_Drone_height,std_h,[1000,1]);
m_Drone_width = 50.65;
std_w = 0.01;
Drone_width = normrnd(m_Drone_width,std_w,[1000,1]);

m_fxk = 551.9830968;
std_fxk = 12.52216056;
f_xk = normrnd(m_fxk,std_fxk,[1000,1]);
m_fyk = 552.1858034;
std_fyk = 12.53326553;
f_yk = normrnd(m_fyk,std_fyk,[1000,1]);
m_cxk = 321.609551;
std_cxk = 9.17093802;
c_xk = normrnd(m_cxk,std_cxk,[1000,1]);
m_cyk = 183.4030846;
std_cyk = 5.93460473;
c_yk = normrnd(m_cyk,std_cyk,[1000,1]);
```

```
m_fxp = 437.9295603;
std_fxp = 12.58862952;
f_xp = normrnd(m_fxp,std_fxp,[1000,1]);
m_fyp = 516.3283819;
std_fyp = 12.50180664;
f_yp = normrnd(m_fyp,std_fyp,[1000,1]);
m_cxp = 323.3832733;
std_cxp = 12.57457631;
c_xp = normrnd(m_cxp,std_cxp,[1000,1]);
m_cyp = 183.7519438;
std_cyp = 6.73694362;
c_yp = normrnd(m_cyp,std_cyp,[1000,1]);

frame_width = 640;
frame_height = 360;

x_min = 359;
x_max = 490;
y_min = 99;
y_max = 127;

%unrectified simulation
for i = 1:1000
    height(i) = abs(y_min - y_max);
    width(i) = abs(x_min - x_max);
    depth_from_height(i) = Drone_height(i,1) * f_yk(i,1) /
        ↪ height(i);
    depth_from_width(i) = Drone_width(i,1) * f_xk(i,1) / width(i
        ↪ );
    depth_Zk(i) = (depth_from_height(i) + depth_from_width(i)) /
        ↪  2;
    Xmid(i) = (x_min + x_max)/2;
    Ymid(i) = (y_min + y_max)/2;
    side_Xk(i) = -(Xmid(i) - c_xk(i,1)) * depth_Zk(i) / f_xk(i
        ↪ ,1);
    height_Yk(i) = -(Ymid(i) - c_yk(i,1)) * depth_Zk(i) / f_yk(i
        ↪ ,1);
end

T_k = [side_Xk; depth_Zk; height_Yk];
T_kt = transpose(T_k);
xlswrite('monte␣carlo␣camera.xlsx',T_kt)


%rectified simulation
```

```
for i = 1: 1000
    height(i) = abs(y_min - y_max);
    width(i) = abs(x_min - x_max);
    depth_from_height(i) = Drone_height(i,1) * f_yp(i,1) /
       ↪ height(i);
    depth_from_width(i) = Drone_width(i,1) * f_xp(i,1) / width(i
       ↪ );
    depth_Zp(i) = (depth_from_height(i) + depth_from_width(i)) /
       ↪  2;
    Xmid(i) = (x_min + x_max)/2;
    Ymid(i) = (y_min + y_max)/2;
    side_Xp(i) = -(Xmid(i) - c_xp(i,1)) * depth_Zp(i) / f_xp(i
       ↪ ,1);
    height_Yp(i) = -(Ymid(i) - c_yp(i,1)) * depth_Zp(i) / f_yp(i
       ↪ ,1);
end


T_p = [side_Xp; depth_Zp; height_Yp];
T_pt = transpose(T_p);
xlswrite('monte␣carlo␣projection.xlsx',T_pt)

disp('finish!')
```

Appendix B

Supplementary Data

B.1 Loss Curves of Training for Object Detection Systems

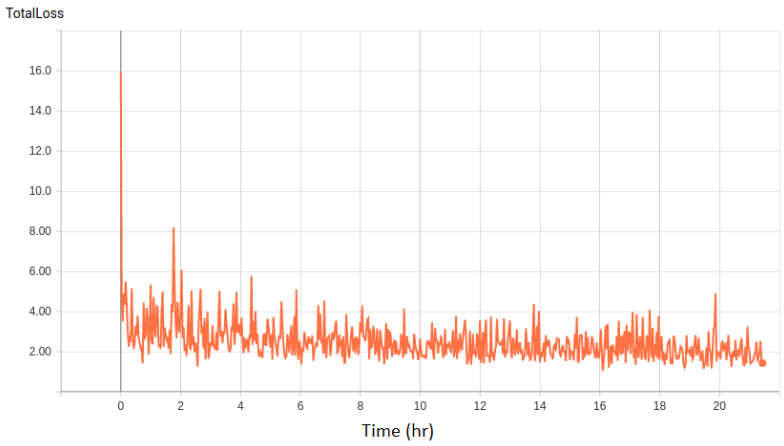

Figure B.1: SSD Inception v2 Total Loss

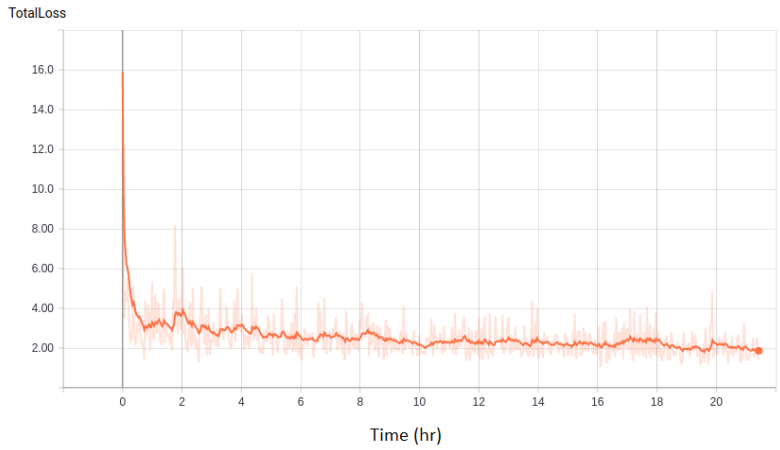

Figure B.2: SSD Inception v2 Total Loss after Filtering

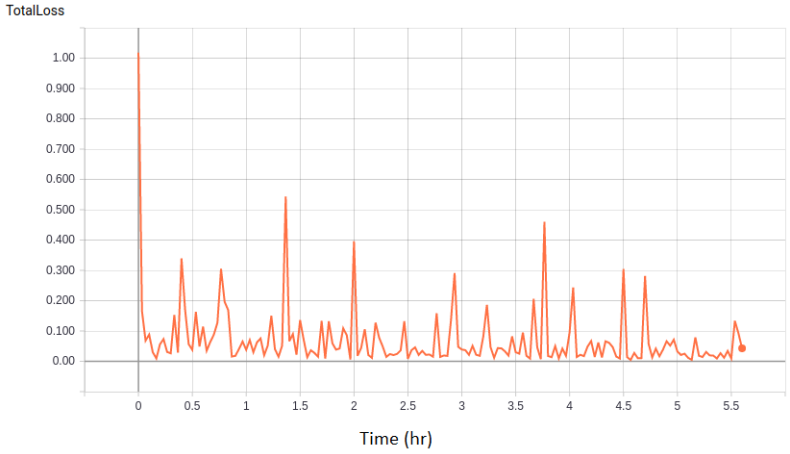

Figure B.3: Faster RCNN Inception v2 Total Loss

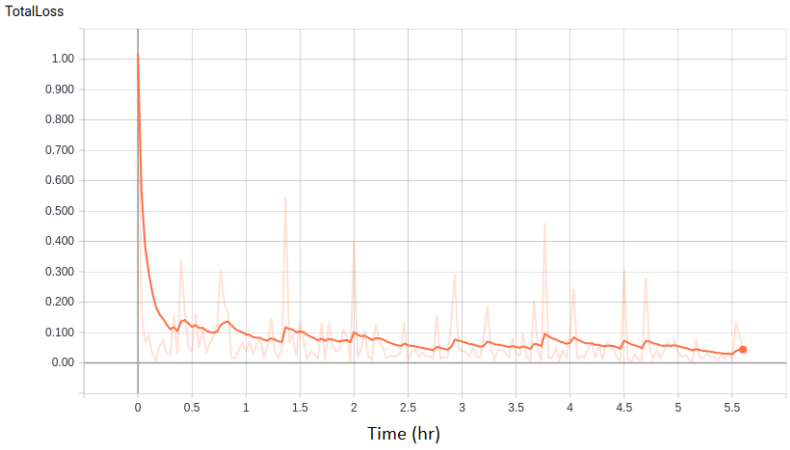

Figure B.4: Faster RCNN Inception v2 Total Loss after Filtering

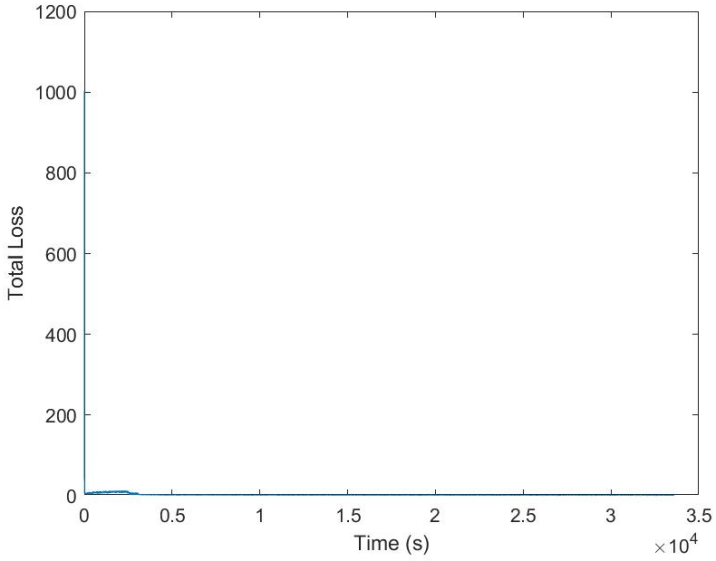

Figure B.5: Tiny YOLO Total Loss

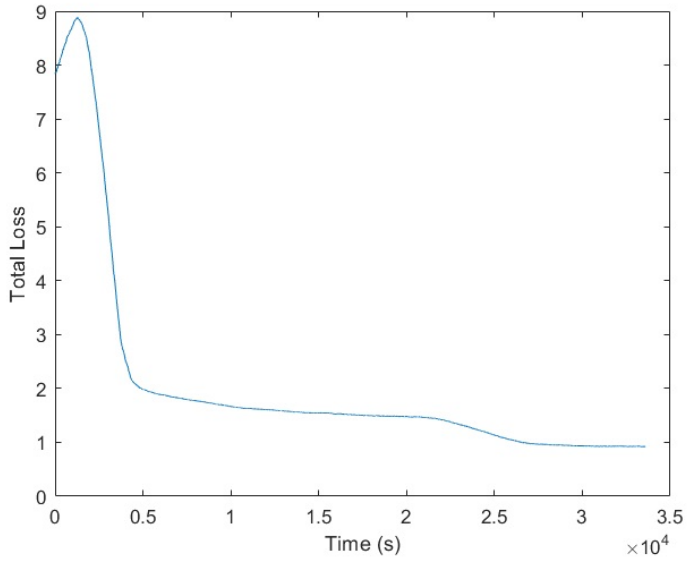

Figure B.6: Zoomed and Smoothed Tiny YOLO Total Loss

B.2 Detection Results

In this section, detailed experimental results from object detection systems of different flight patterns are listed. The abbreviation of different experiment setups are listed below.

Table B.1: Test Configuration and Abbreviation

Test Configuration	**Abbreviation**
With White Curtain	WWC
Without White Curtain	OWC
Video Rectified	VR
Video Unrectified	VU
Training with Rectified Images	TR
Training with Unrectified Images	TU

B.2.1 Differences between Distance Estimations and Vicon Data

In this Section, histograms of differences probability distribution between distance estimations using the equations derived in Section 2.5 and Vicon data are illustrated below.

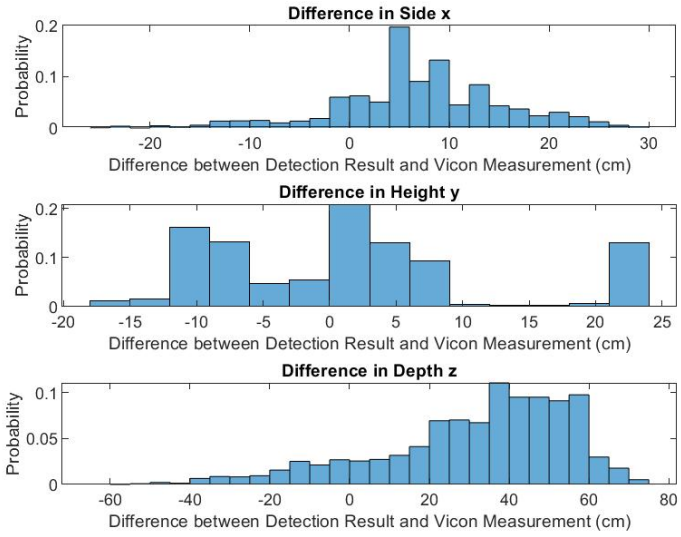

Figure B.7: SSD MobileNet v1 Pure Translation in Side and Height Direction Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

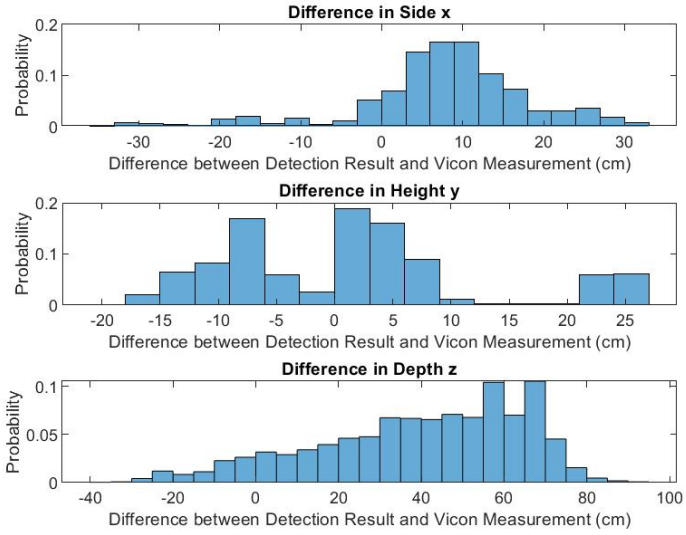

Figure B.8: SSD MobileNet v1 Pure Translation in Side and Height Direction Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

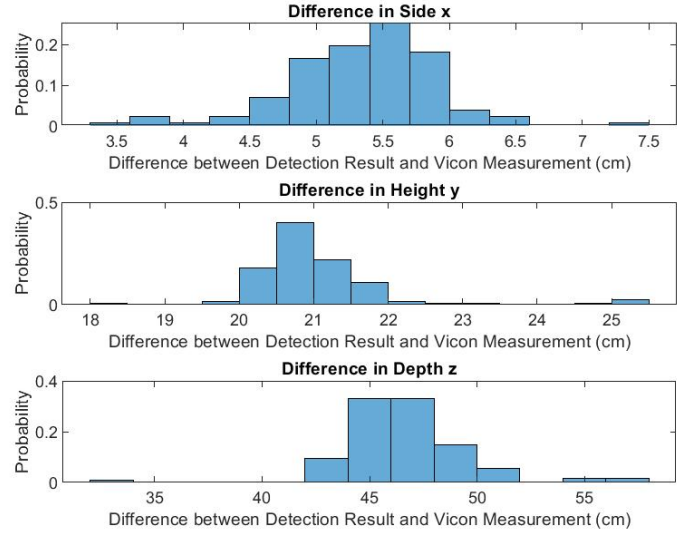

Figure B.9: SSD MobileNet v1 Pure Translation in Side and Height Direction Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

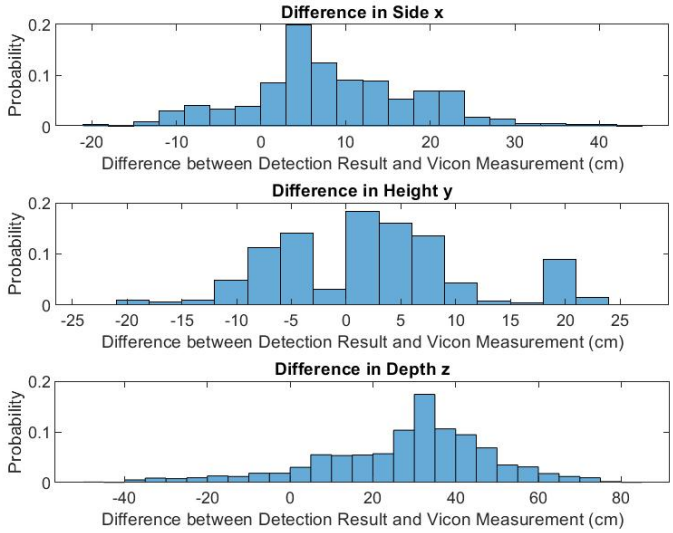

Figure B.10: SSD MobileNet v1 Pure Translation in Side and Height Direction Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

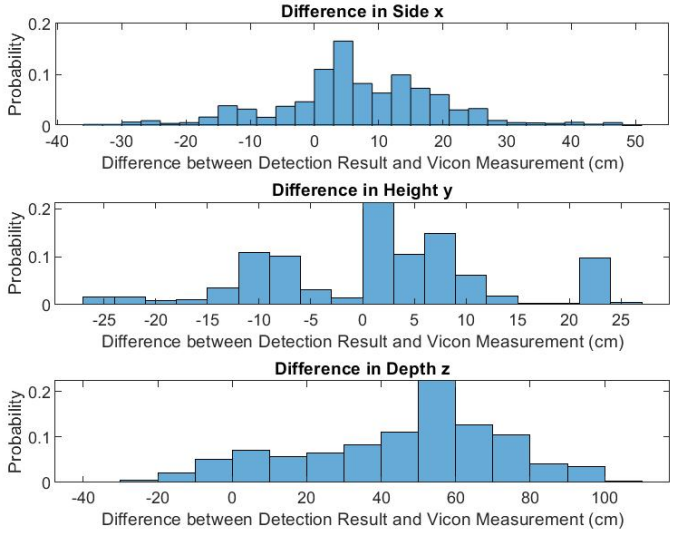

Figure B.11: SSD MobileNet v1 Pure Translation in Side and Height Direction Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

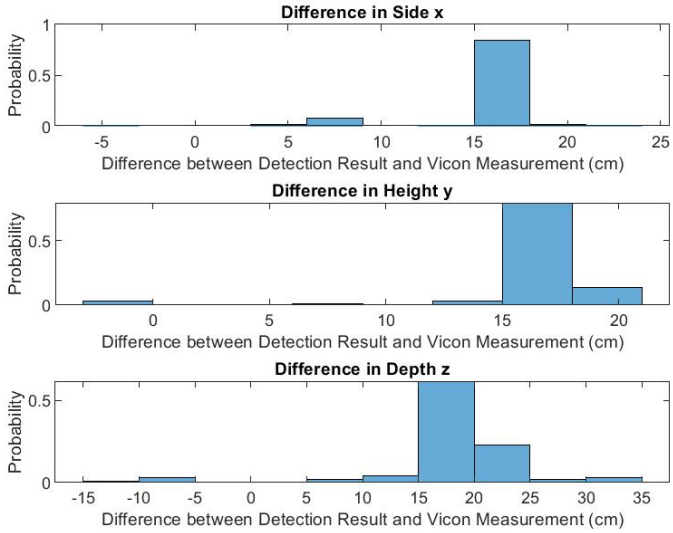

Figure B.12: SSD MobileNet v1 Pure Translation in Side and Height Direction Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

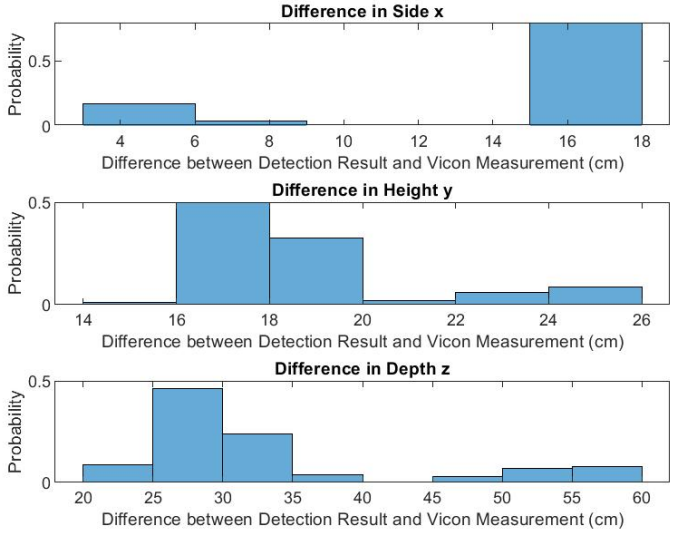

Figure B.13: SSD MobileNet v1 Pure Translation in Side and Height Direction Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

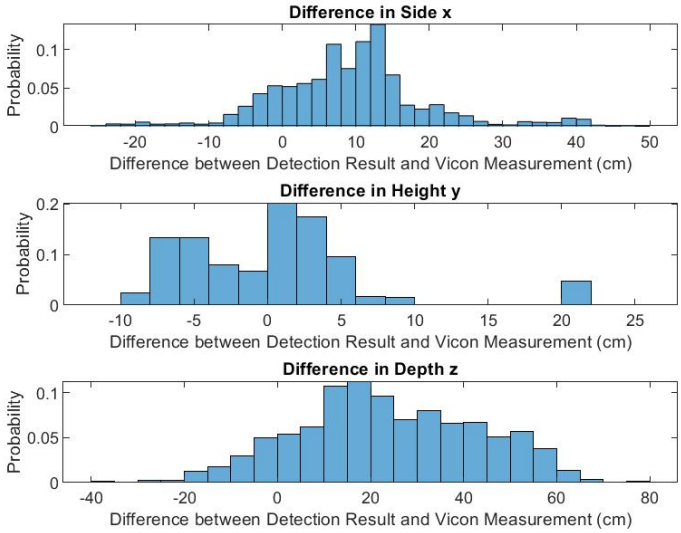

Figure B.14: SSD MobileNet v1 Pure Translation in Depth Direction Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

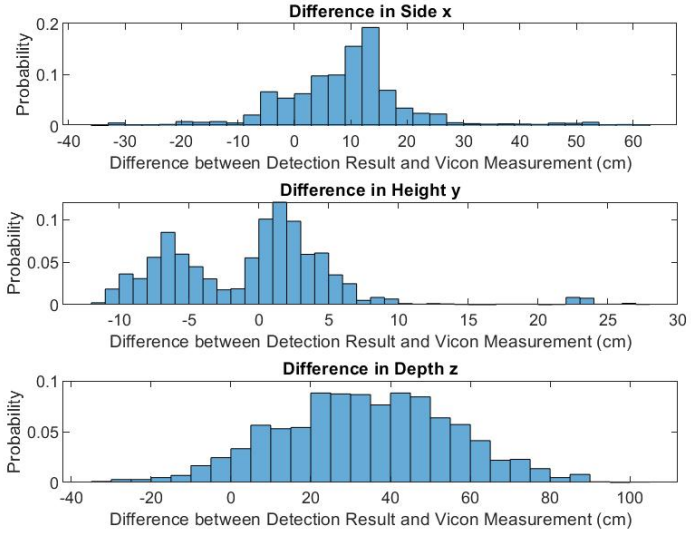

Figure B.15: SSD MobileNet v1 Pure Translation in Depth Direction Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

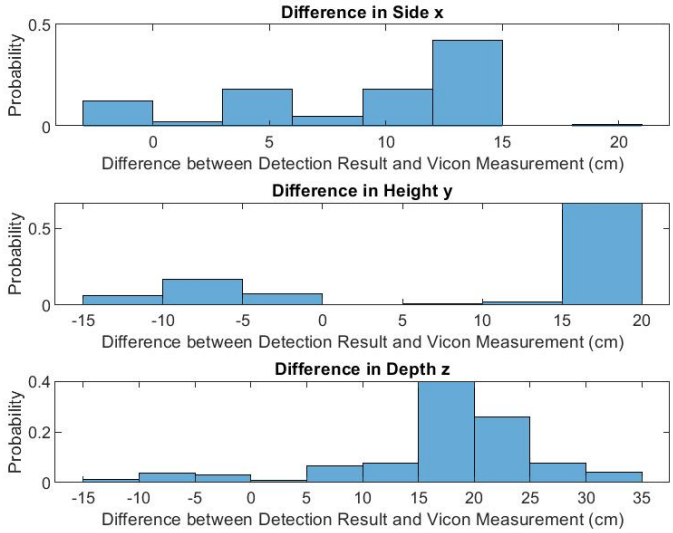

Figure B.16: SSD MobileNet v1 Pure Translation in Depth Direction Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

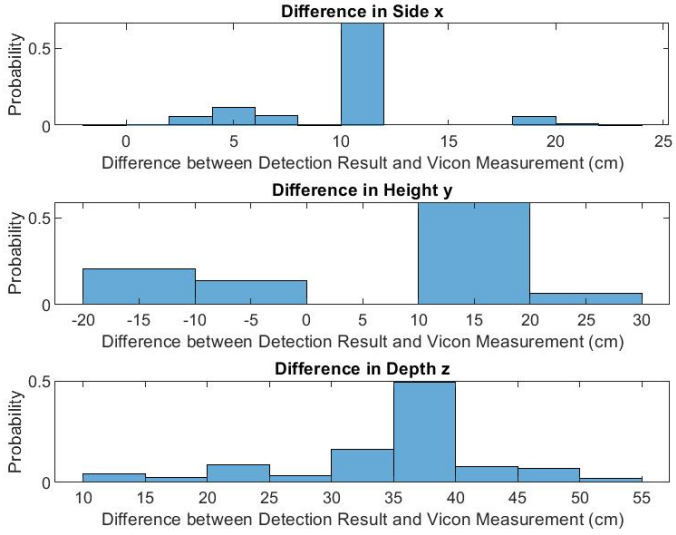

Figure B.17: SSD MobileNet v1 Pure Translation in Depth Direction Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

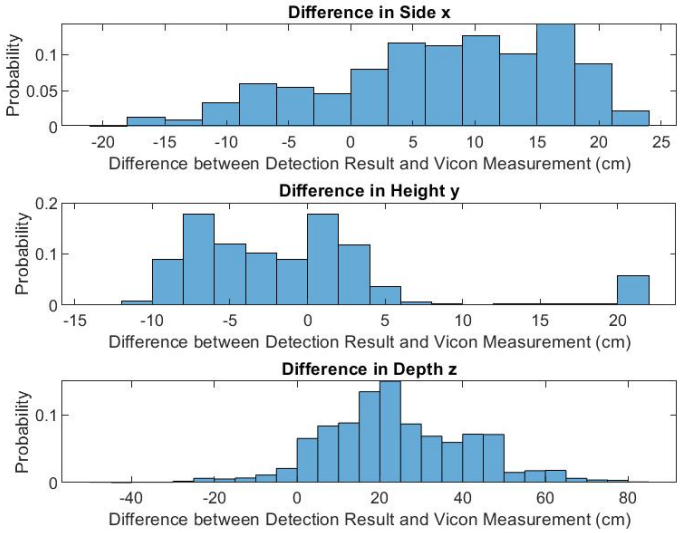

Figure B.18: SSD MobileNet v1 Pure Translation in Depth Direction Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

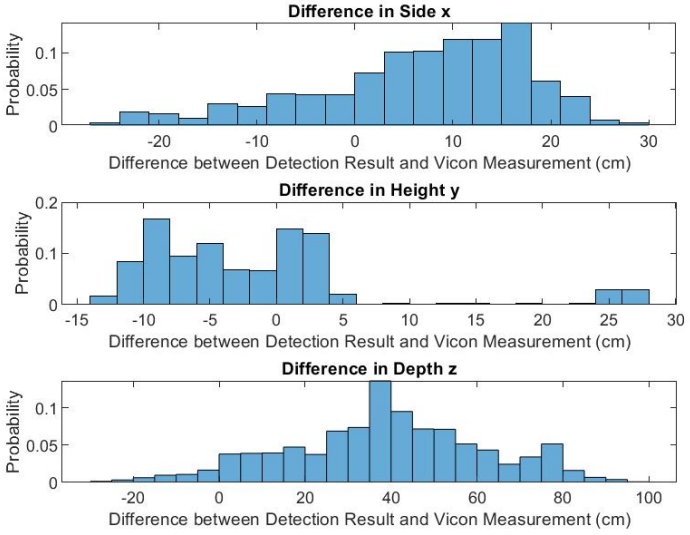

Figure B.19: SSD MobileNet v1 Pure Translation in Depth Direction Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

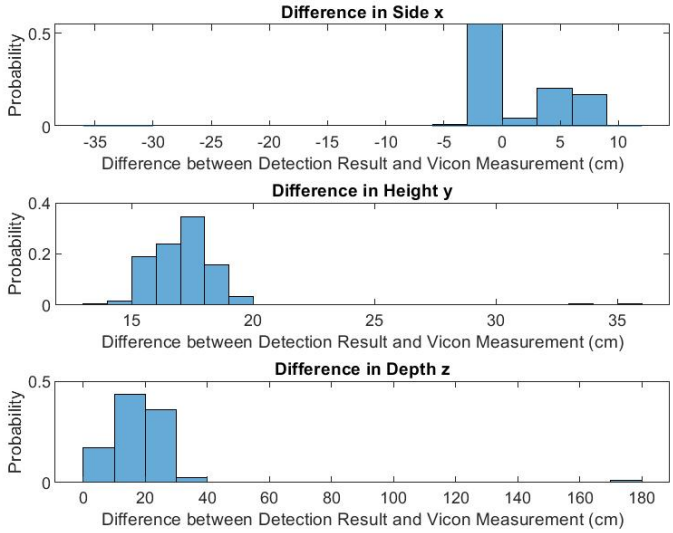

Figure B.20: SSD MobileNet v1 Pure Translation in Depth Direction Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

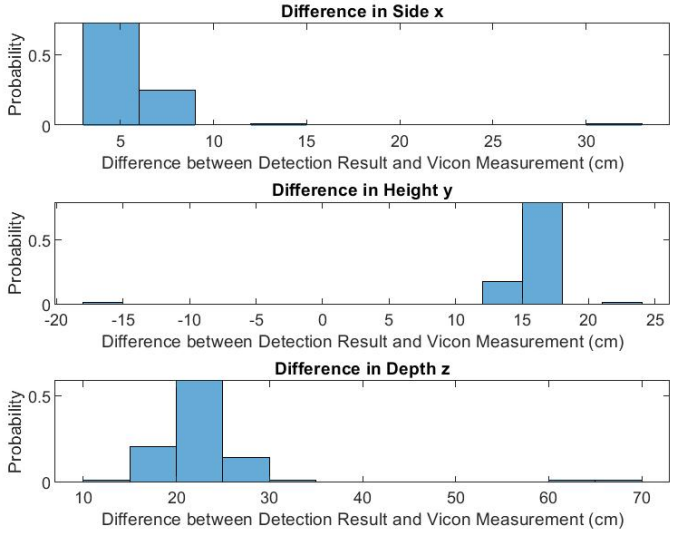

Figure B.21: SSD MobileNet v1 Pure Translation in Depth Direction Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

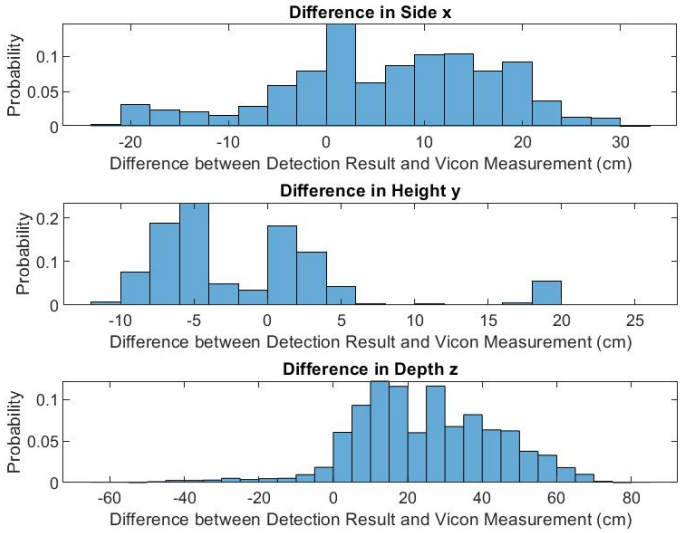

Figure B.22: SSD MobileNet v1 Pure Rotation Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

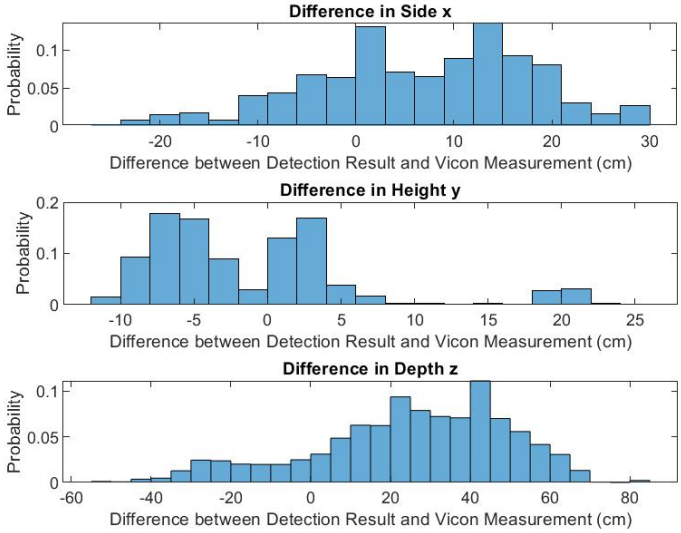

Figure B.23: SSD MobileNet v1 Pure Rotation Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

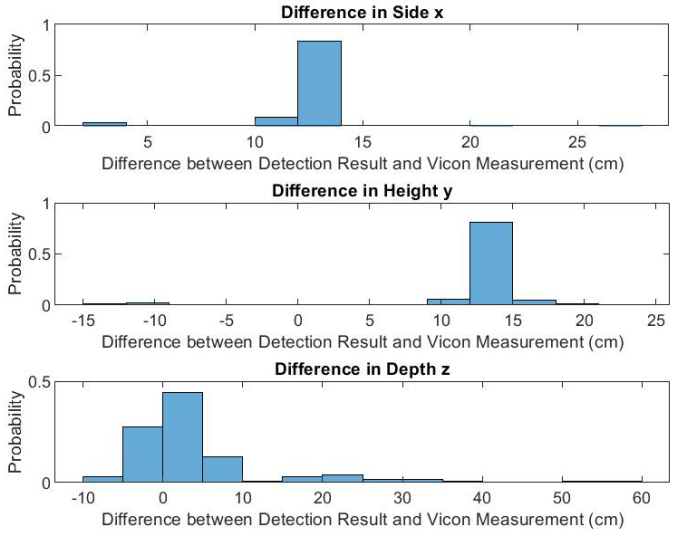

Figure B.24: SSD MobileNet v1 Pure Rotation Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

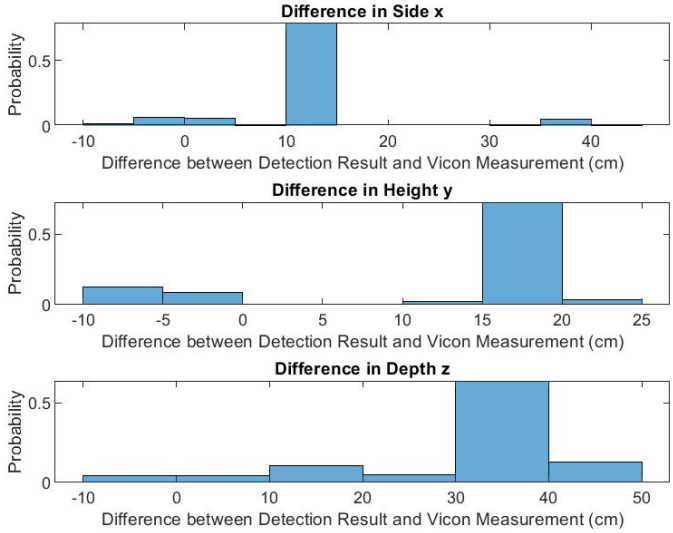

Figure B.25: SSD MobileNet v1 Pure Rotation Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

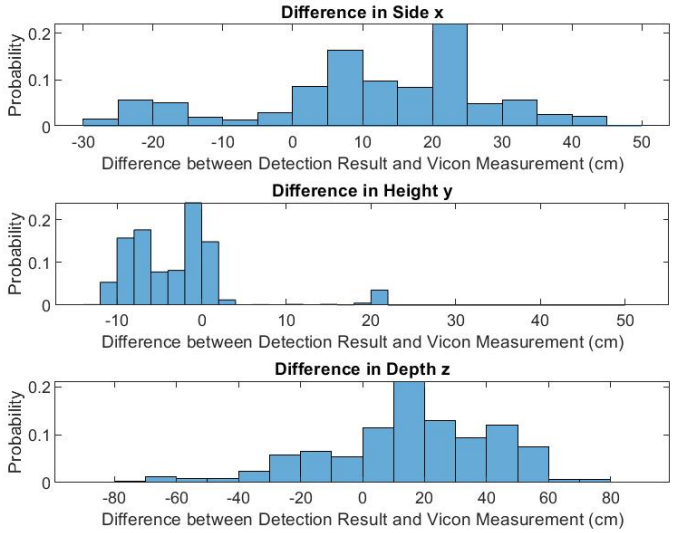

Figure B.26: SSD MobileNet v1 Pure Rotation Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

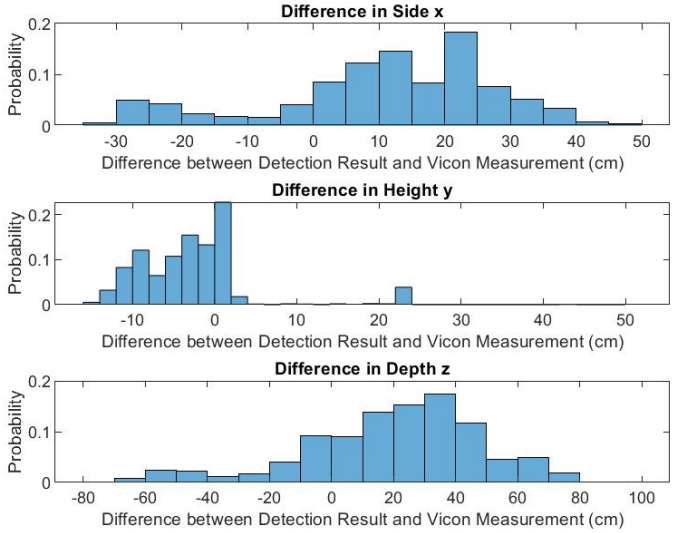

Figure B.27: SSD MobileNet v1 Pure Rotation Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

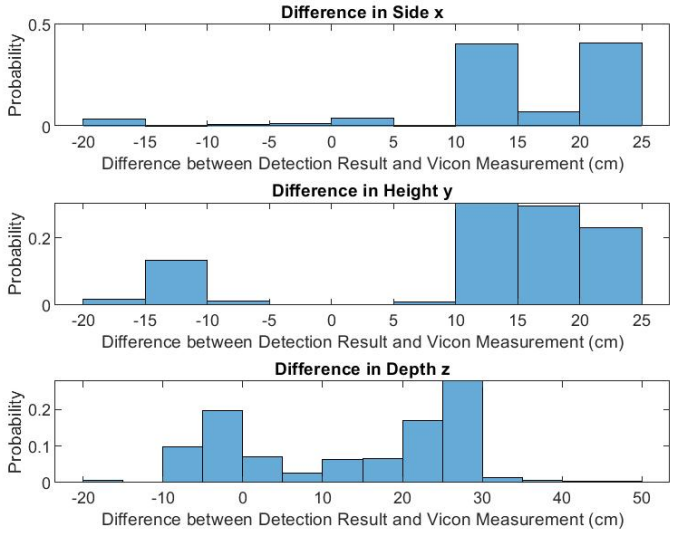

Figure B.28: SSD MobileNet v1 Pure Rotation Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

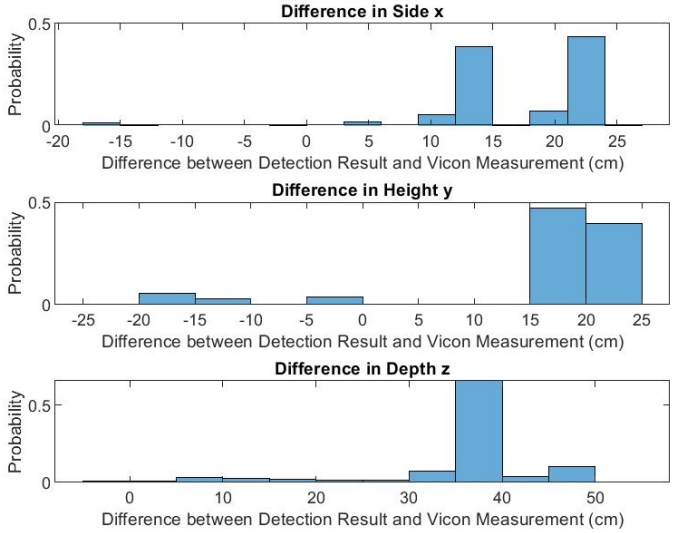

Figure B.29: SSD MobileNet v1 Pure Rotation Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

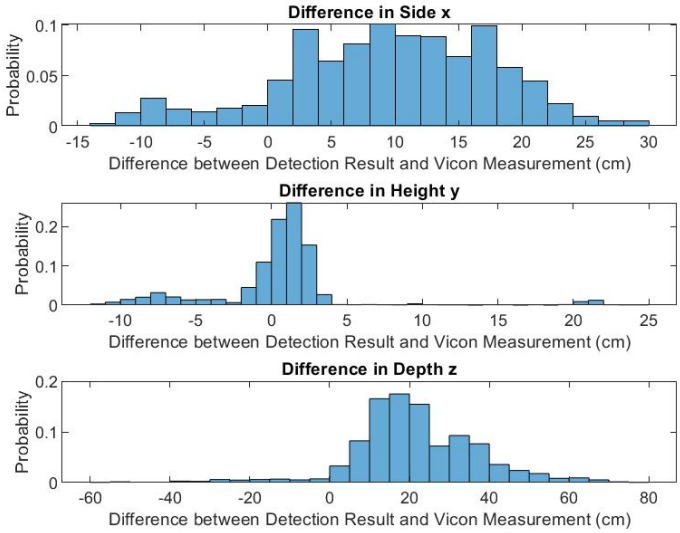

Figure B.30: SSD MobileNet v1 Random Flight Pattern Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

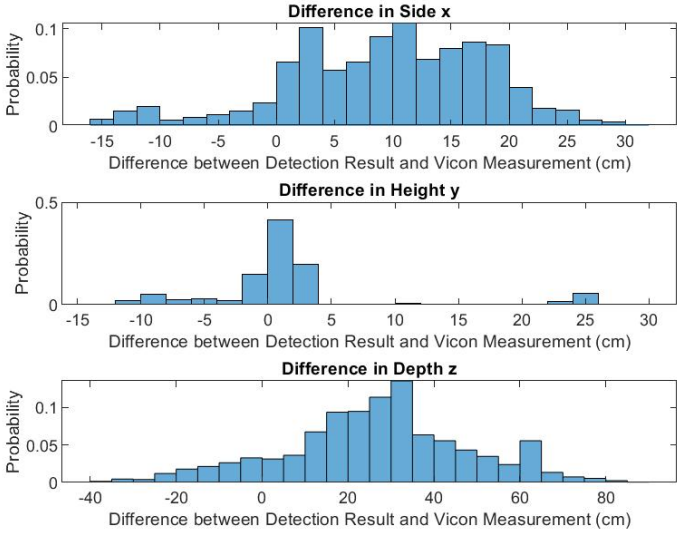

Figure B.31: SSD MobileNet v1 Random Flight Pattern Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

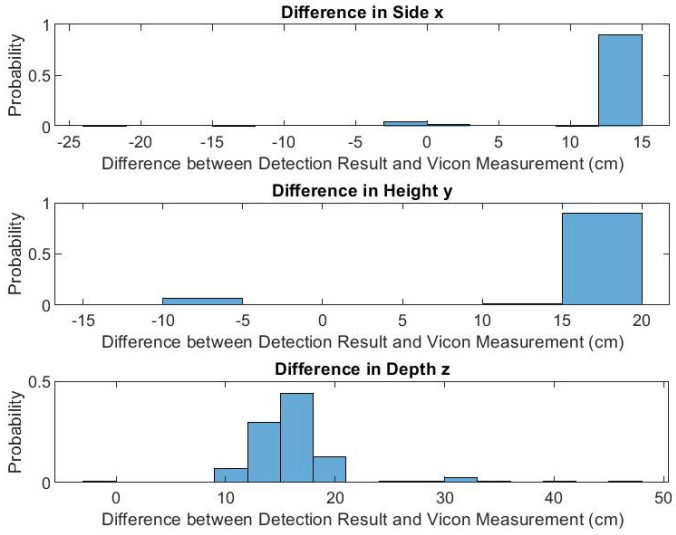

Figure B.32: SSD MobileNet v1 Random Flight Pattern Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

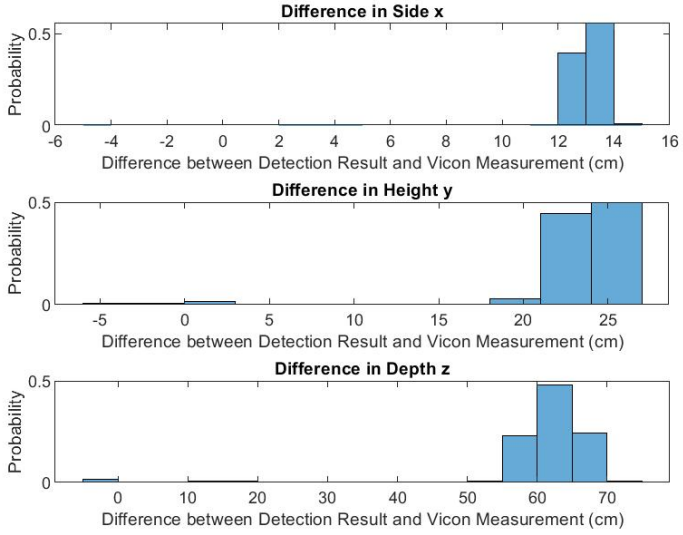

Figure B.33: SSD MobileNet v1 Random Flight Pattern Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

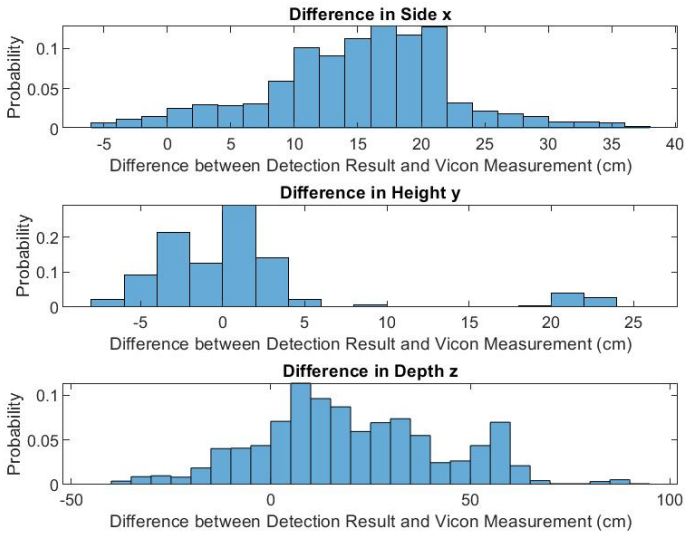

Figure B.34: SSD MobileNet v1 Random Flight Pattern Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

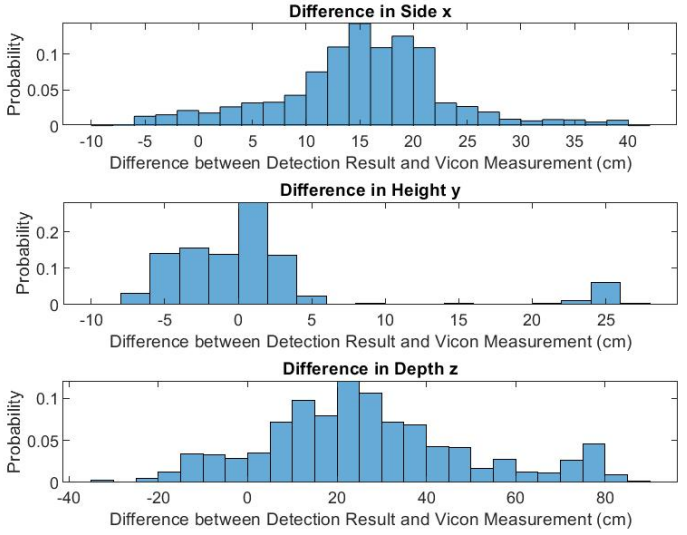

Figure B.35: SSD MobileNet v1 Random Flight Pattern Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

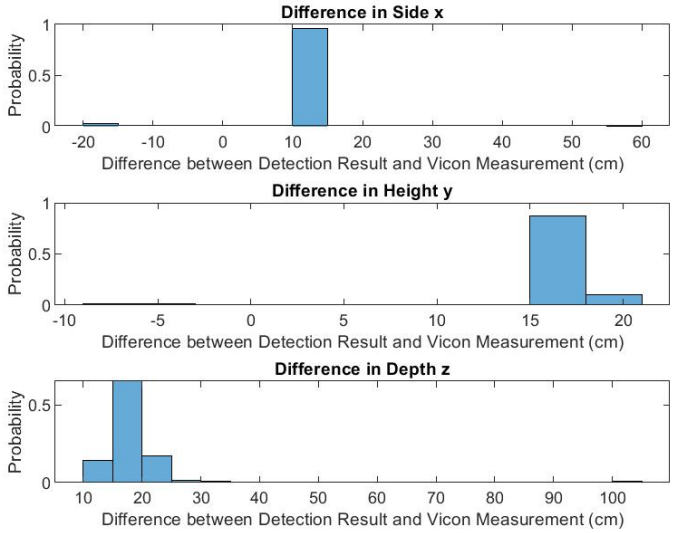

Figure B.36: SSD MobileNet v1 Random Flight Pattern Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

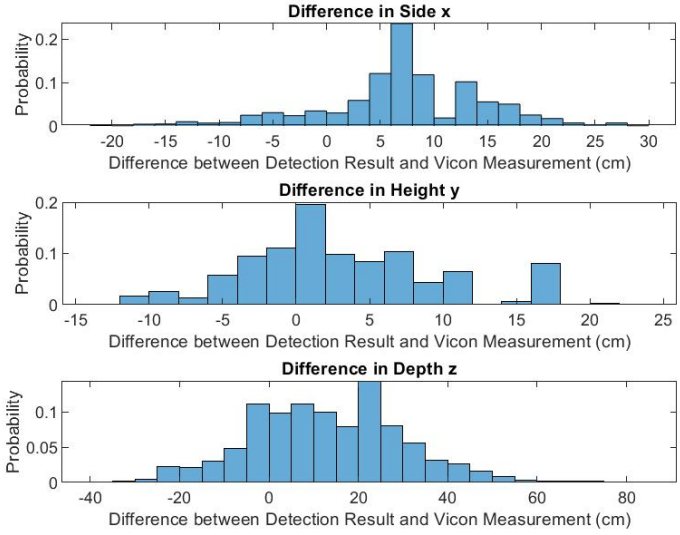

Figure B.37: SSD Inception v2 Pure Translation in Side and Height Direction Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

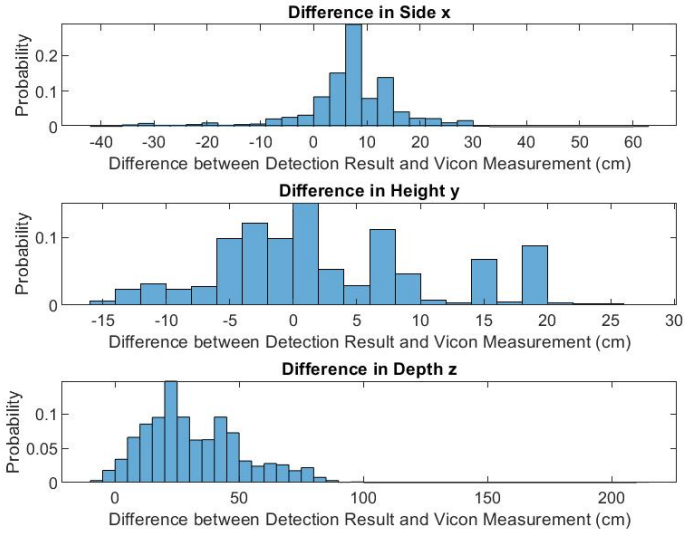

Figure B.38: SSD Inception v2 Pure Translation in Side and Height Direction Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

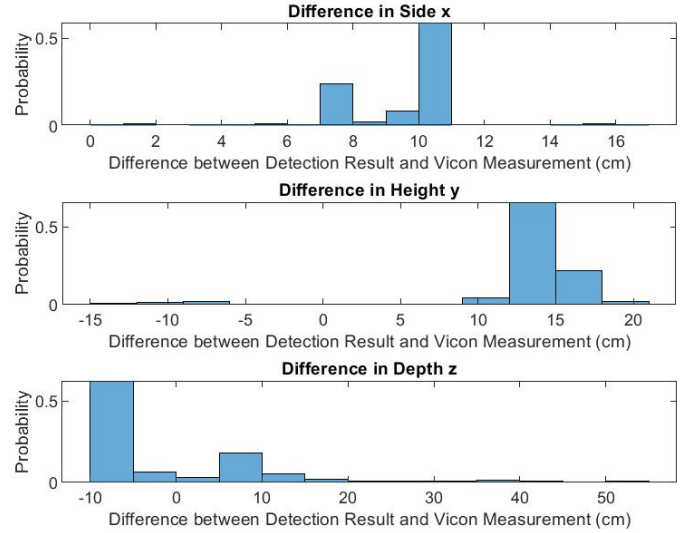

Figure B.39: SSD Inception v2 Pure Translation in Side and Height Direction Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

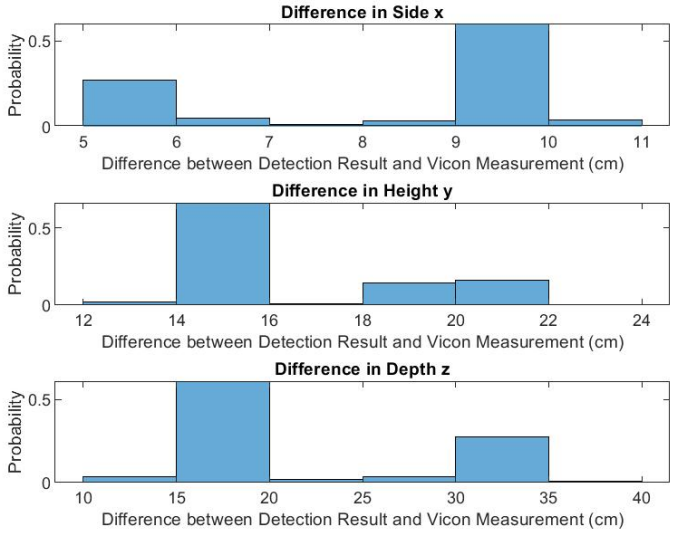

Figure B.40: SSD Inception v2 Pure Translation in Side and Height Direction Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

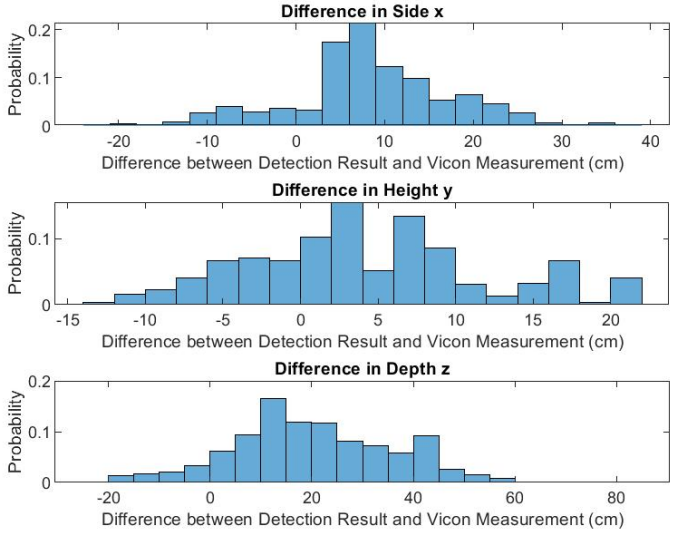

Figure B.41: SSD Inception v2 Pure Translation in Side and Height Direction Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

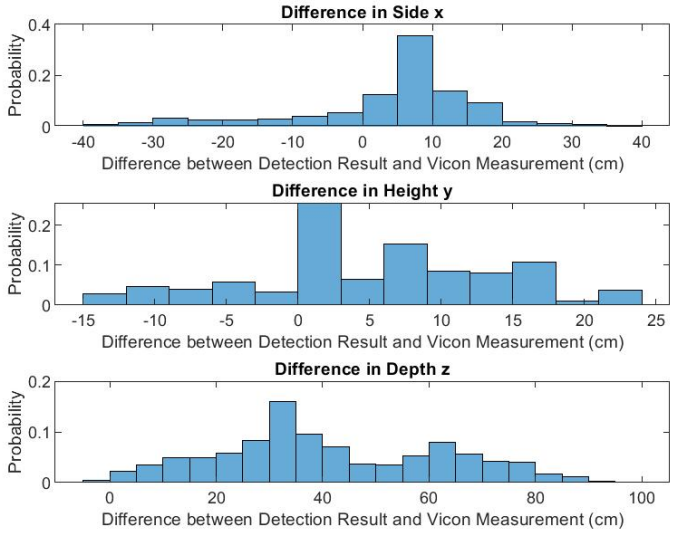

Figure B.42: SSD Inception v2 Pure Translation in Side and Height Direction Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

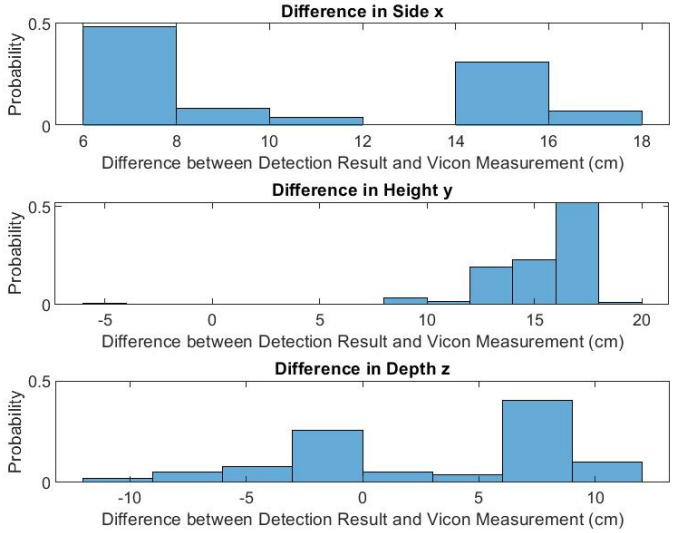

Figure B.43: SSD Inception v2 Pure Translation in Side and Height Direction Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

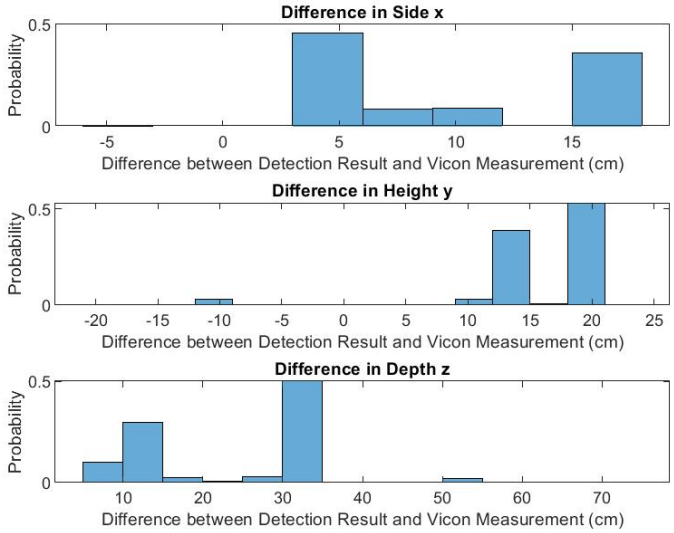

Figure B.44: SSD Inception v2 Pure Translation in Side and Height Direction Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

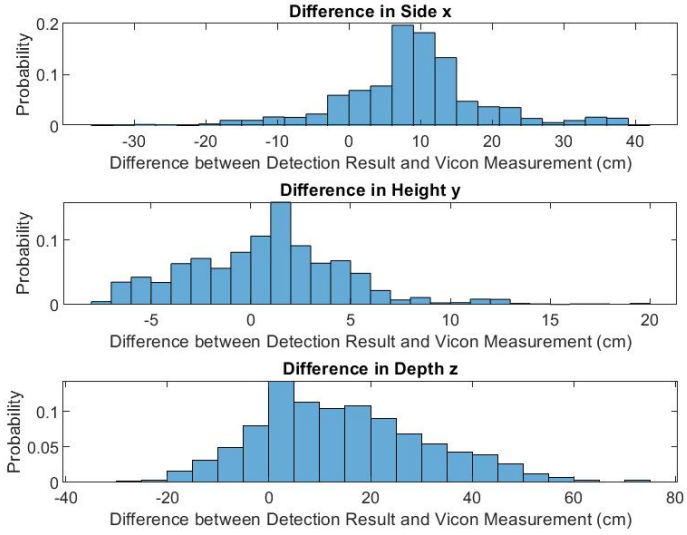

Figure B.45: SSD Inception v2 Pure Translation in Depth Direction Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

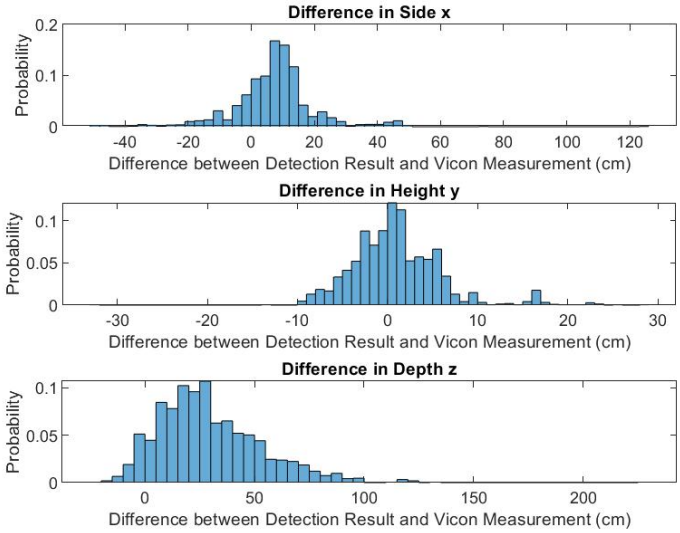

Figure B.46: SSD Inception v2 Pure Translation in Depth Direction Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

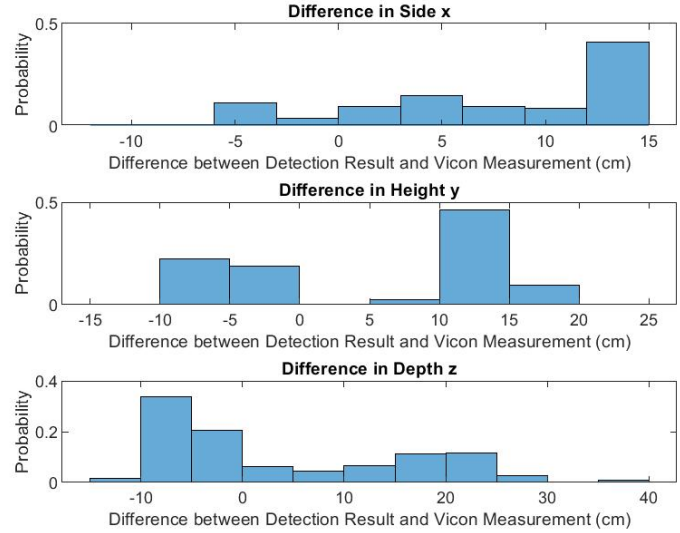

Figure B.47: SSD Inception v2 Pure Translation in Depth Direction Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

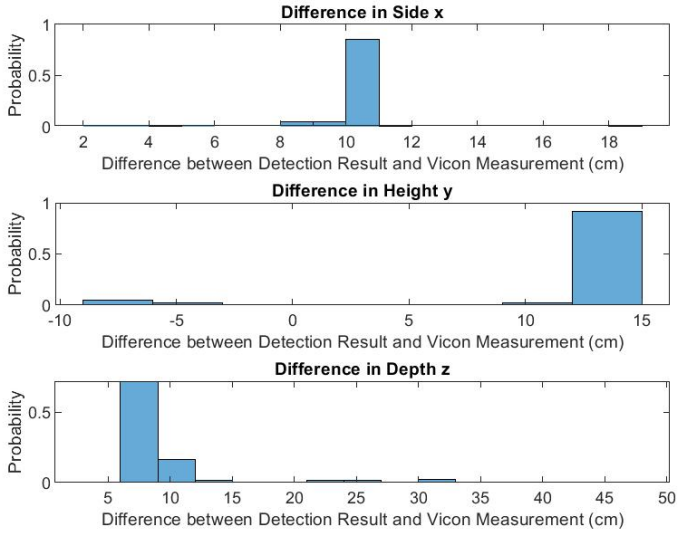

Figure B.48: SSD Inception v2 Pure Translation in Depth Direction Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

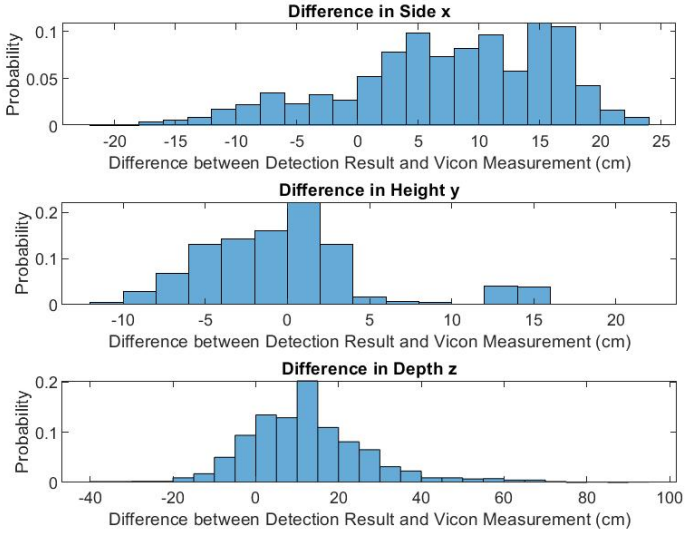

Figure B.49: SSD Inception v2 Pure Translation in Depth Direction Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

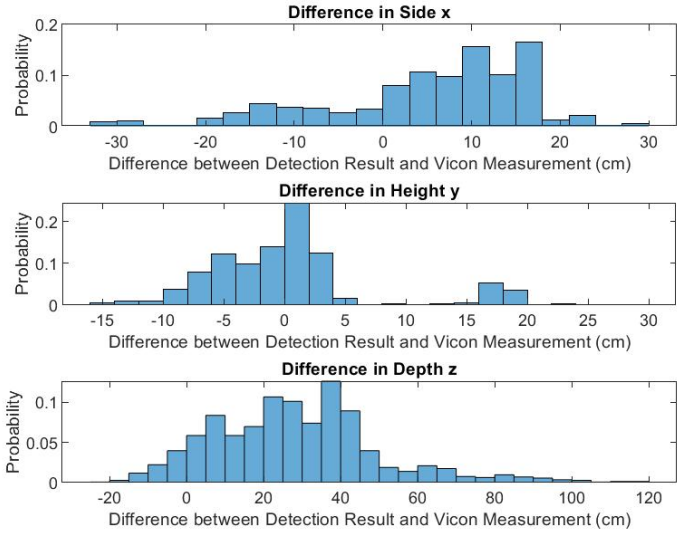

Figure B.50: SSD Inception v2 Pure Translation in Depth Direction Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

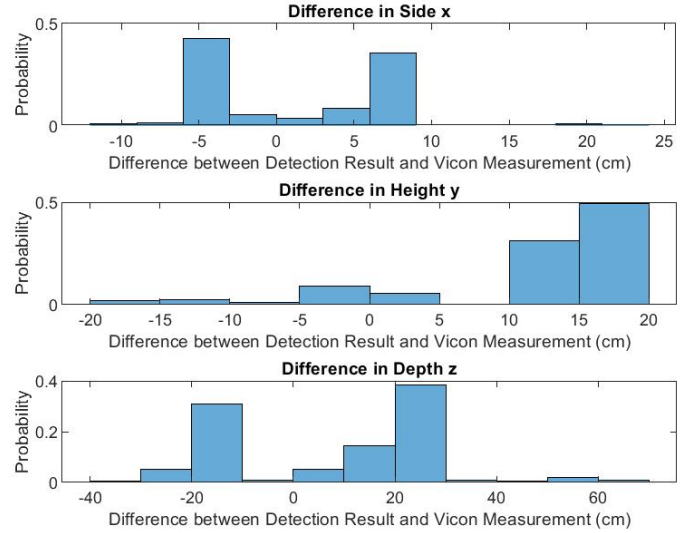

Figure B.51: SSD Inception v2 Pure Translation in Depth Direction Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

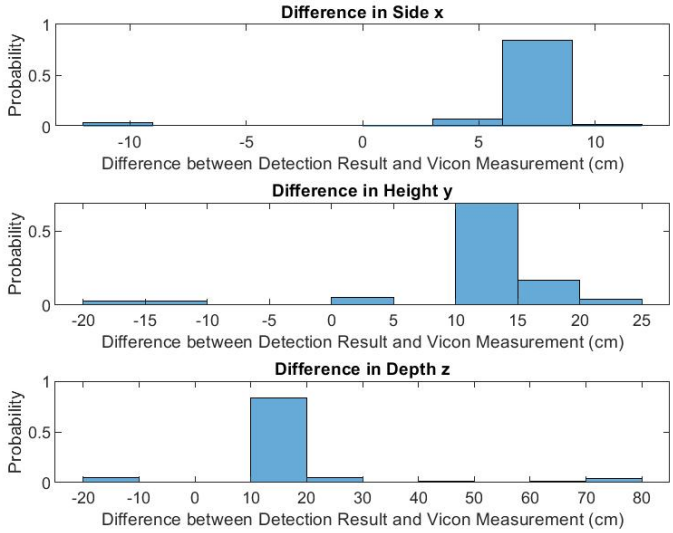

Figure B.52: SSD Inception v2 Pure Translation in Depth Direction Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

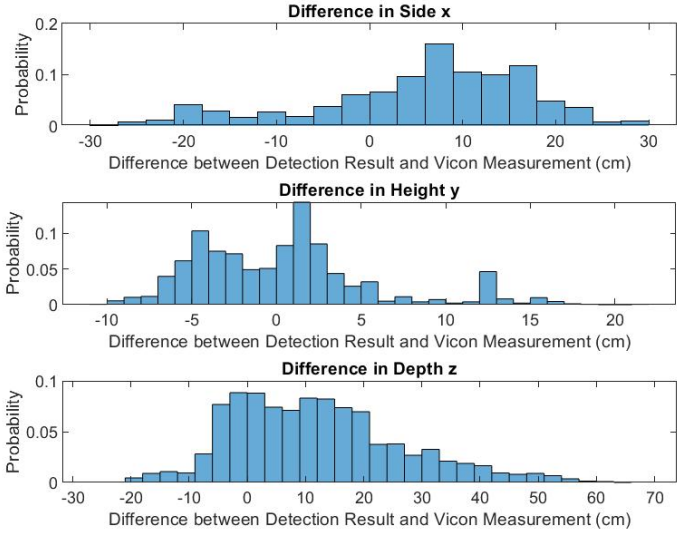

Figure B.53: SSD Inception v2 Pure Rotation Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

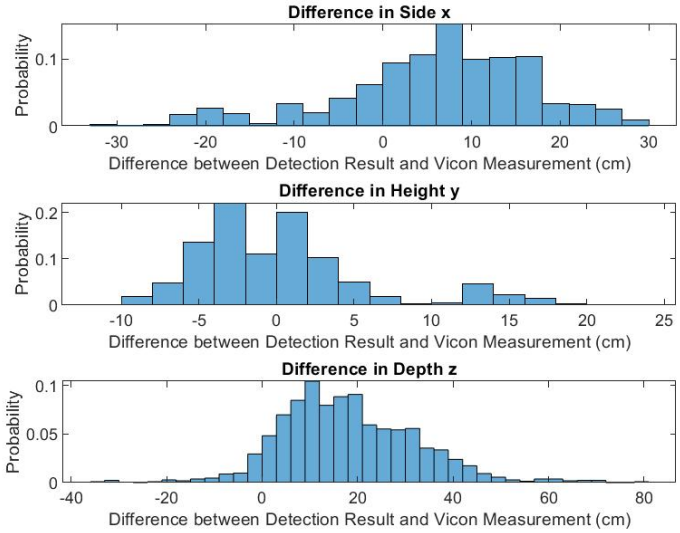

Figure B.54: SSD Inception v2 Pure Rotation Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

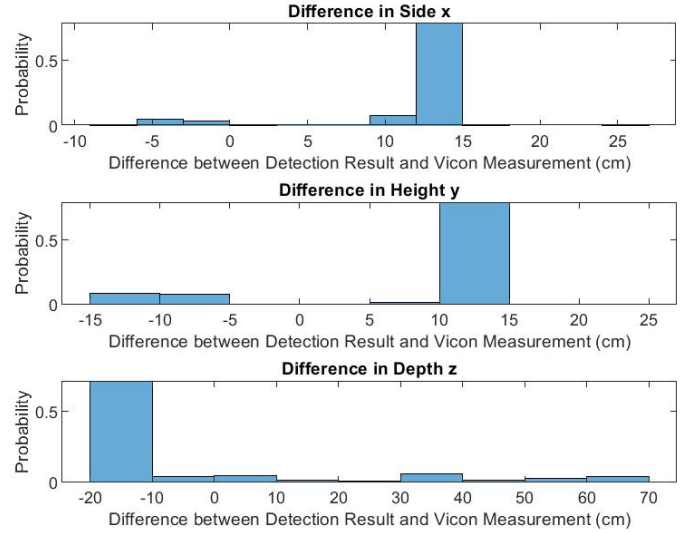

Figure B.55: SSD Inception v2 Pure Rotation Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

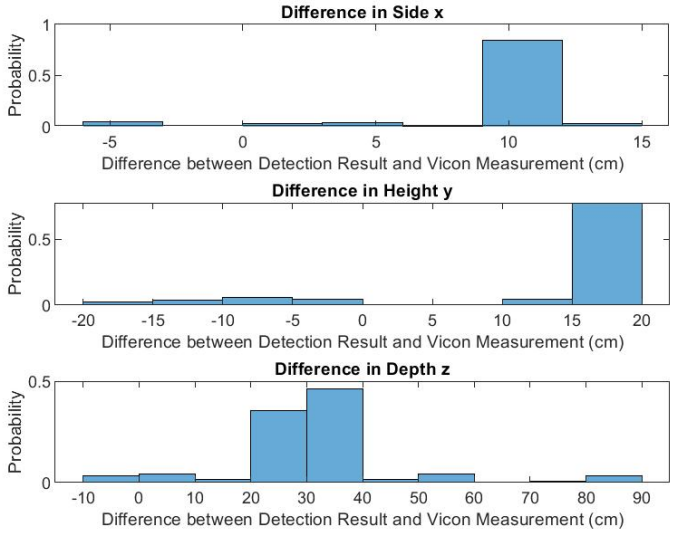

Figure B.56: SSD Inception v2 Pure Rotation Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

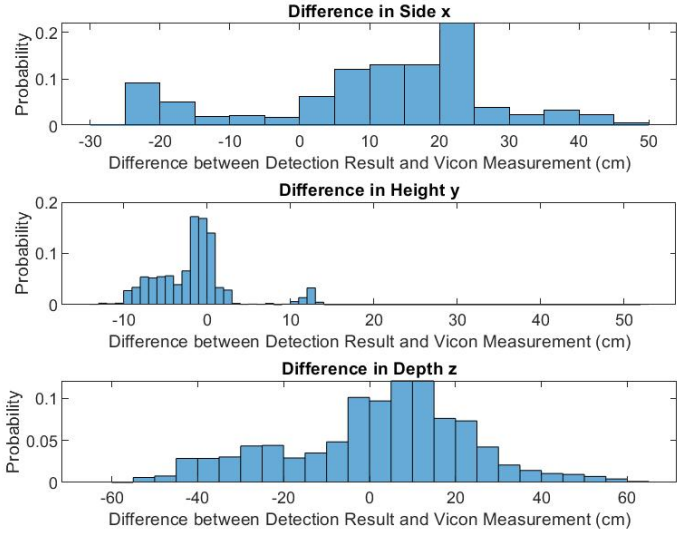

Figure B.57: SSD Inception v2 Pure Rotation Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

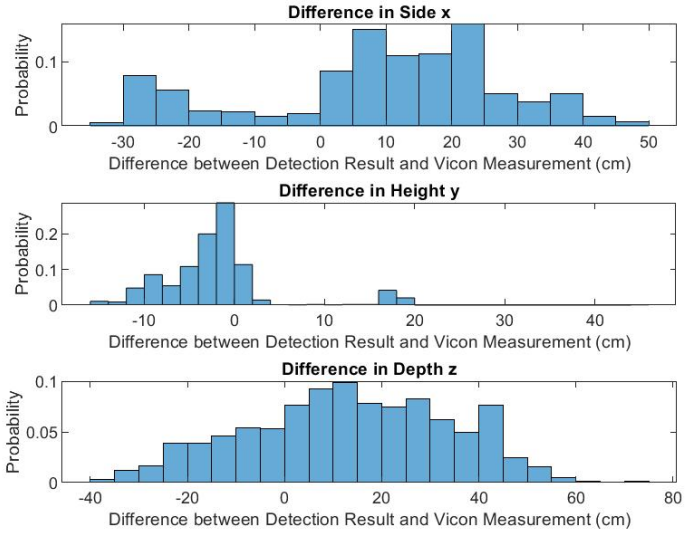

Figure B.58: SSD Inception v2 Pure Rotation Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

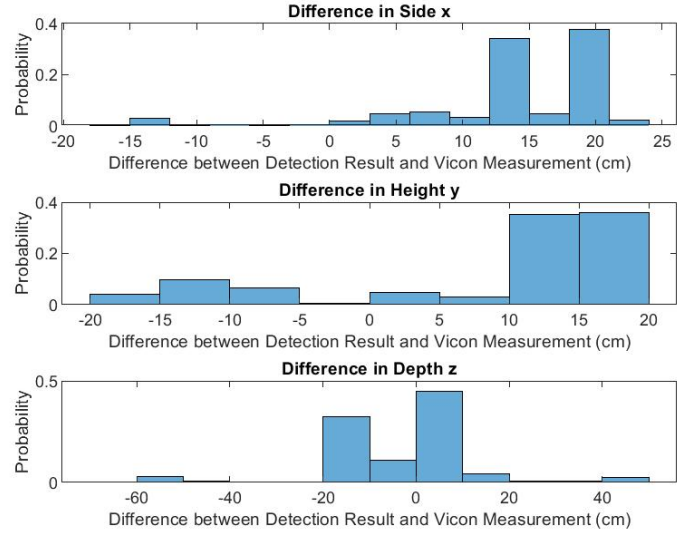

Figure B.59: SSD Inception v2 Pure Rotation Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

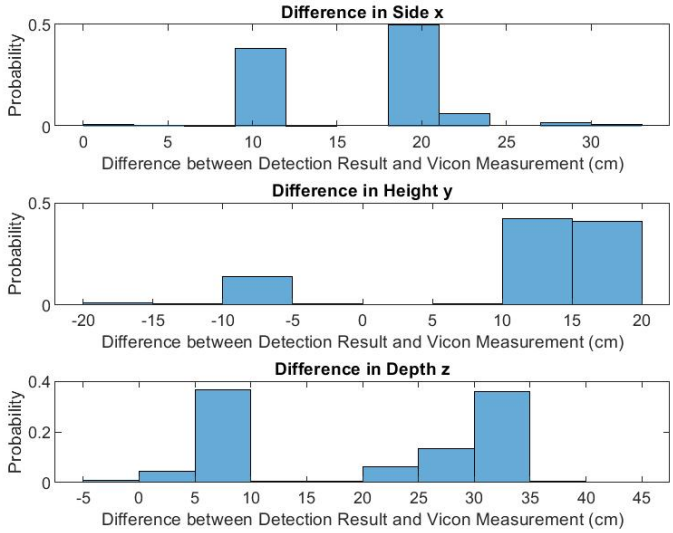

Figure B.60: SSD Inception v2 Pure Rotation Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

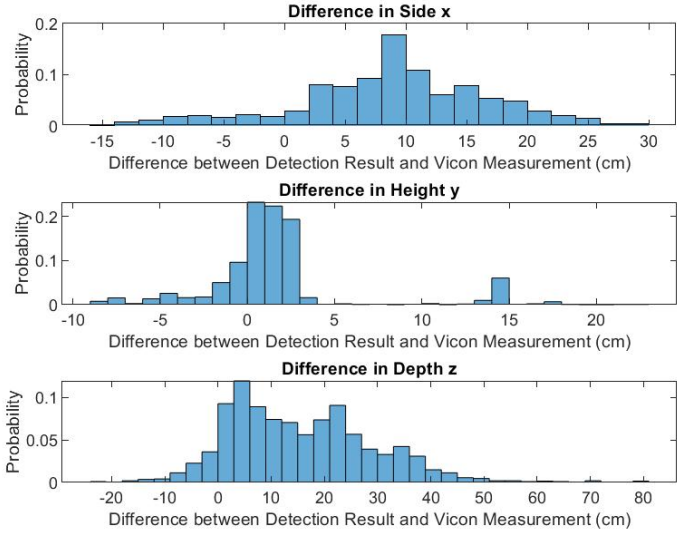

Figure B.61: SSD Inception v2 Random Flight Pattern Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

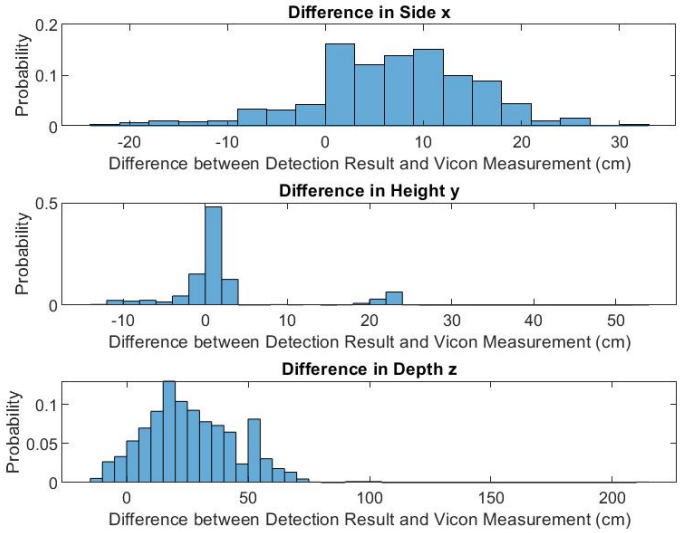

Figure B.62: SSD Inception v2 Random Flight Pattern Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

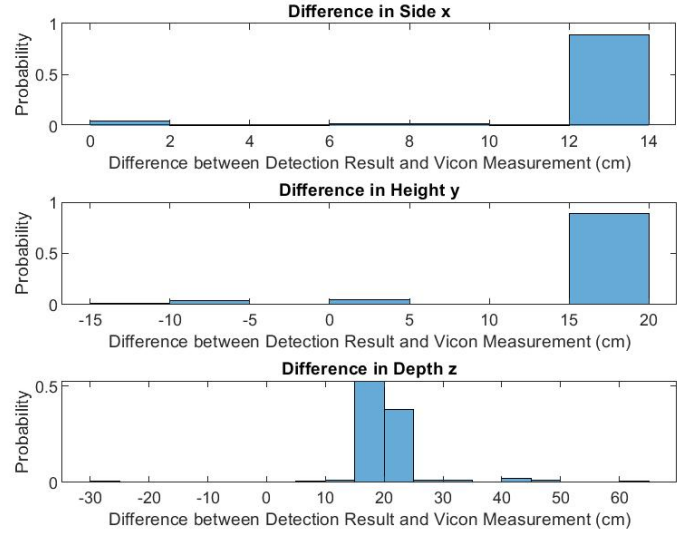

Figure B.63: SSD Inception v2 Random Flight Pattern Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

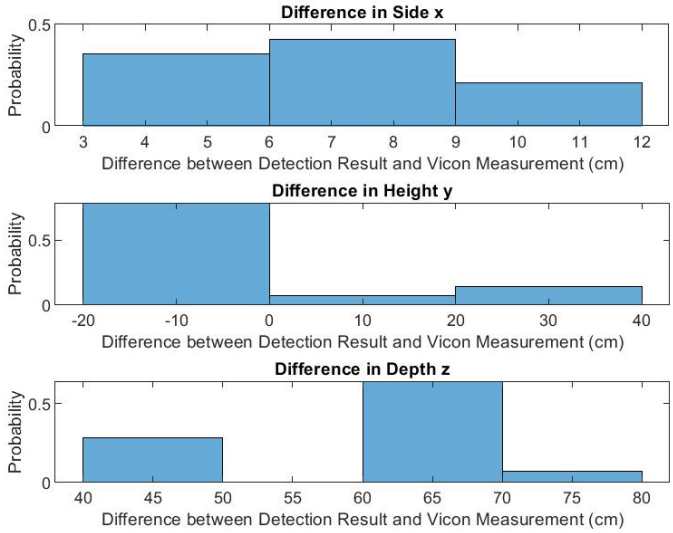

Figure B.64: SSD Inception v2 Random Flight Pattern Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

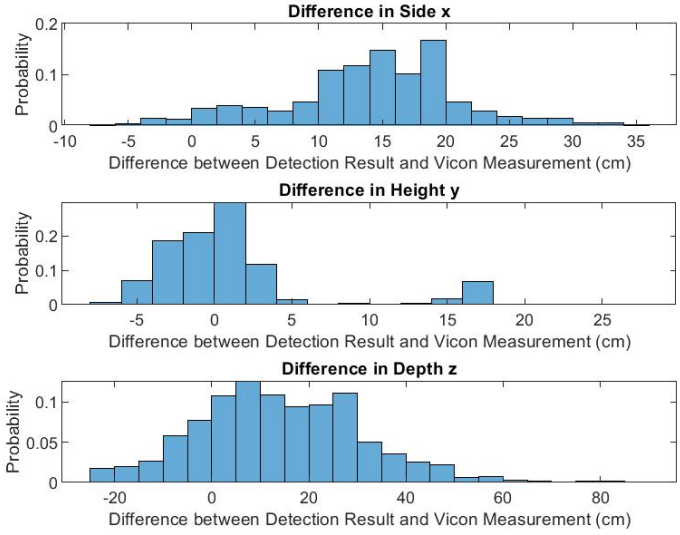

Figure B.65: SSD Inception v2 Random Flight Pattern Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

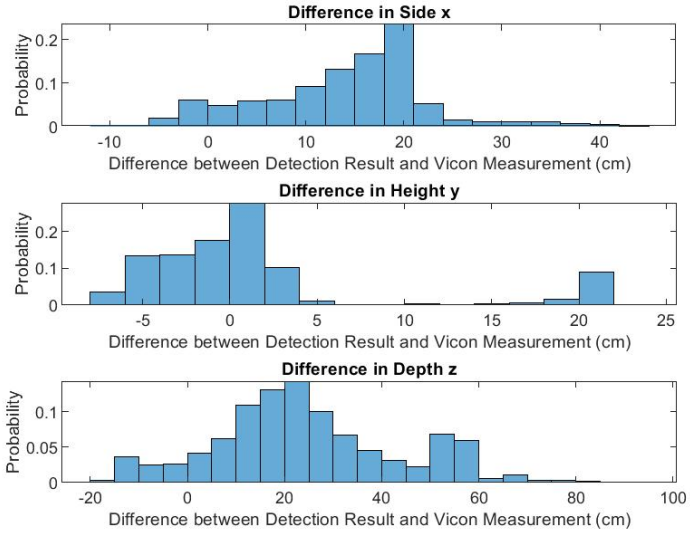

Figure B.66: SSD Inception v2 Random Flight Pattern Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

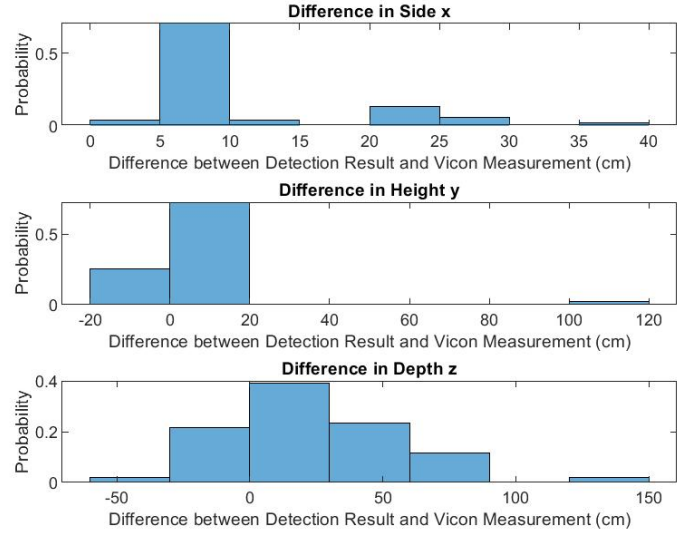

Figure B.67: SSD Inception v2 Random Flight Pattern Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

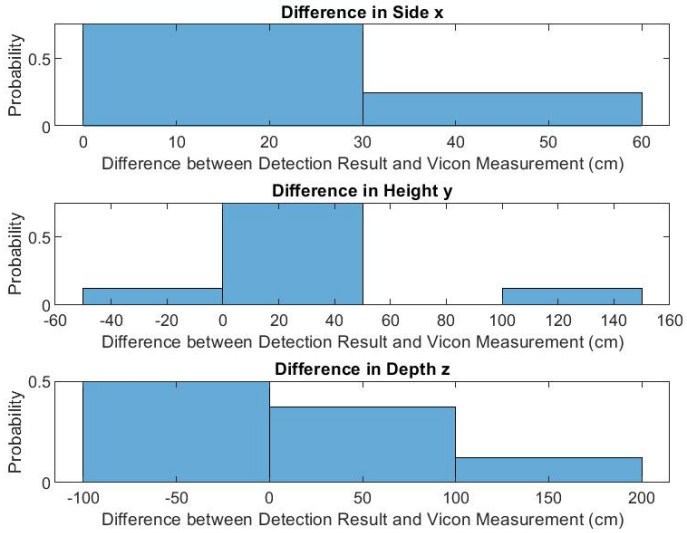

Figure B.68: SSD Inception v2 Random Flight Pattern Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

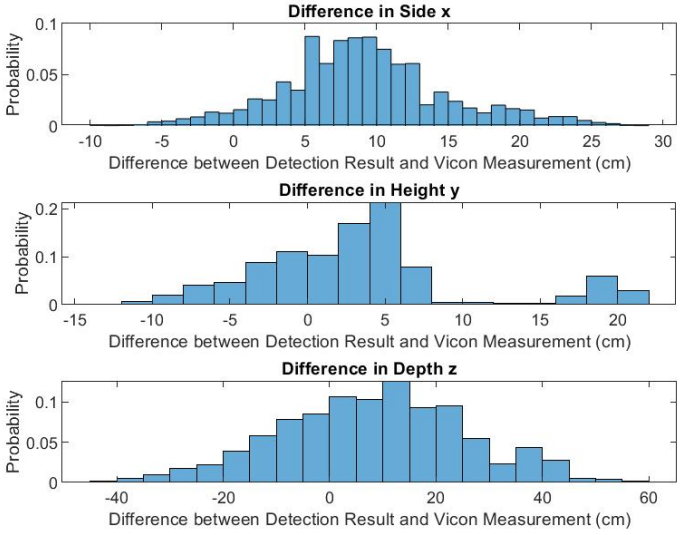

Figure B.69: Faster RCNN Inception v2 Pure Translation in Side and Height Direction Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

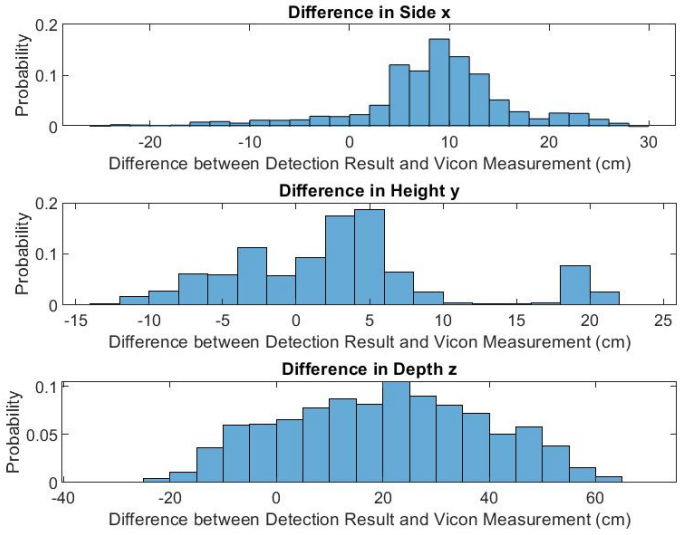

Figure B.70: Faster RCNN Inception v2 Pure Translation in Side and Height Direction Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

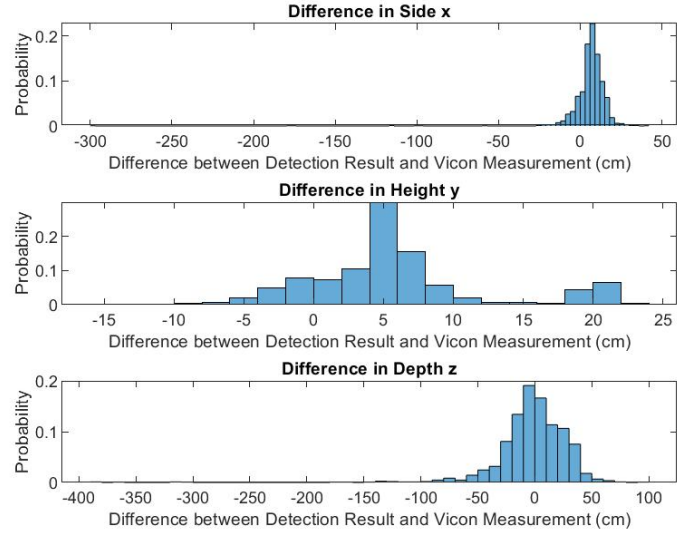

Figure B.71: Faster RCNN Inception v2 Pure Translation in Side and Height Direction Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

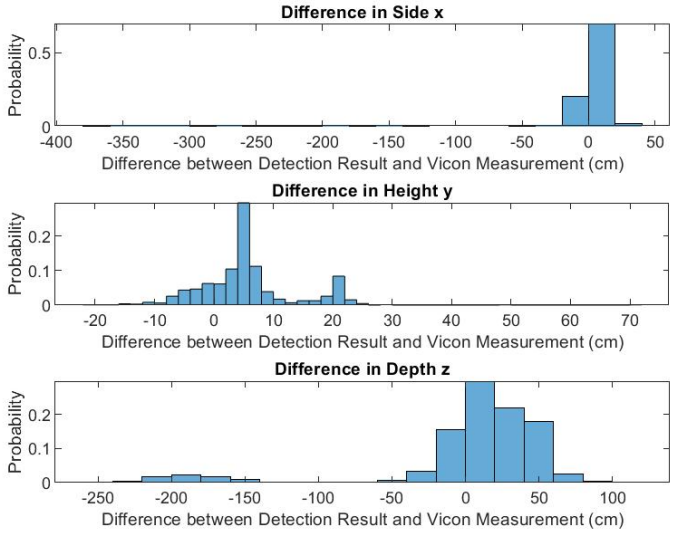

Figure B.72: Faster RCNN Inception v2 Pure Translation in Side and Height Direction Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

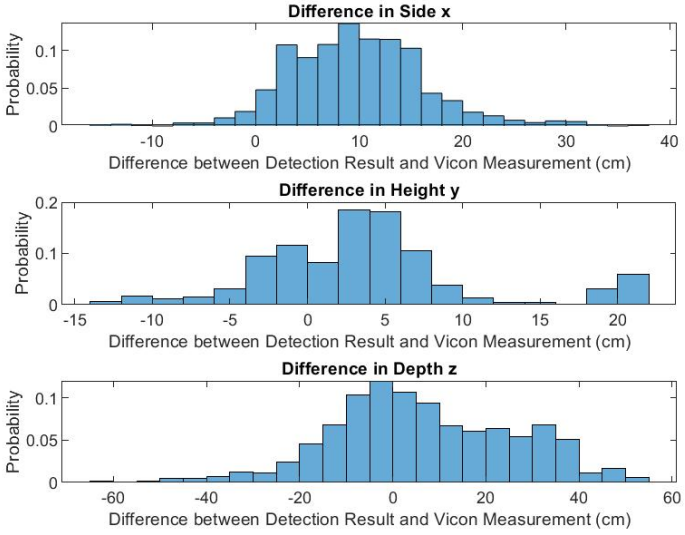

Figure B.73: Faster RCNN Inception v2 Pure Translation in Side and Height Direction Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

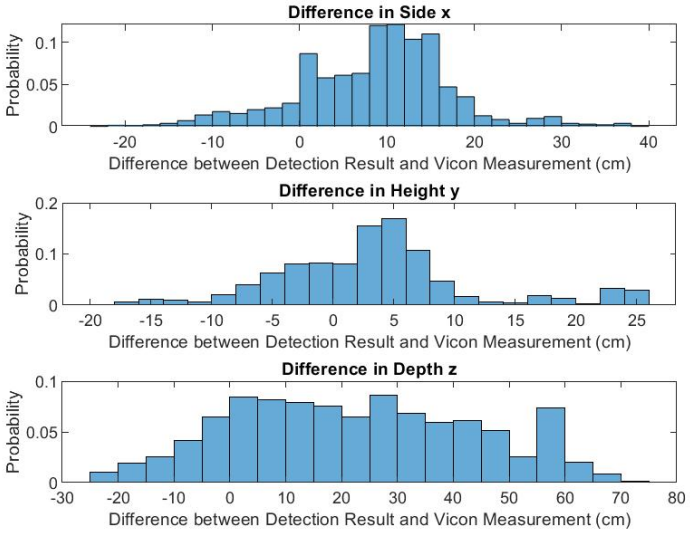

Figure B.74: Faster RCNN Inception v2 Pure Translation in Side and Height Direction Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

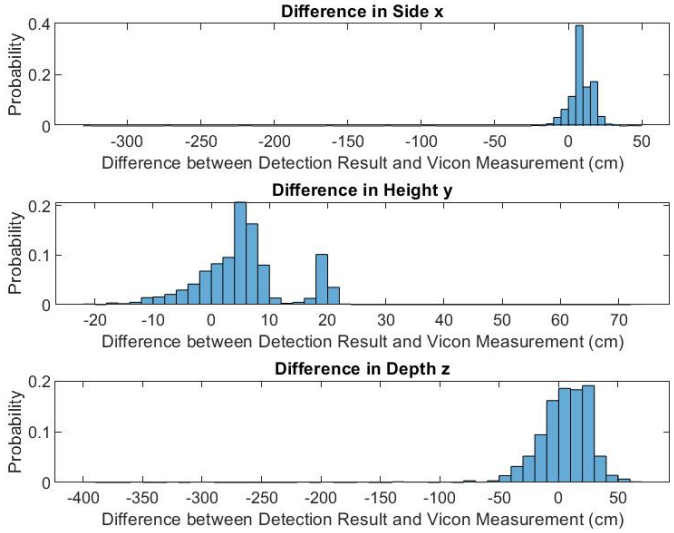

Figure B.75: Faster RCNN Inception v2 Pure Translation in Side and Height Direction Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

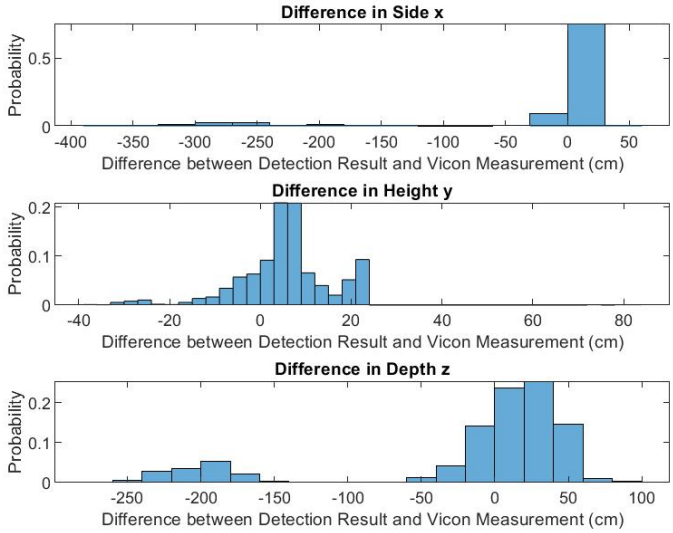

Figure B.76: Faster RCNN Inception v2 Pure Translation in Side and Height Direction Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

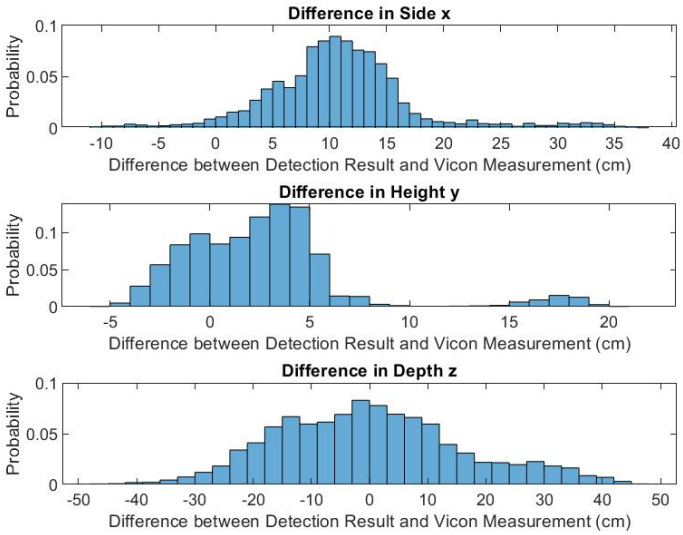

Figure B.77: Faster RCNN Inception v2 Pure Translation in Depth Direction Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

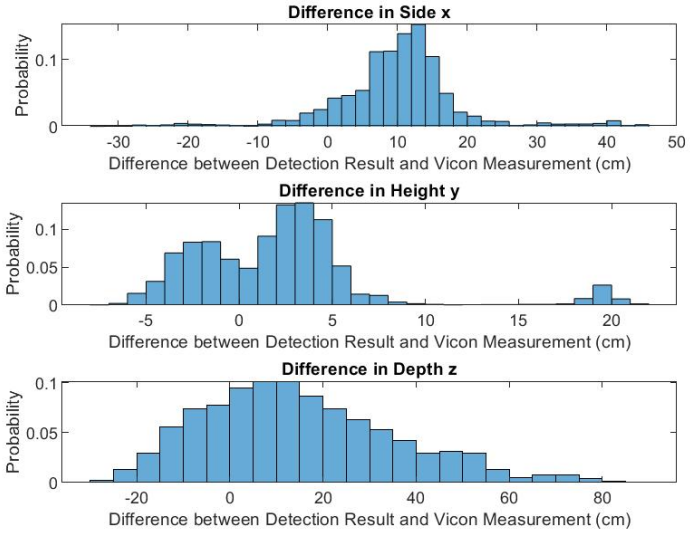

Figure B.78: Faster RCNN Inception v2 Pure Translation in Depth Direction Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

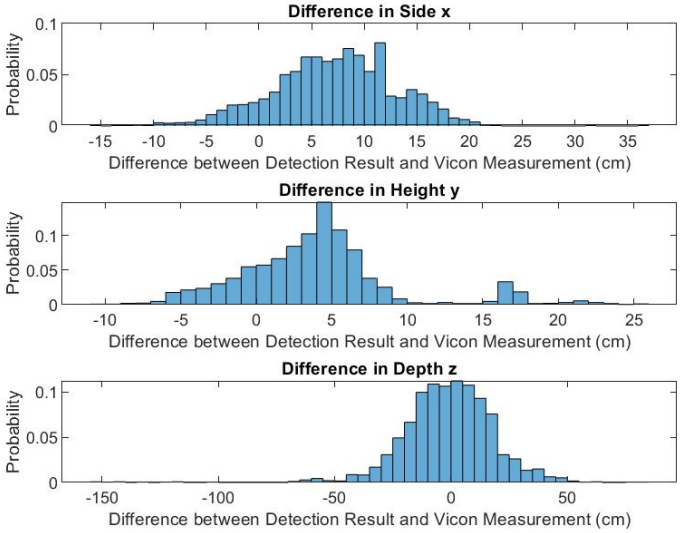

Figure B.79: Faster RCNN Inception v2 Pure Translation in Depth Direction Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

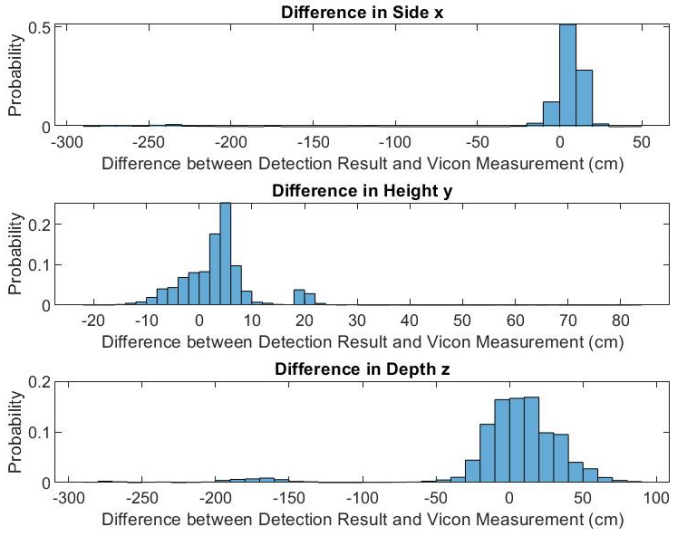

Figure B.80: Faster RCNN Inception v2 Pure Translation in Depth Direction Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

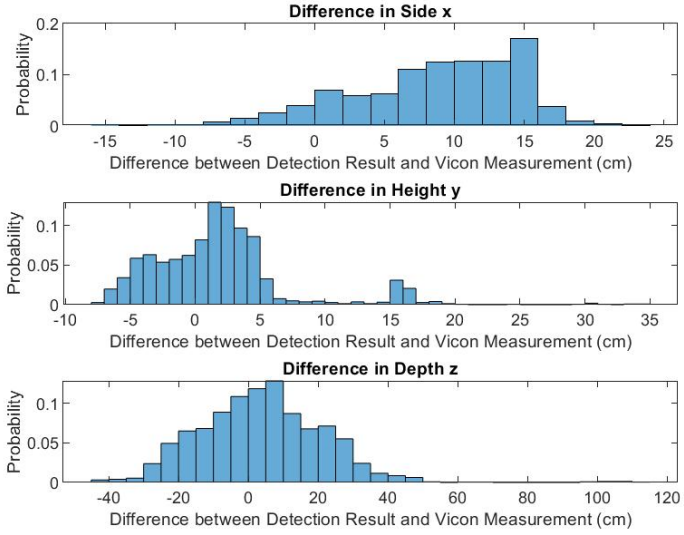

Figure B.81: Faster RCNN Inception v2 Pure Translation in Depth Direction Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

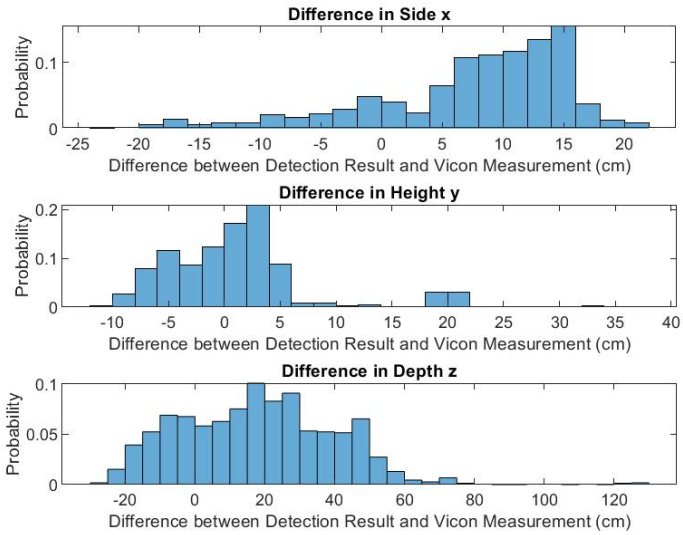

Figure B.82: Faster RCNN Inception v2 Pure Translation in Depth Direction Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

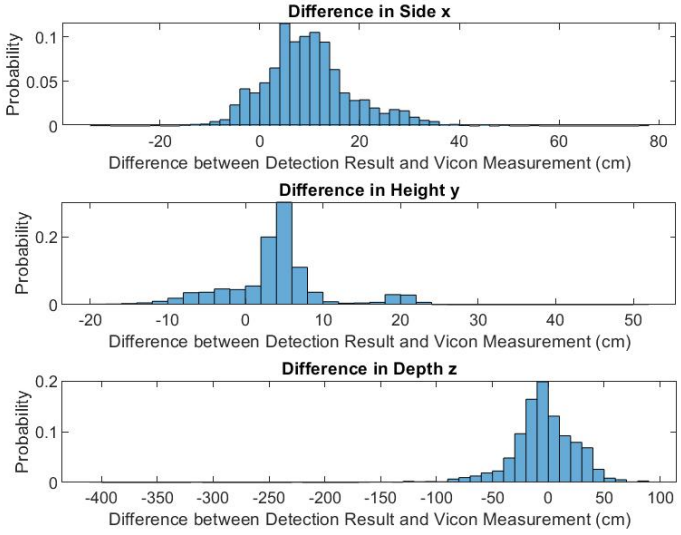

Figure B.83: Faster RCNN Inception v2 Pure Translation in Depth Direction Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

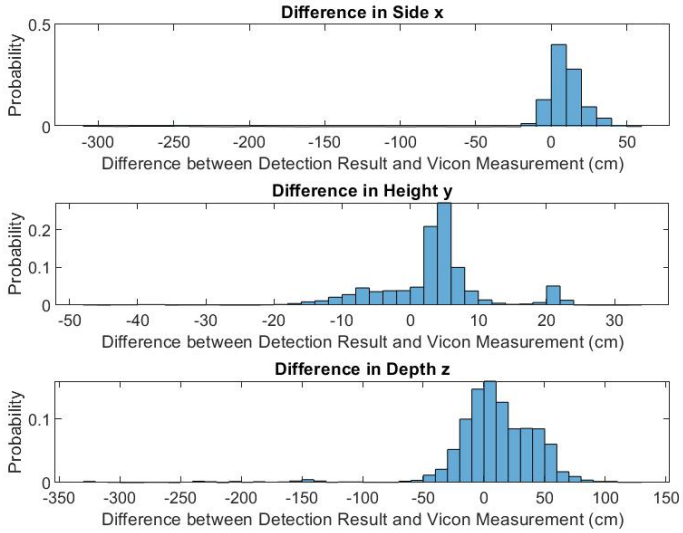

Figure B.84: Faster RCNN Inception v2 Pure Translation in Depth Direction Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

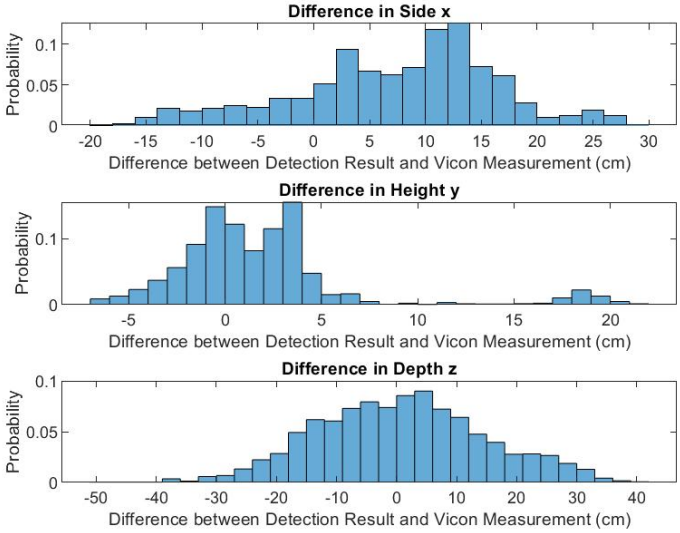

Figure B.85: Faster RCNN Inception v2 Pure Rotation Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

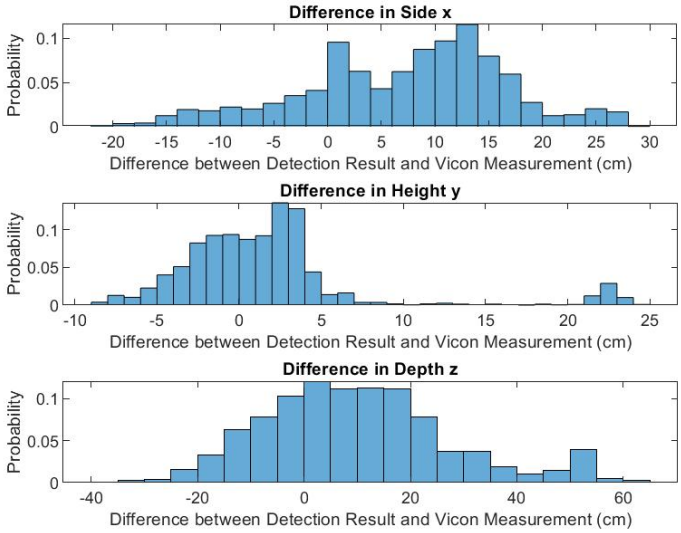

Figure B.86: Faster RCNN Inception v2 Pure Rotation Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

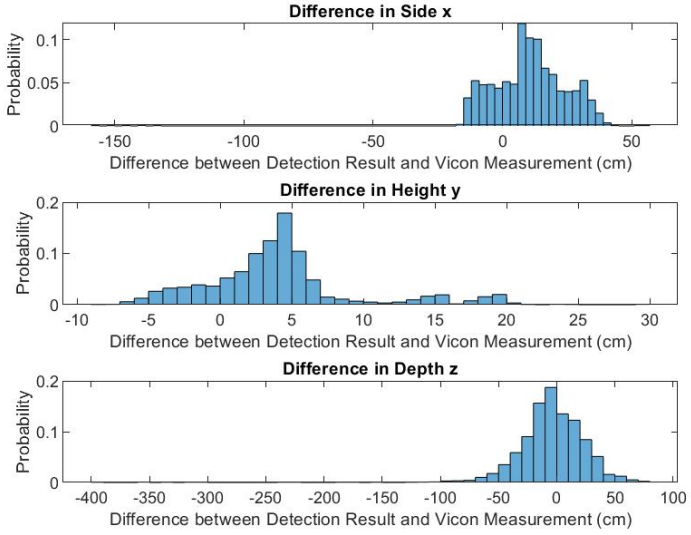

Figure B.87: Faster RCNN Inception v2 Pure Rotation Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

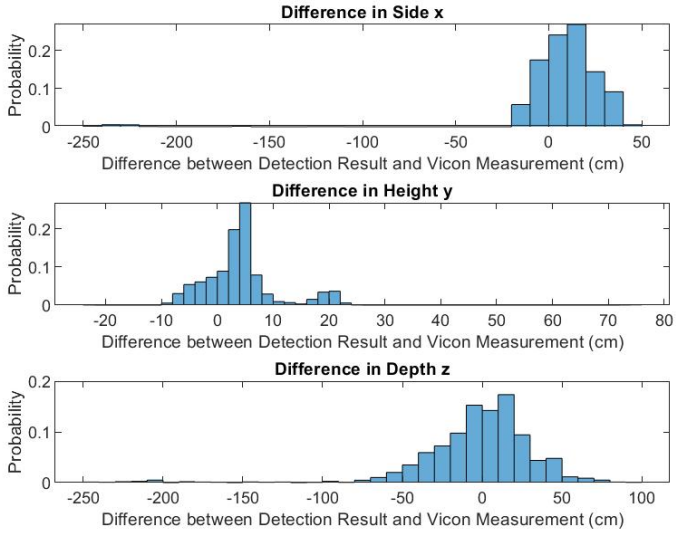

Figure B.88: Faster RCNN Inception v2 Pure Rotation Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

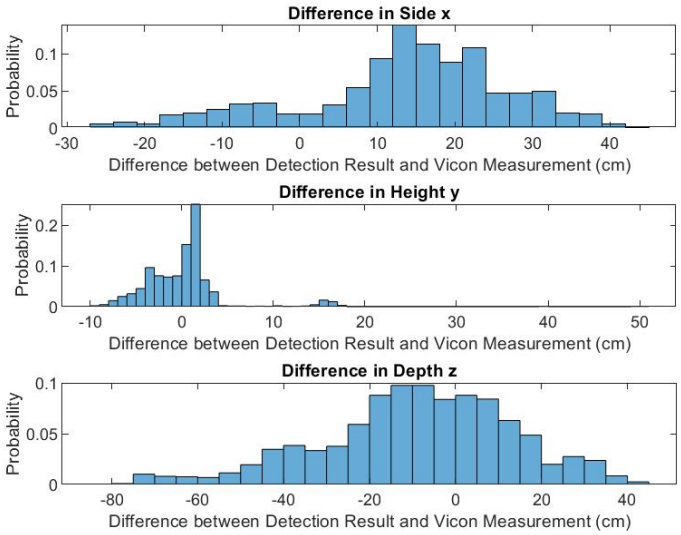

Figure B.89: Faster RCNN Inception v2 Pure Rotation Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

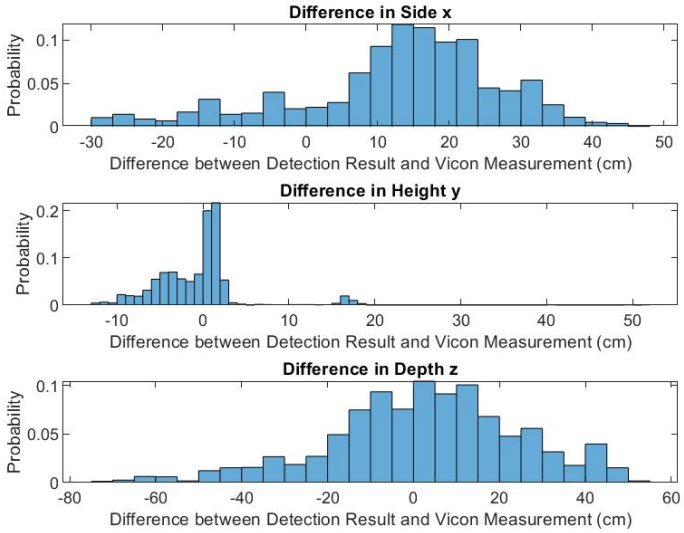

Figure B.90: Faster RCNN Inception v2 Pure Rotation Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

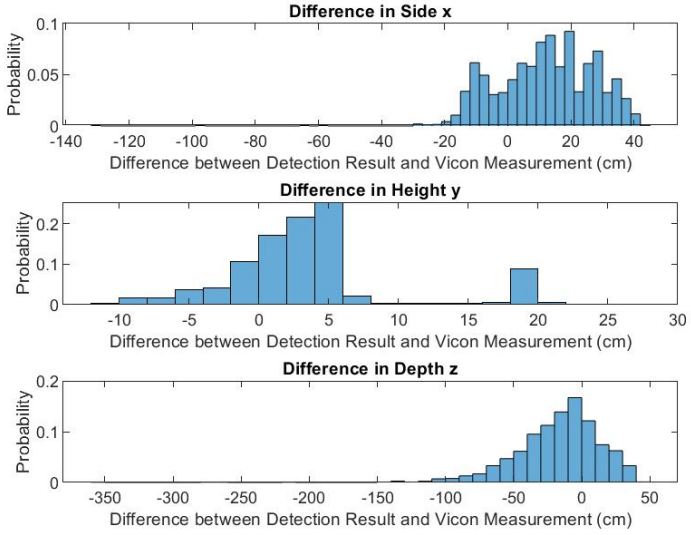

Figure B.91: Faster RCNN Inception v2 Pure Rotation Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

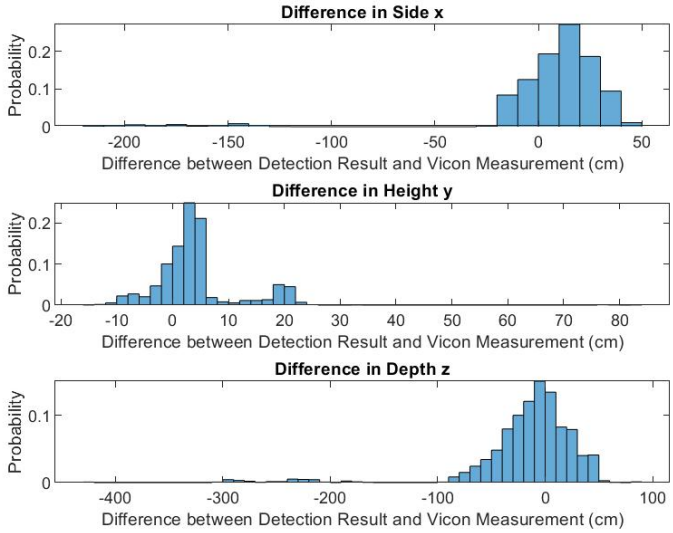

Figure B.92: Faster RCNN Inception v2 Pure Rotation Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

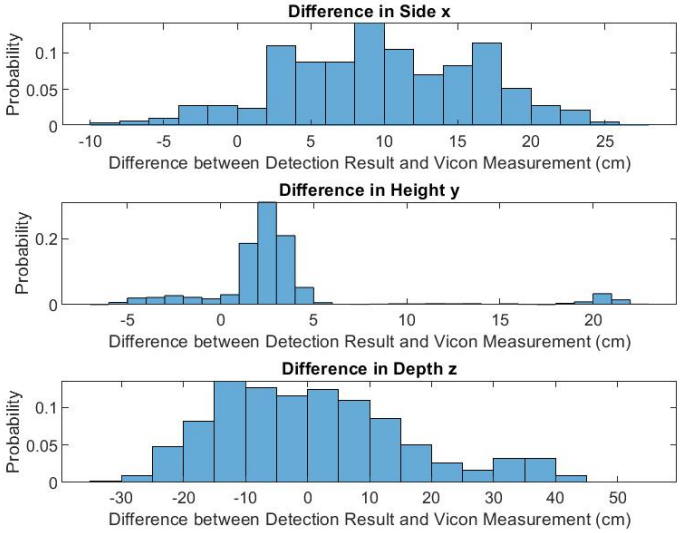

Figure B.93: Faster RCNN Inception v2 Random Flight Pattern Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

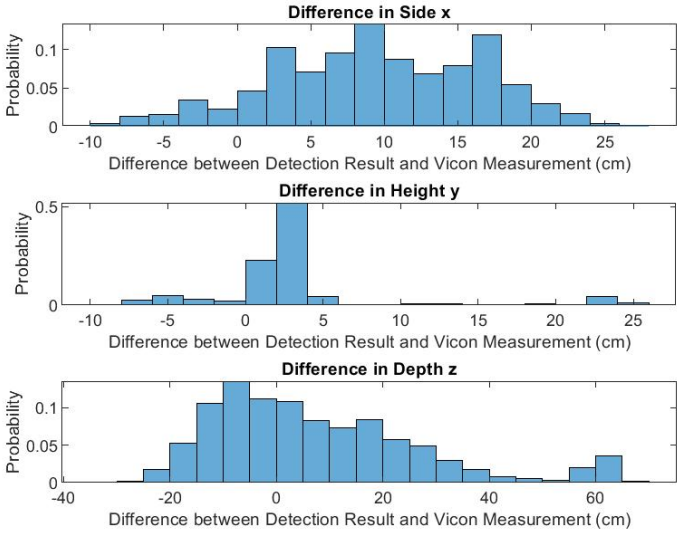

Figure B.94: Faster RCNN Inception v2 Random Flight Pattern Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

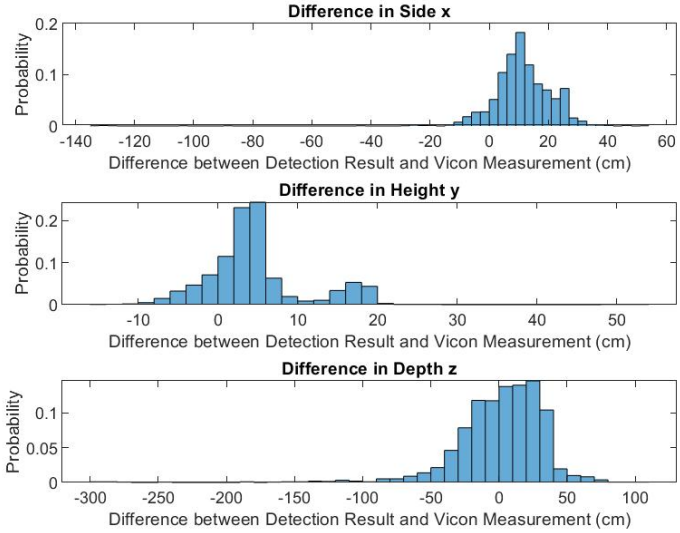

Figure B.95: Faster RCNN Inception v2 Random Flight Pattern Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

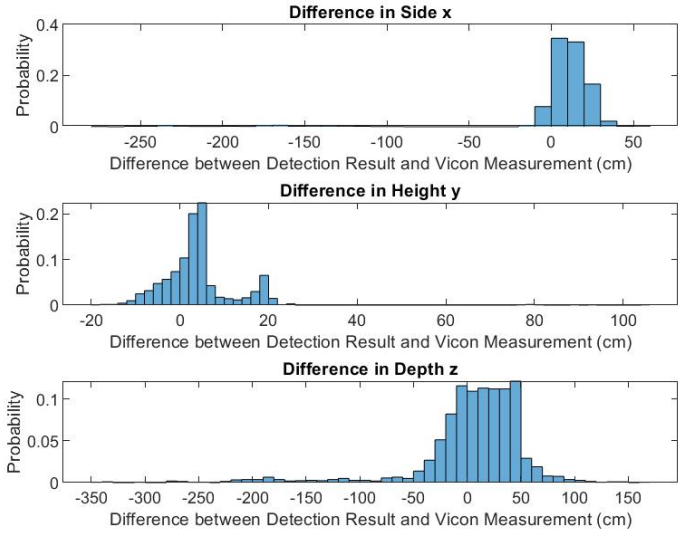

Figure B.96: Faster RCNN Inception v2 Random Flight Pattern Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

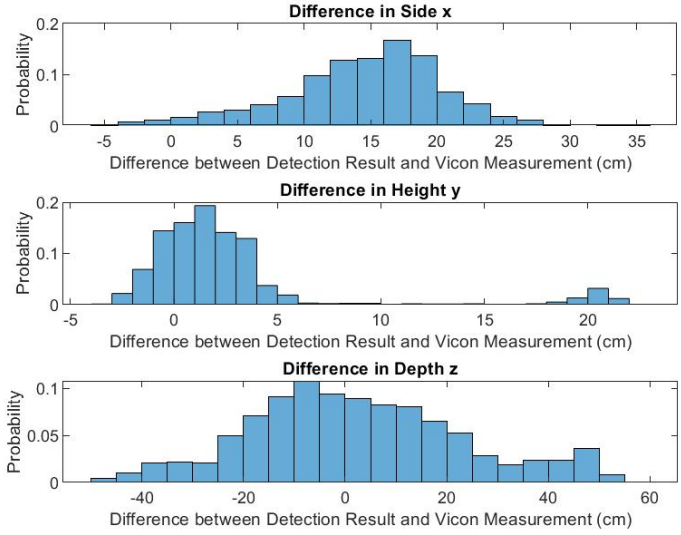

Figure B.97: Faster RCNN Inception v2 Random Flight Pattern Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

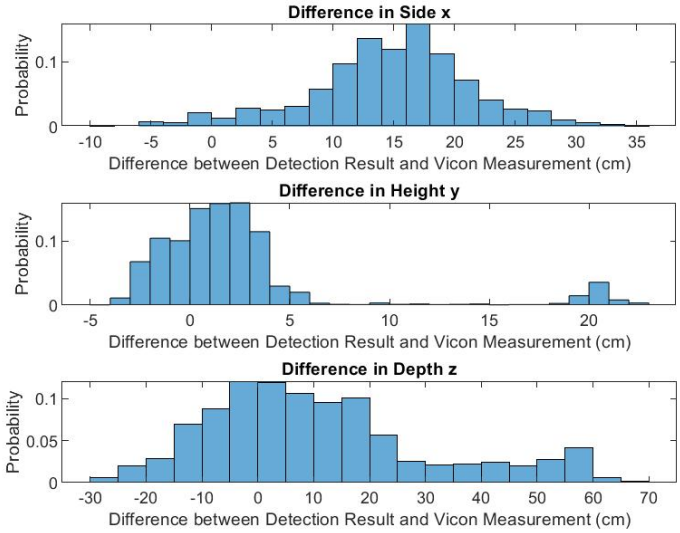

Figure B.98: Faster RCNN Inception v2 Random Flight Pattern Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

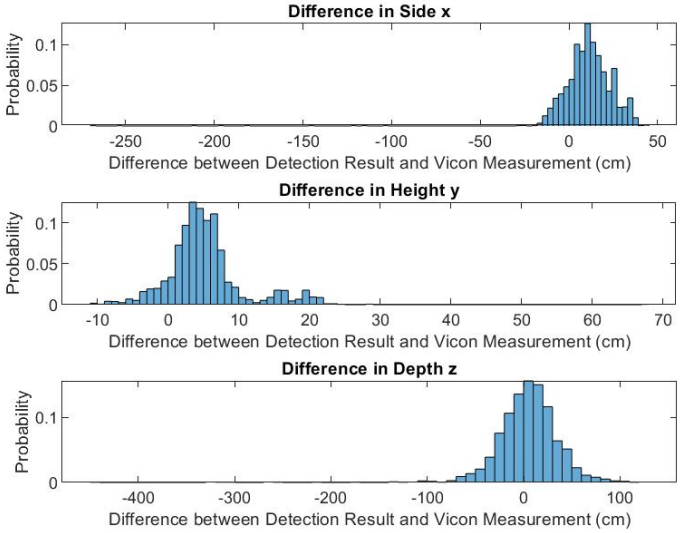

Figure B.99: Faster RCNN Inception v2 Random Flight Pattern Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

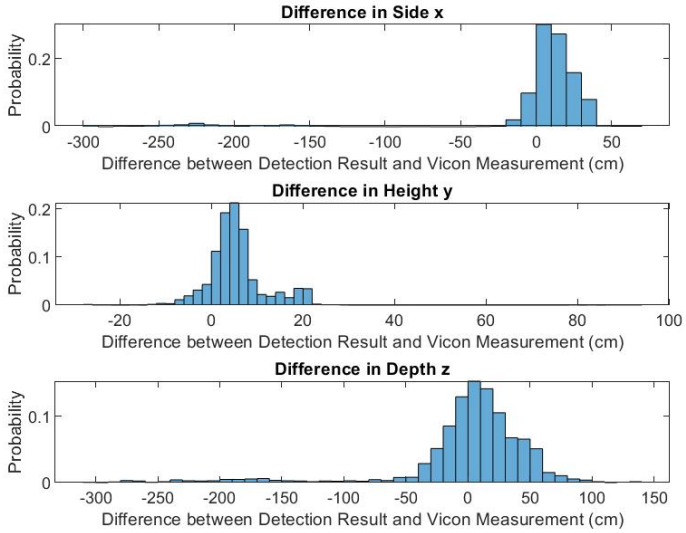

Figure B.100: Faster RCNN Inception v2 Random Flight Pattern Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

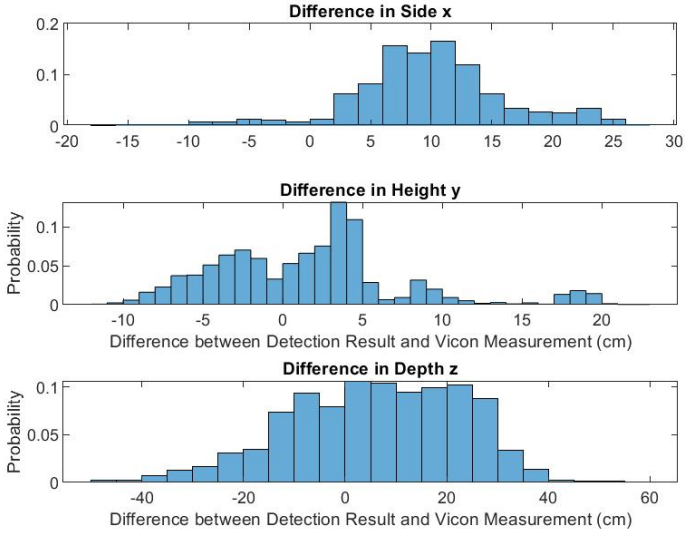

Figure B.101: YOLO v2 Pure Translation in Side and Height Direction Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

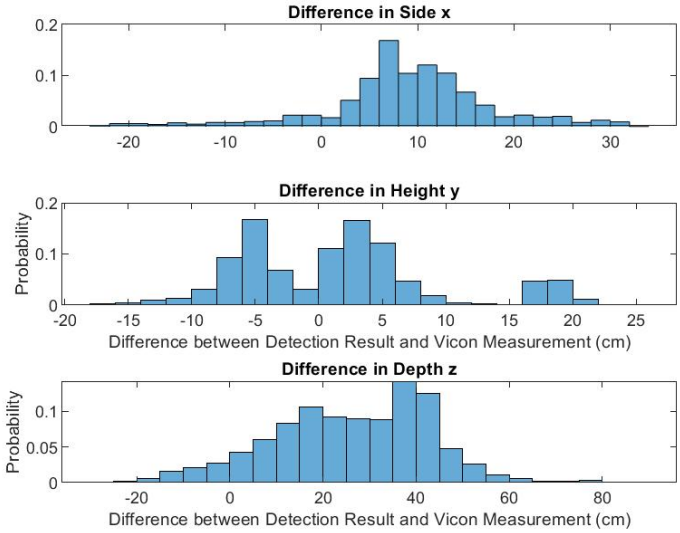

Figure B.102: YOLO v2 Pure Translation in Side and Height Direction Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

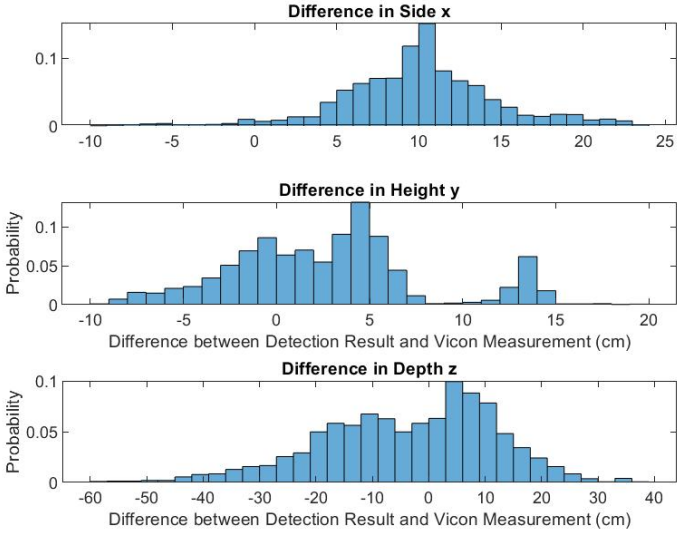

Figure B.103: YOLO v2 Pure Translation in Side and Height Direction Test 1 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

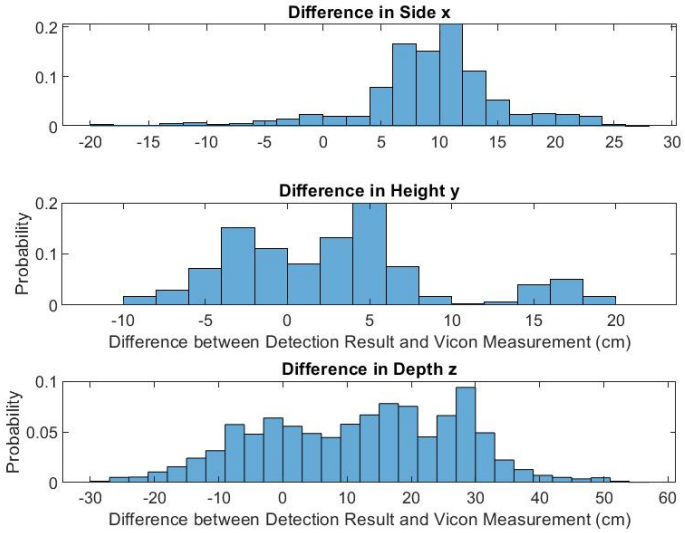

Figure B.104: YOLO v2 Pure Translation in Side and Height Direction Test 1 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

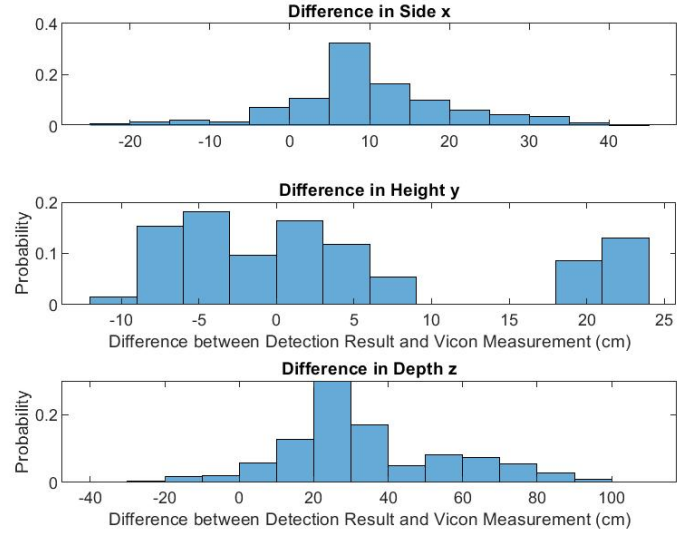

Figure B.105: YOLO v2 Pure Translation in Side and Height Direction Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

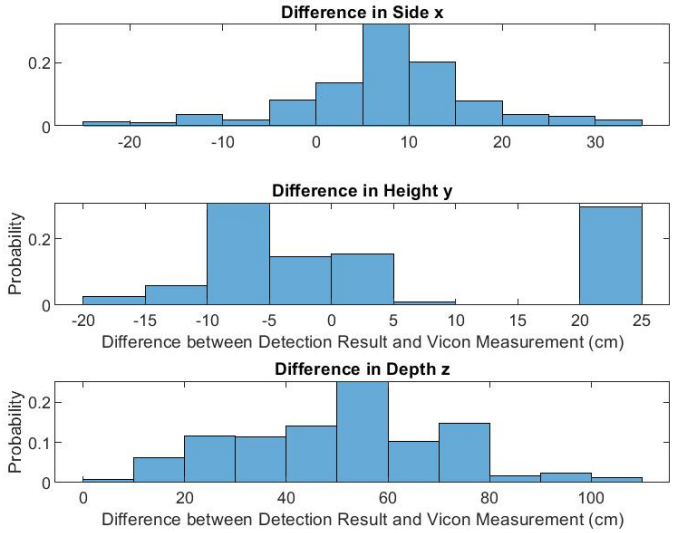

Figure B.106: YOLO v2 Pure Translation in Side and Height Direction Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

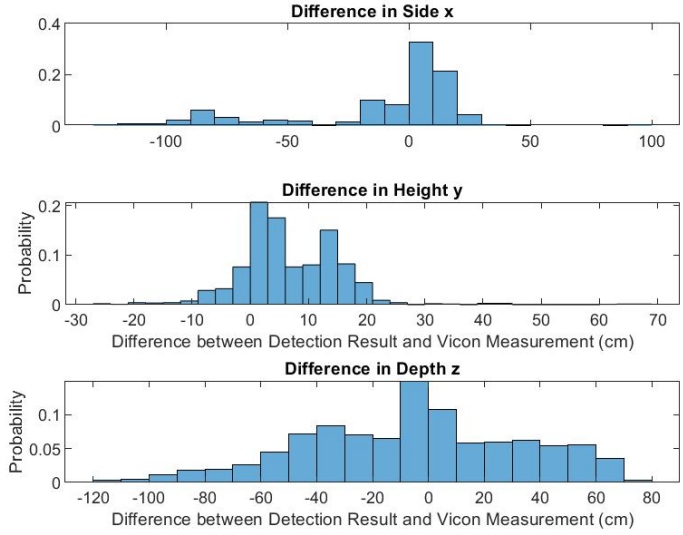

Figure B.107: YOLO v2 Pure Translation in Side and Height Direction Test 1 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

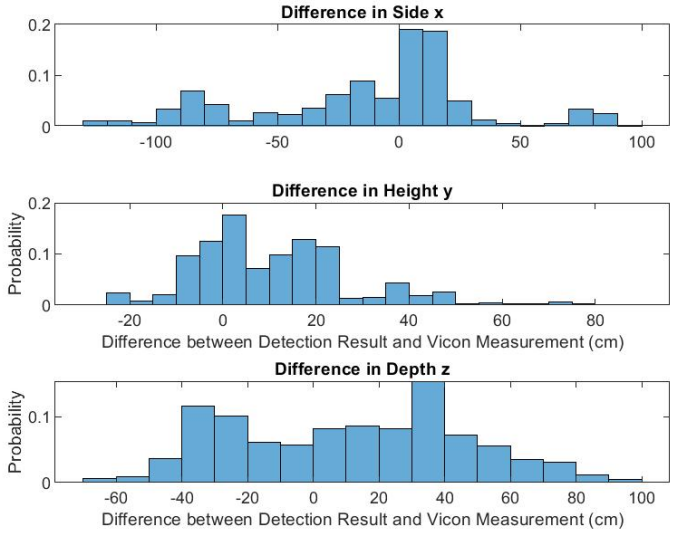

Figure B.108: YOLO v2 Pure Translation in Side and Height Direction Test 1 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

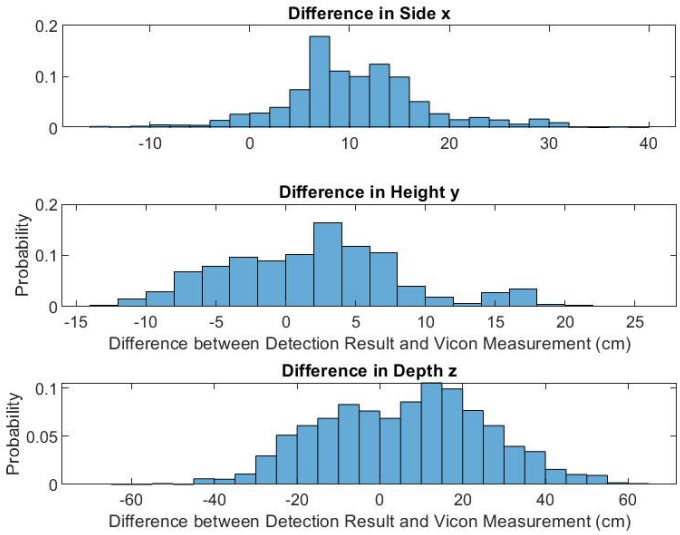

Figure B.109: YOLO v2 Pure Translation in Side and Height Direction Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

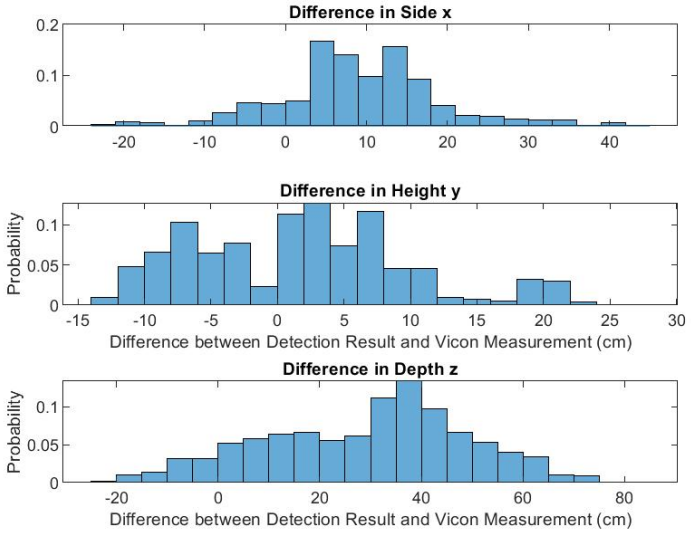

Figure B.110: YOLO v2 Pure Translation in Side and Height Direction Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

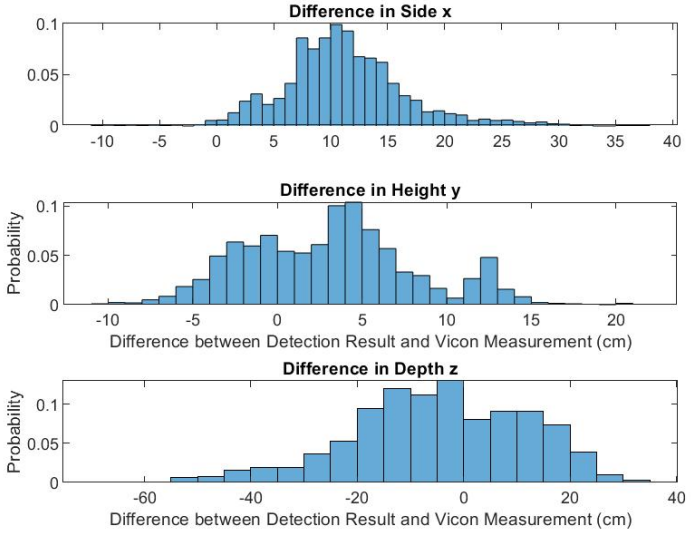

Figure B.111: YOLO v2 Pure Translation in Side and Height Direction Test 2 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

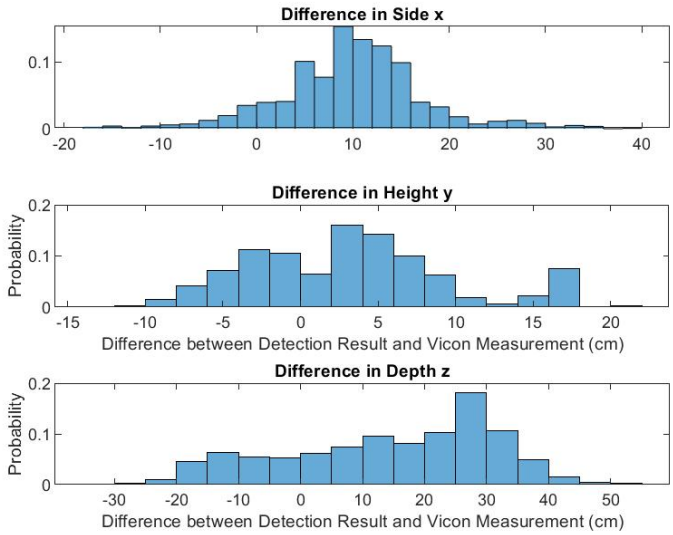

Figure B.112: YOLO v2 Pure Translation in Side and Height Direction Test 2 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

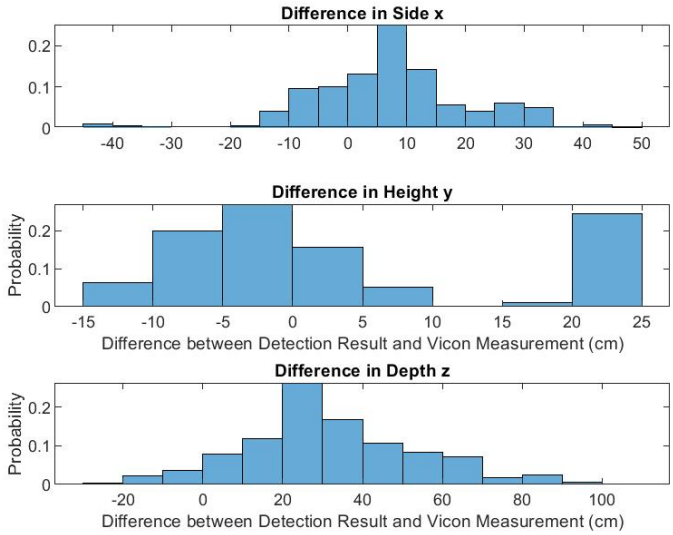

Figure B.113: YOLO v2 Pure Translation in Side and Height Direction Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

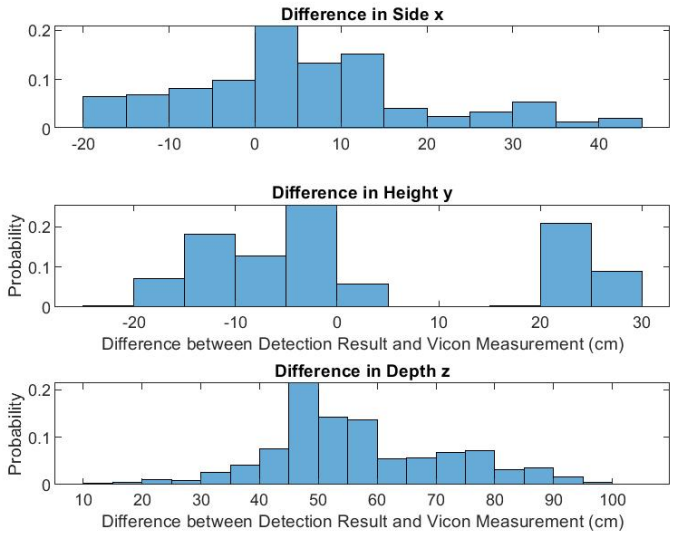

Figure B.114: YOLO v2 Pure Translation in Side and Height Direction Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

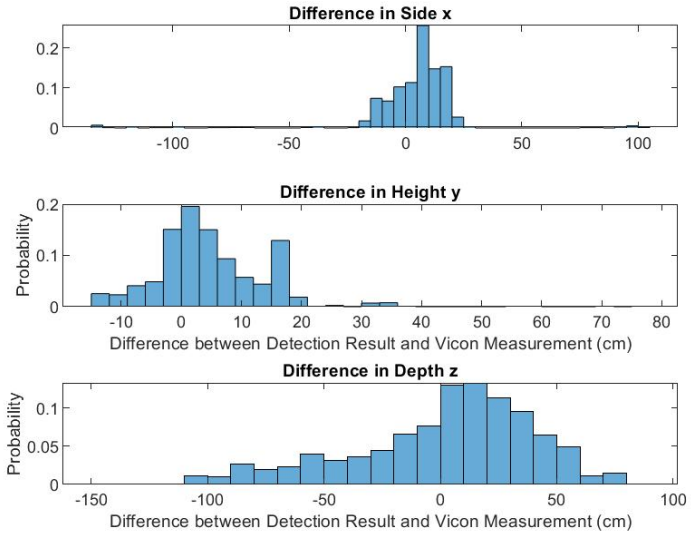

Figure B.115: YOLO v2 Pure Translation in Side and Height Direction Test 2 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

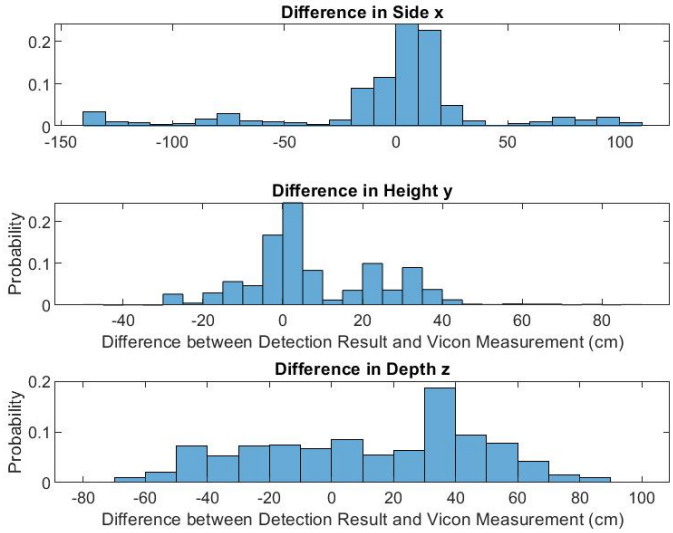

Figure B.116: YOLO v2 Pure Translation in Side and Height Direction Test 2 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

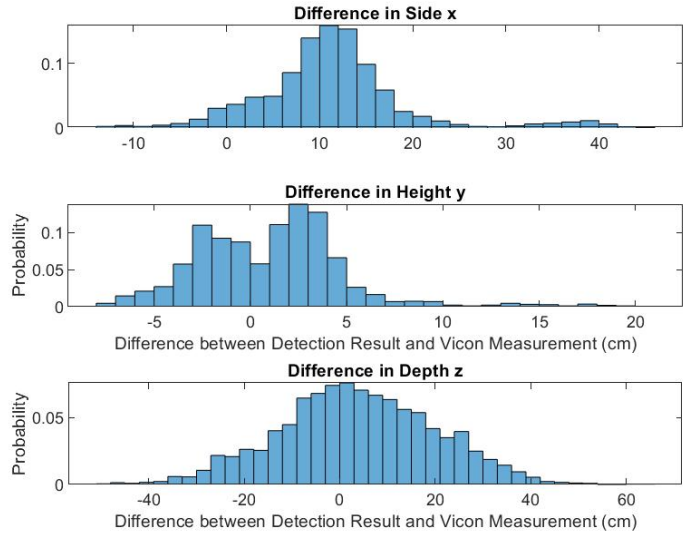

Figure B.117: YOLO v2 Pure Translation in Depth Direction Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

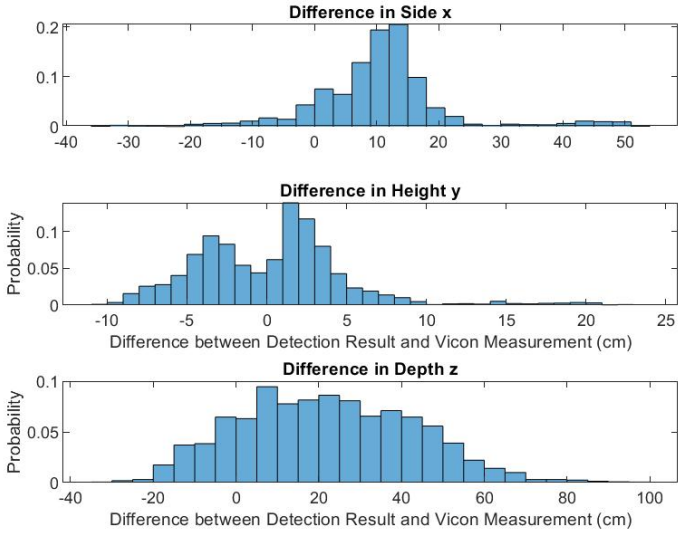

Figure B.118: YOLO v2 Pure Translation in Depth Direction Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

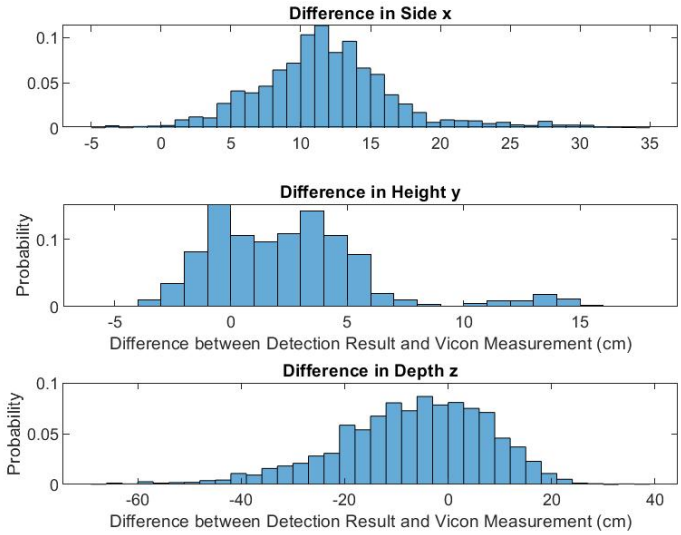

Figure B.119: YOLO v2 Pure Translation in Depth Direction Test 1 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

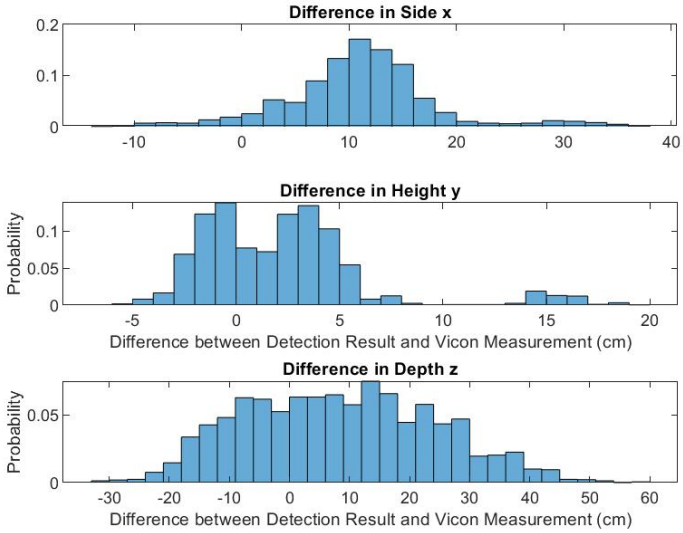

Figure B.120: YOLO v2 Pure Translation in Depth Direction Test 1 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

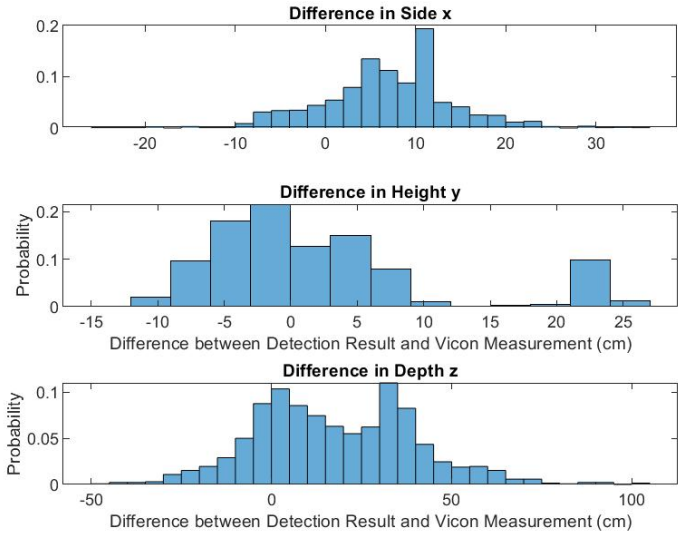

Figure B.121: YOLO v2 Pure Translation in Depth Direction Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

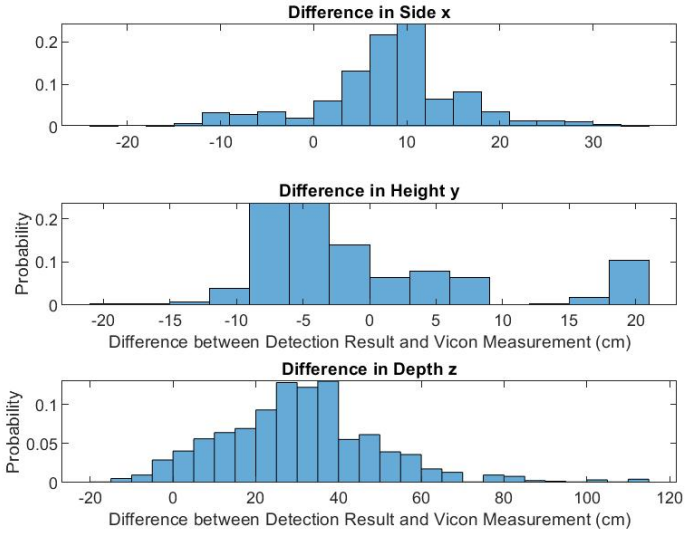

Figure B.122: YOLO v2 Pure Translation in Depth Direction Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

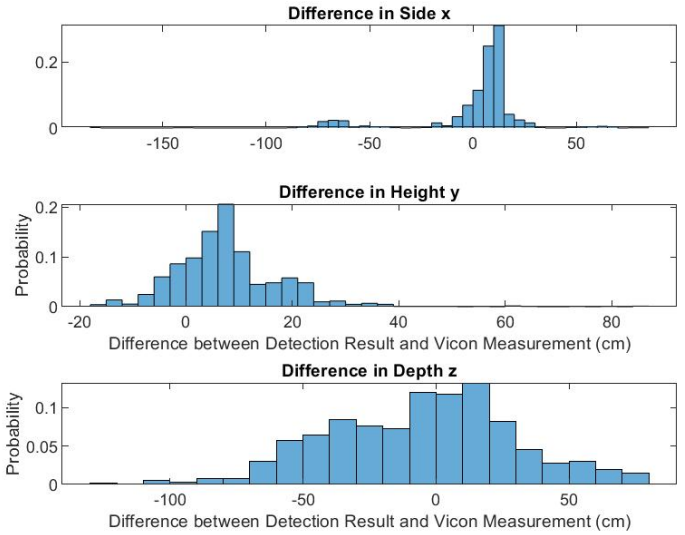

Figure B.123: YOLO v2 Pure Translation in Depth Direction Test 1 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

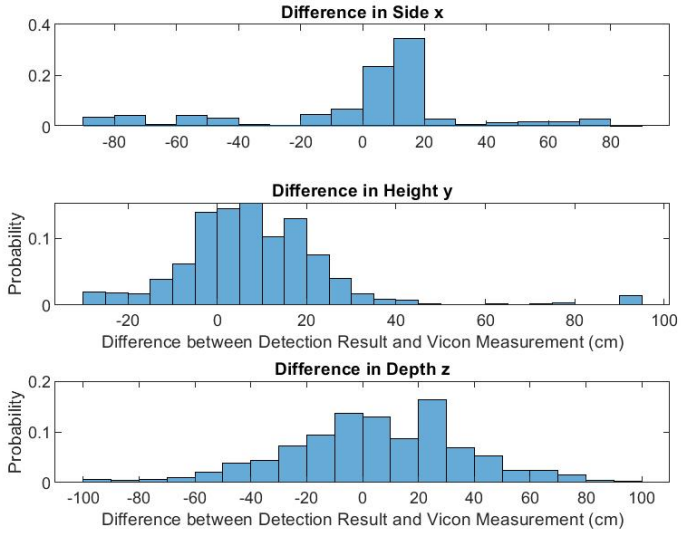

Figure B.124: YOLO v2 Pure Translation in Depth Direction Test 1 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

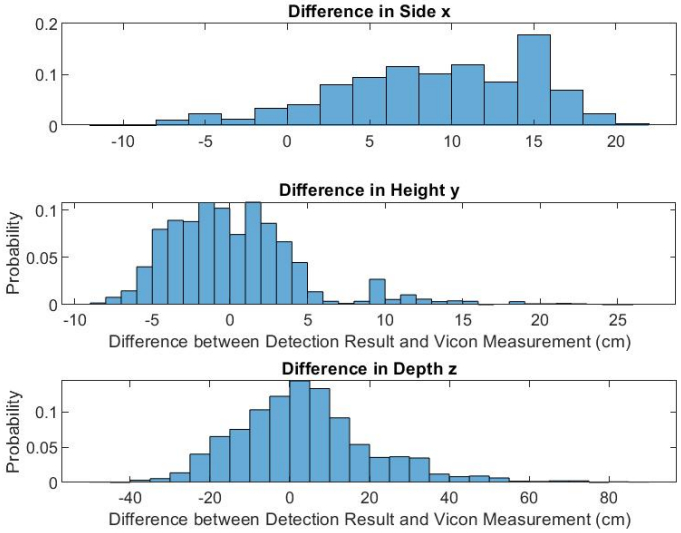

Figure B.125: YOLO v2 Pure Translation in Depth Direction Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

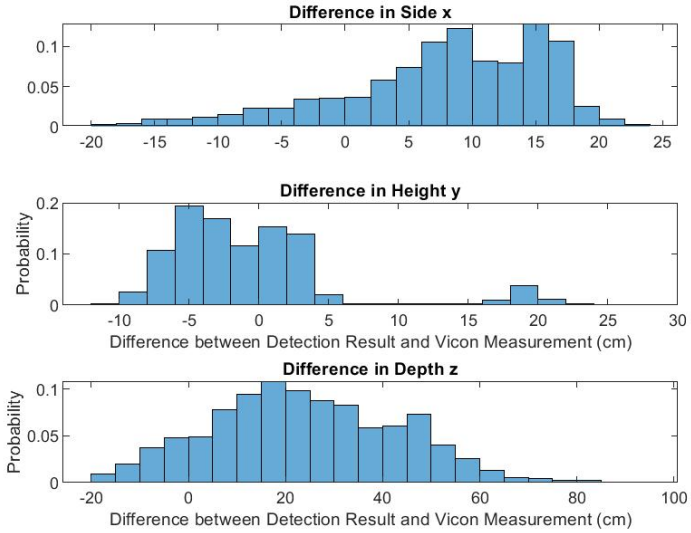

Figure B.126: YOLO v2 Pure Translation in Depth Direction Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

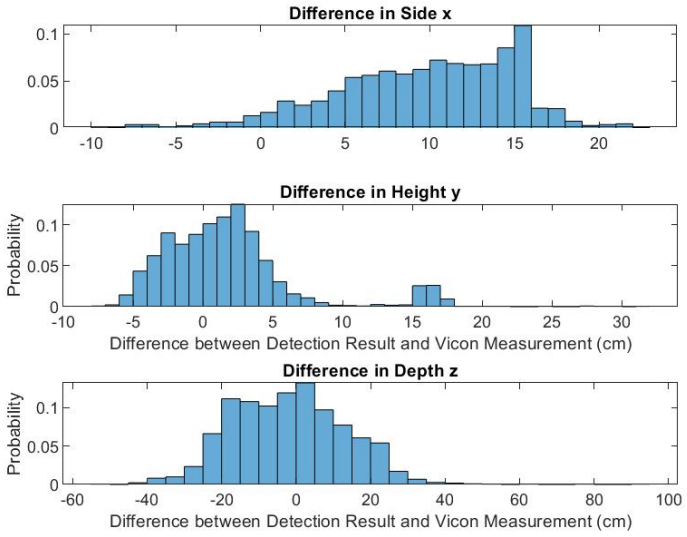

Figure B.127: YOLO v2 Pure Translation in Depth Direction Test 2 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

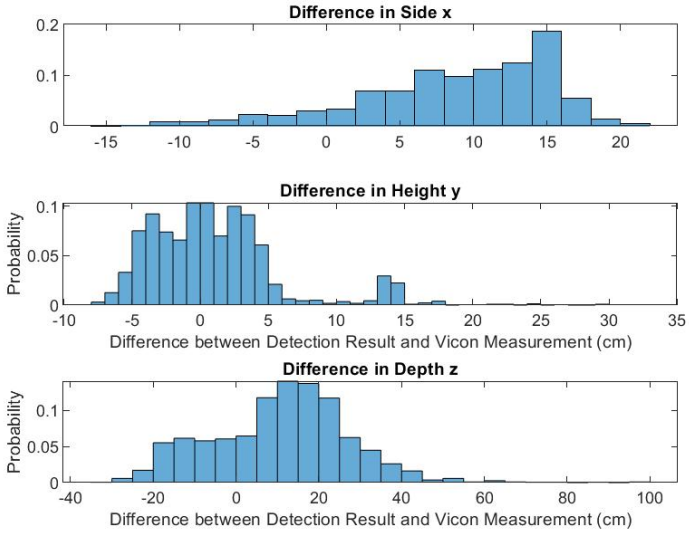

Figure B.128: YOLO v2 Pure Translation in Depth Direction Test 2 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

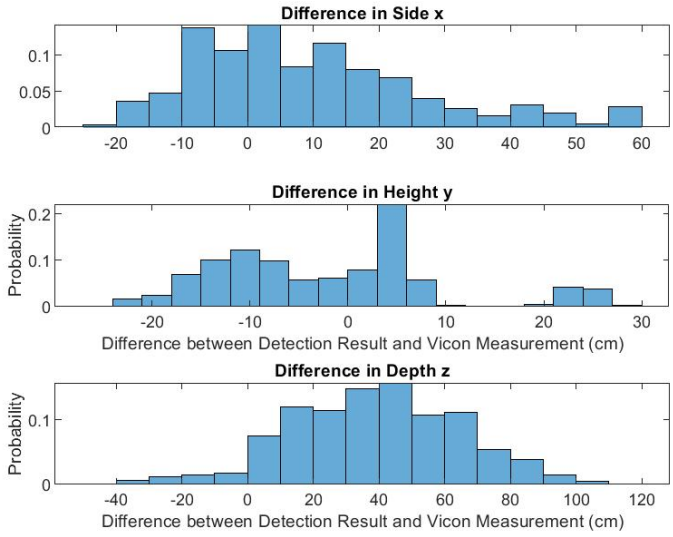

Figure B.129: YOLO v2 Pure Translation in Depth Direction Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

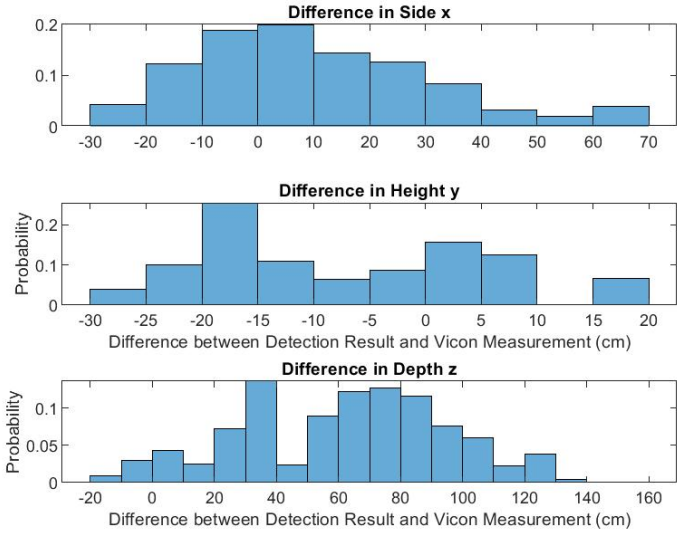

Figure B.130: YOLO v2 Pure Translation in Depth Direction Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

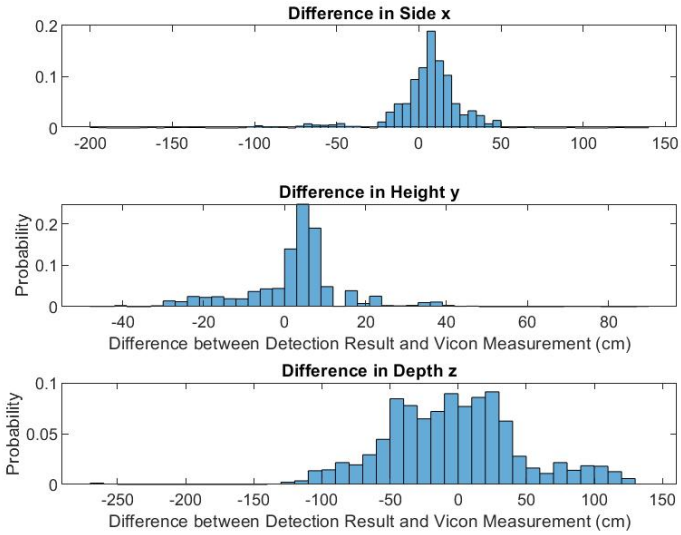

Figure B.131: YOLO v2 Pure Translation in Depth Direction Test 2 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

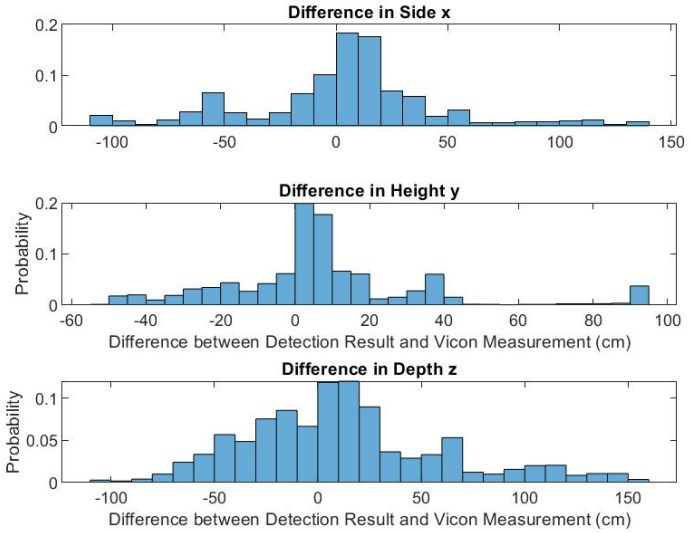

Figure B.132: YOLO v2 Pure Translation in Depth Direction Test 2 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

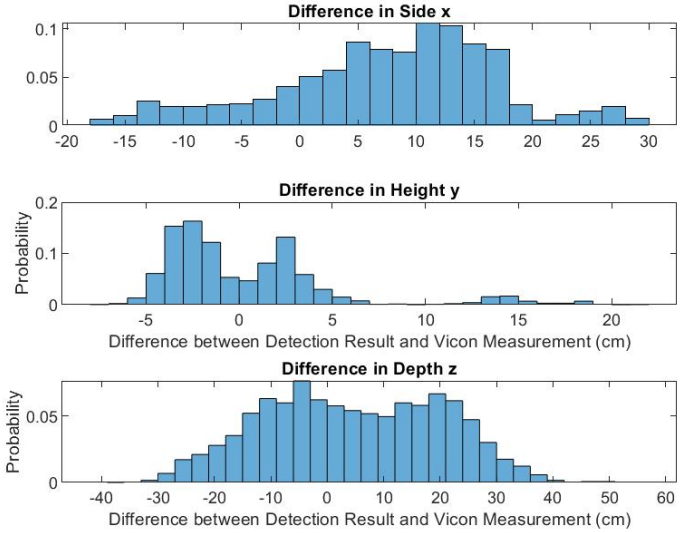

Figure B.133: YOLO v2 Pure Rotation With Test 1 White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

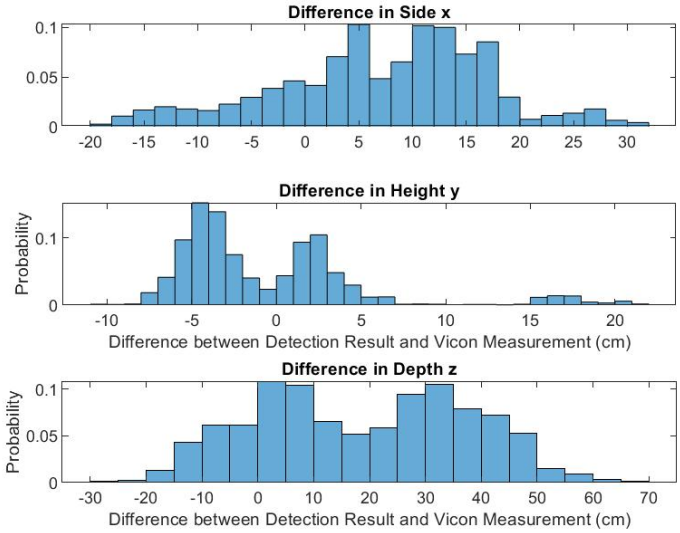

Figure B.134: YOLO v2 Pure Rotation With Test 1 White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

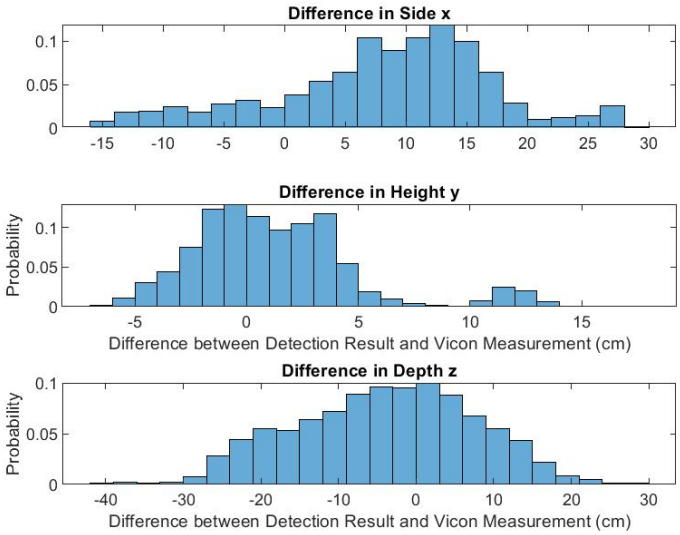

Figure B.135: YOLO v2 Pure Rotation With Test 1 White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

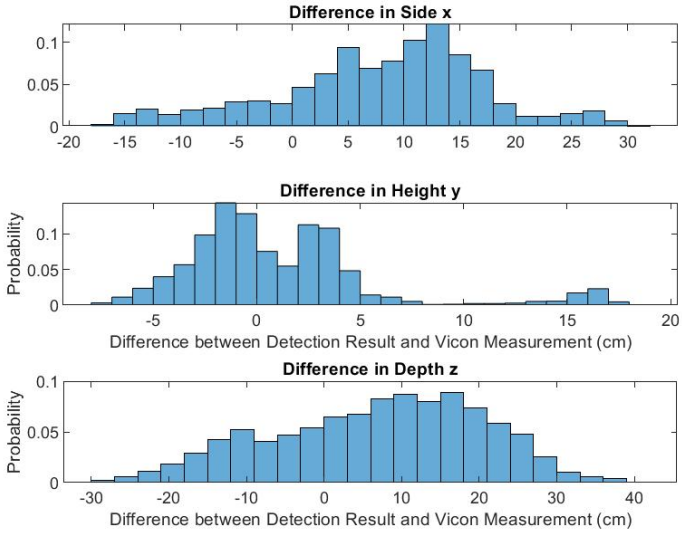

Figure B.136: YOLO v2 Pure Rotation With Test 1 White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

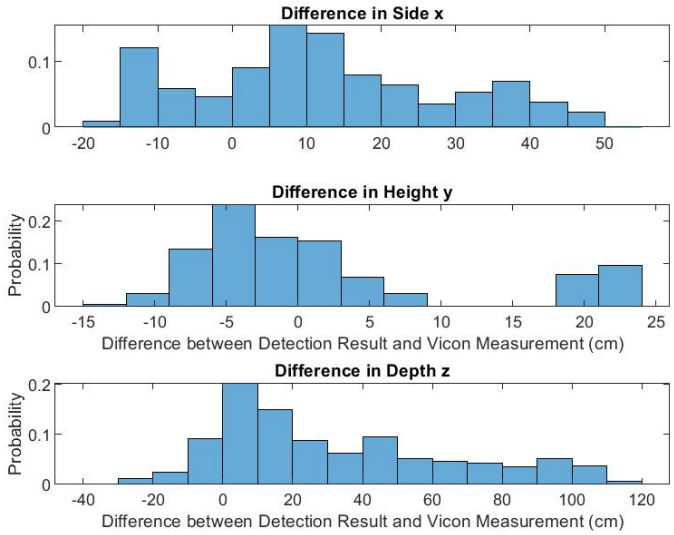

Figure B.137: YOLO v2 Pure Rotation With Test 1 Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

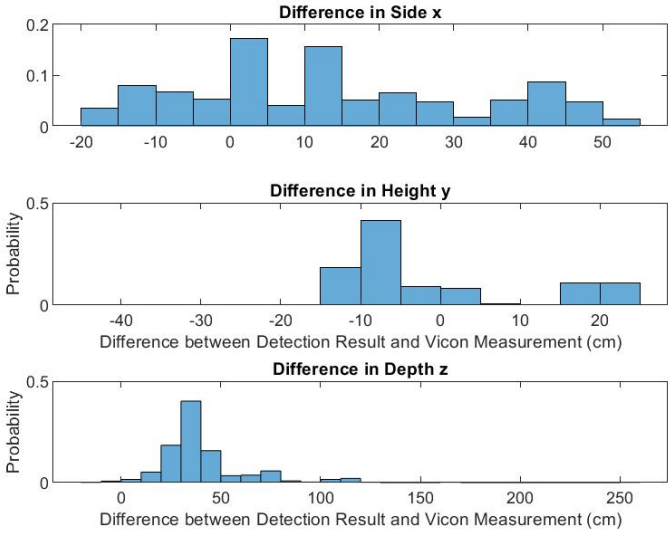

Figure B.138: YOLO v2 Pure Rotation With Test 1 Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

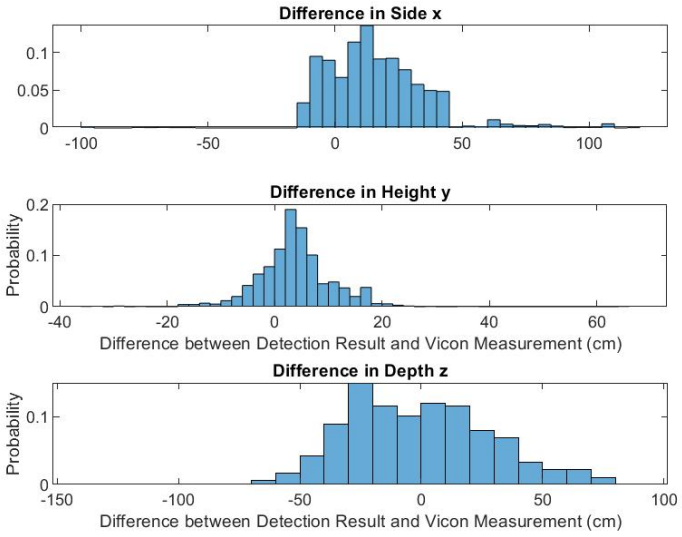

Figure B.139: YOLO v2 Pure Rotation With Test 1 Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

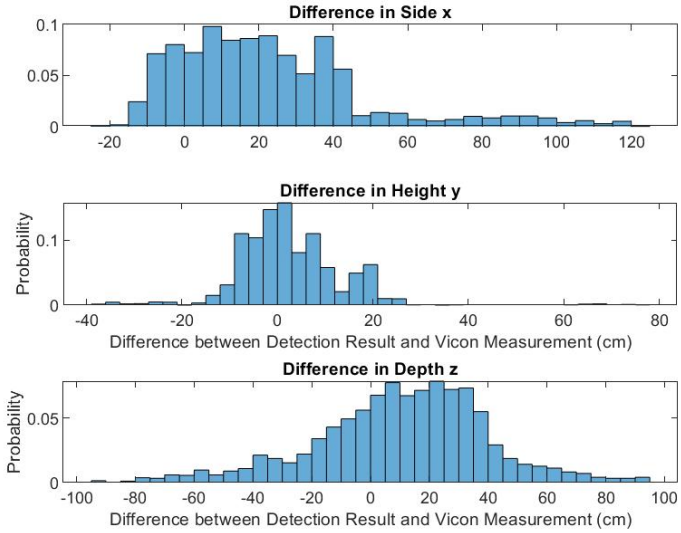

Figure B.140: YOLO v2 Pure Rotation With Test 1 Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

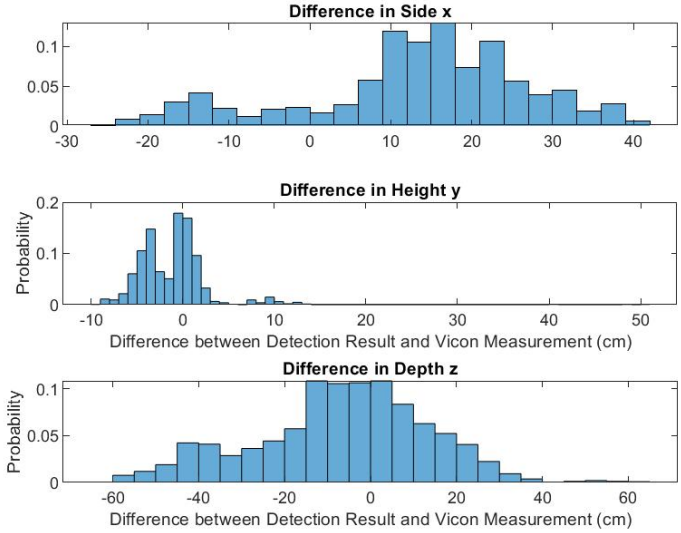

Figure B.141: YOLO v2 Pure Rotation With Test 2 White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

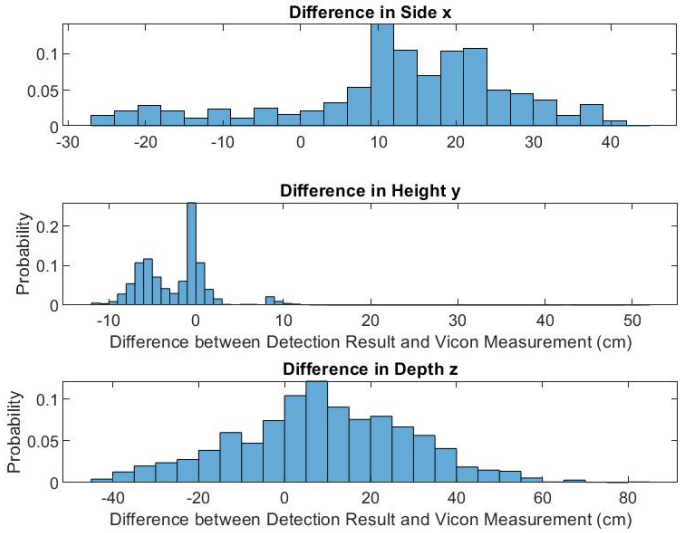

Figure B.142: YOLO v2 Pure Rotation With Test 2 White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

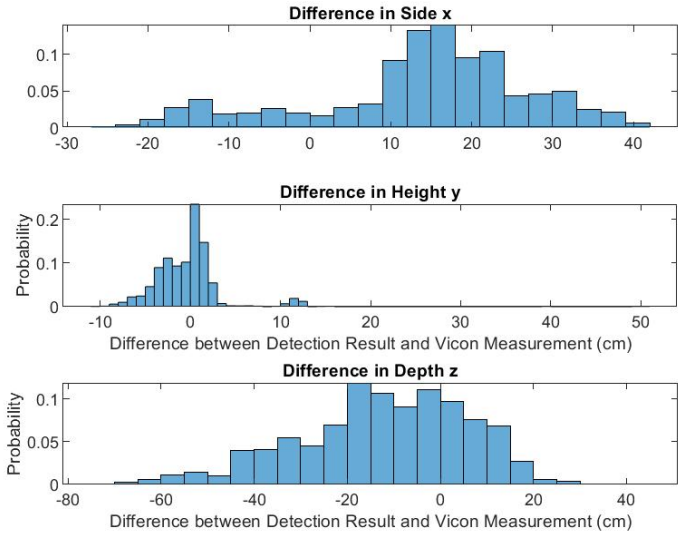

Figure B.143: YOLO v2 Pure Rotation With Test 2 White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

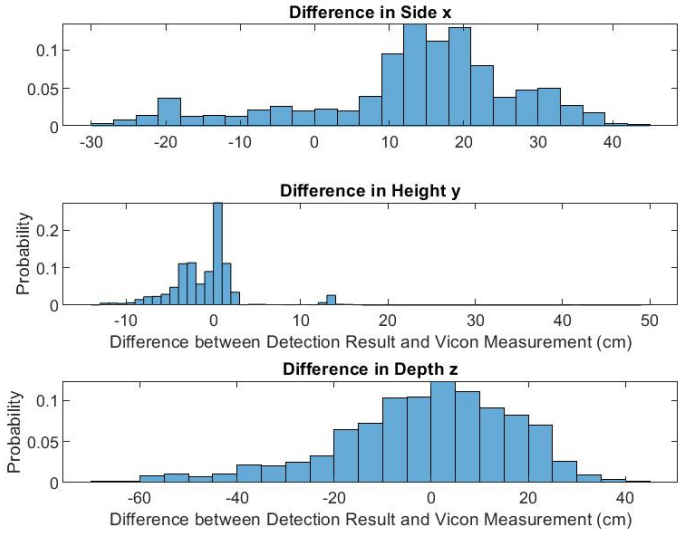

Figure B.144: YOLO v2 Pure Rotation With Test 2 White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

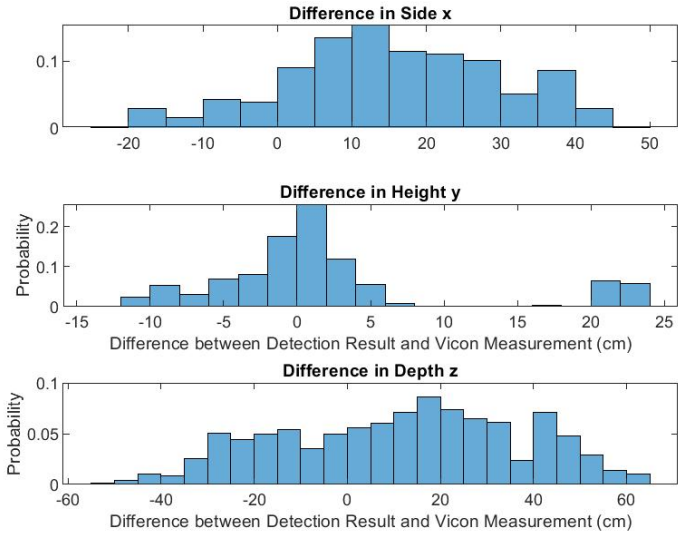

Figure B.145: YOLO v2 Pure Rotation With Test 2 Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

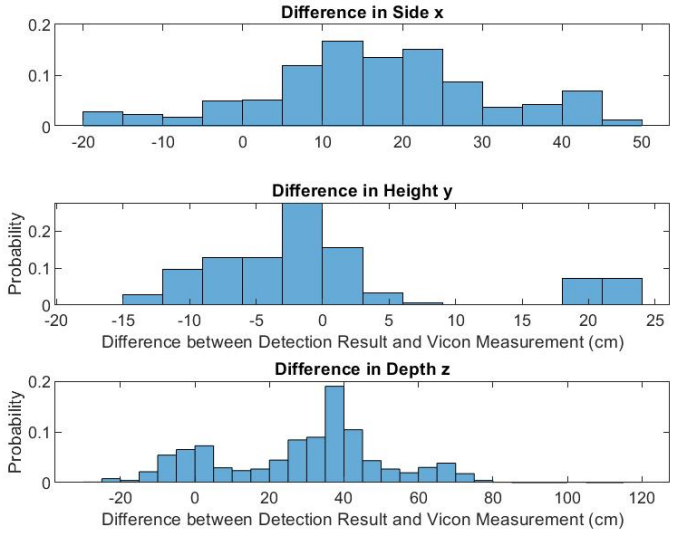

Figure B.146: YOLO v2 Pure Rotation With Test 2 Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

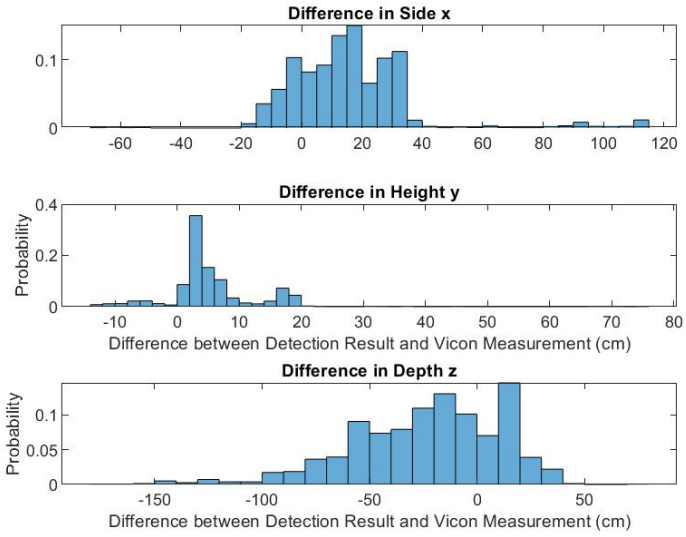

Figure B.147: YOLO v2 Pure Rotation With Test 2 Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

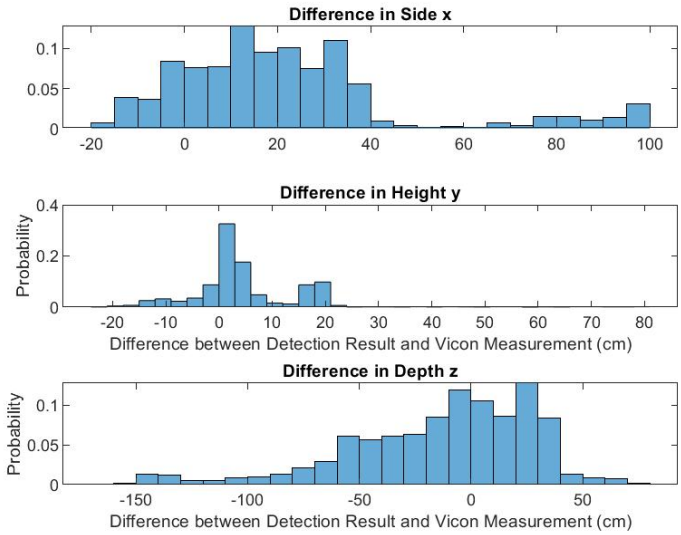

Figure B.148: YOLO v2 Pure Rotation With Test 2 Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

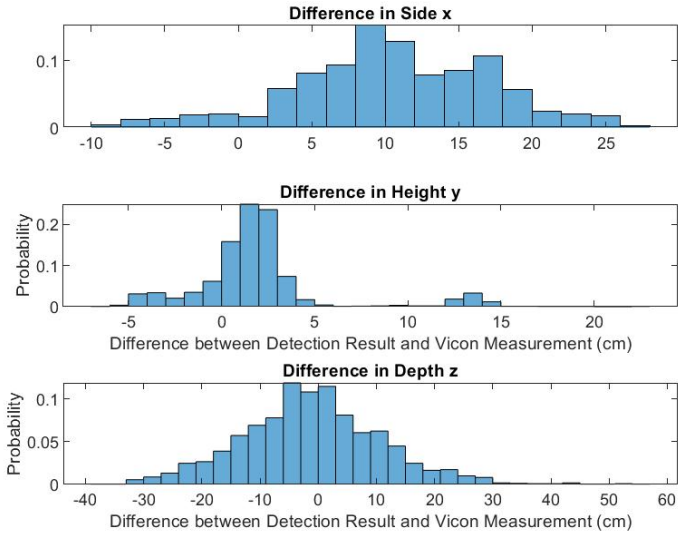

Figure B.149: YOLO v2 Random Flight Pattern Test 1 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

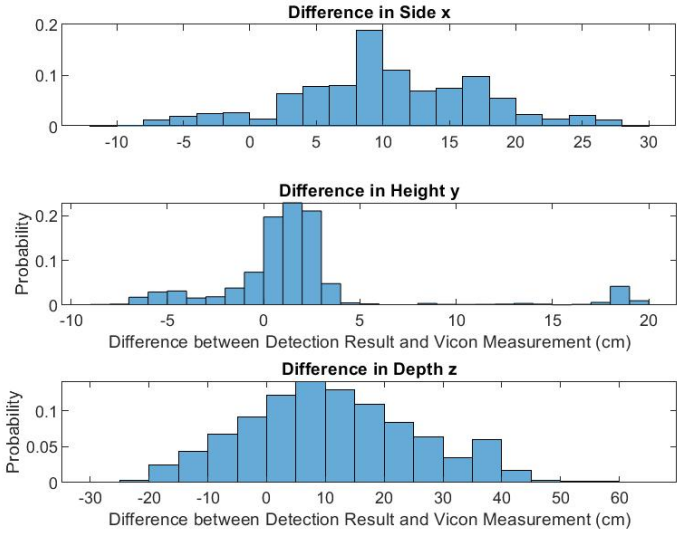

Figure B.150: YOLO v2 Random Flight Pattern Test 1 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

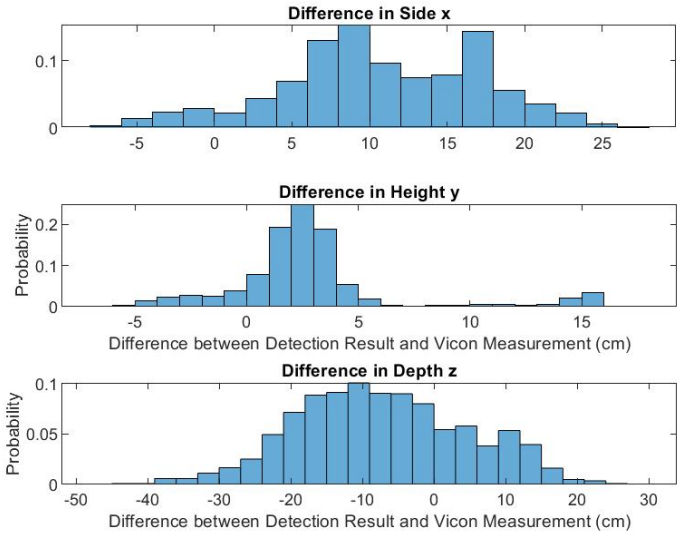

Figure B.151: YOLO v2 Random Flight Pattern Test 1 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

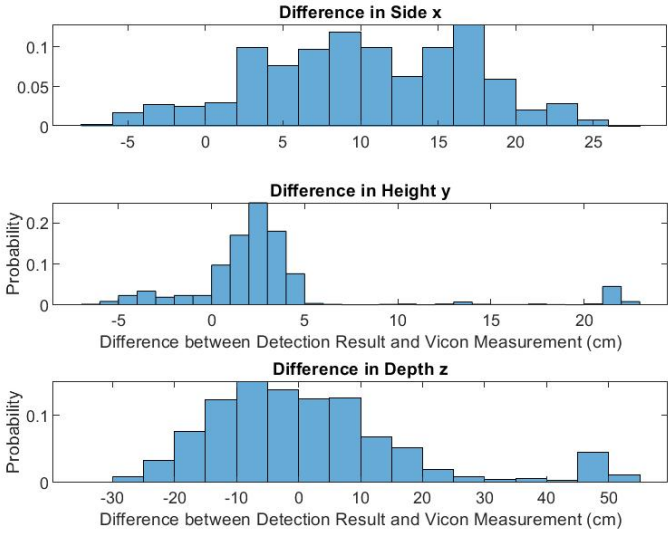

Figure B.152: YOLO v2 Random Flight Pattern Test 1 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

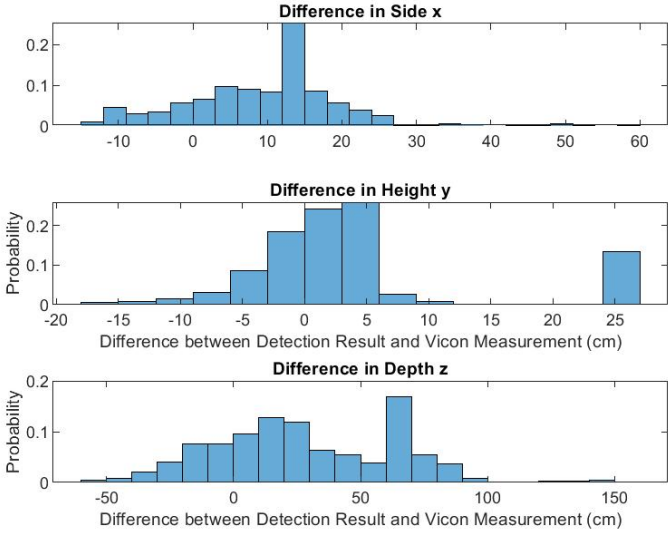

Figure B.153: YOLO v2 Random Flight Pattern Test 1 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

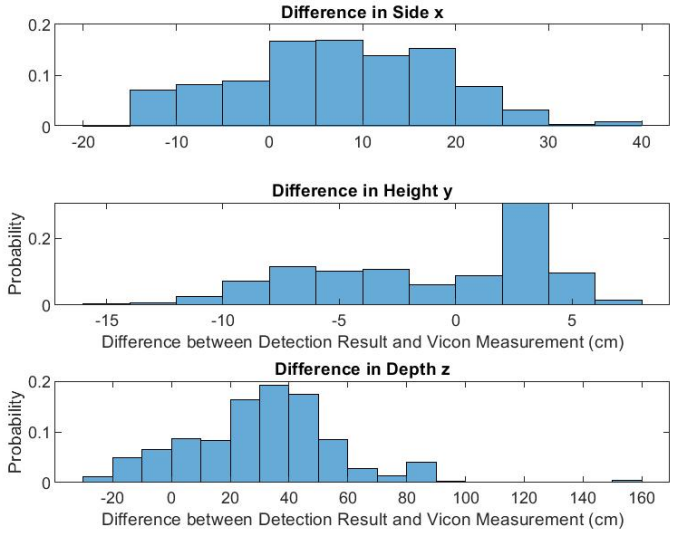

Figure B.154: YOLO v2 Random Flight Pattern Test 1 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

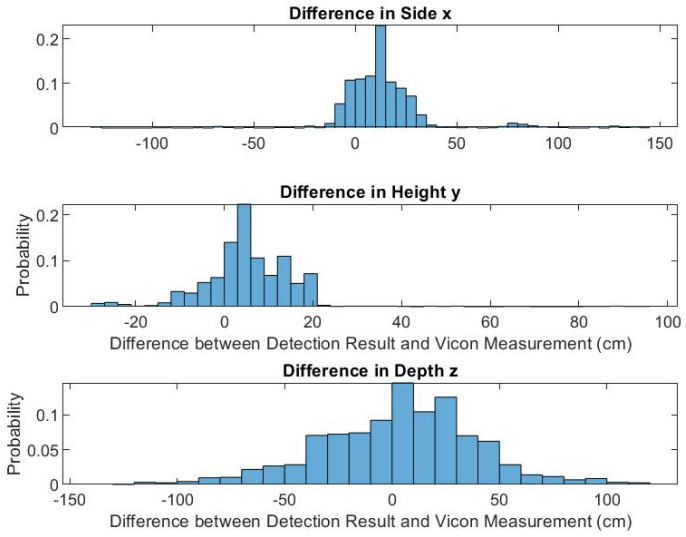

Figure B.155: YOLO v2 Random Flight Pattern Test 1 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

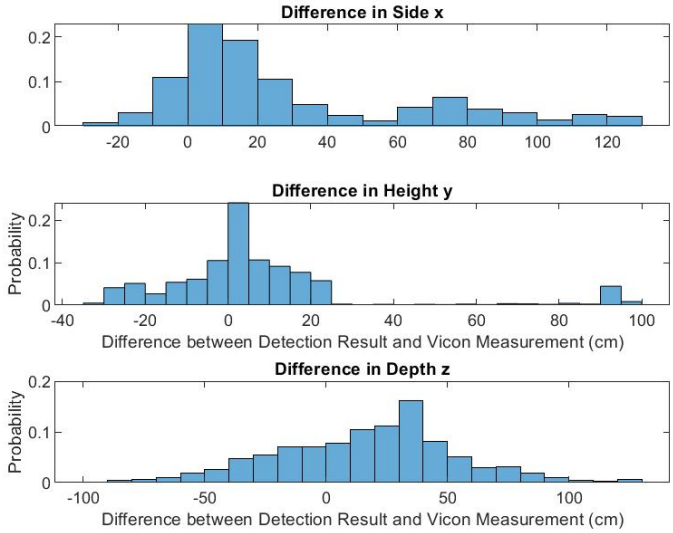

Figure B.156: YOLO v2 Random Flight Pattern Test 1 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

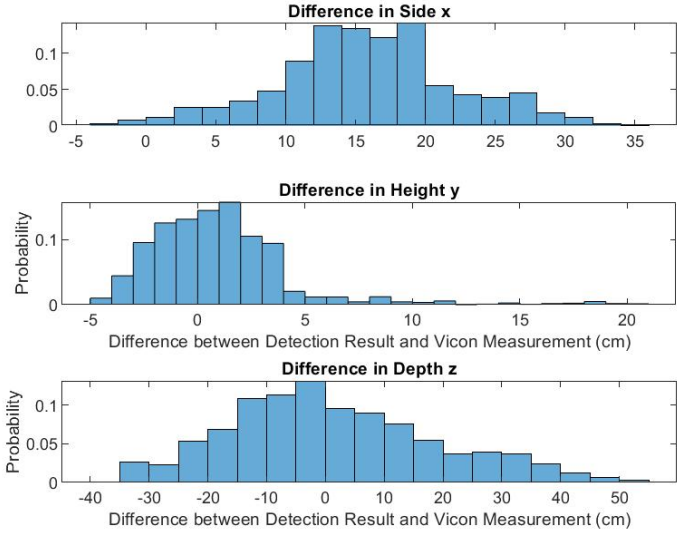

Figure B.157: YOLO v2 Random Flight Pattern Test 2 With White Curtain with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

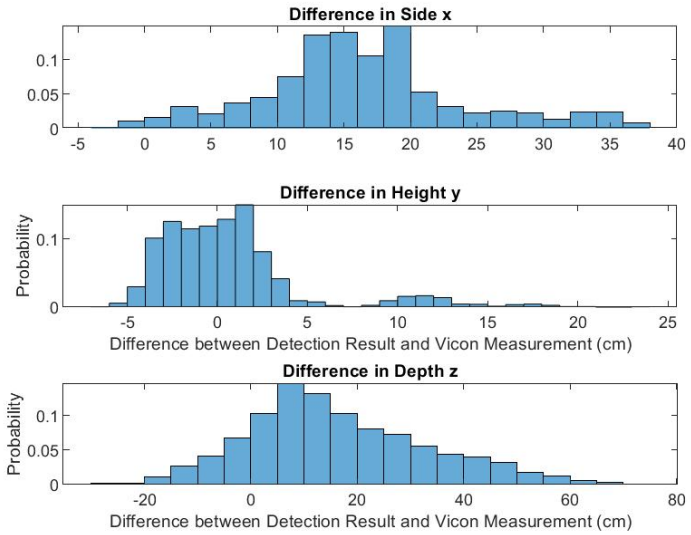

Figure B.158: YOLO v2 Random Flight Pattern Test 2 With White Curtain with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

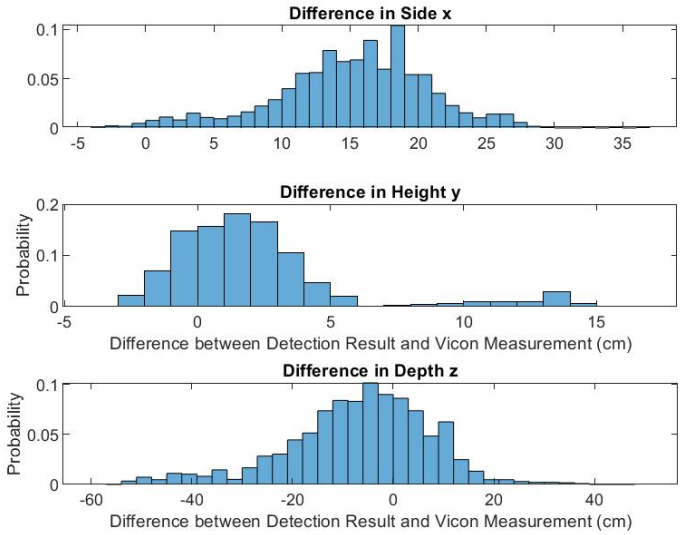

Figure B.159: YOLO v2 Random Flight Pattern Test 2 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

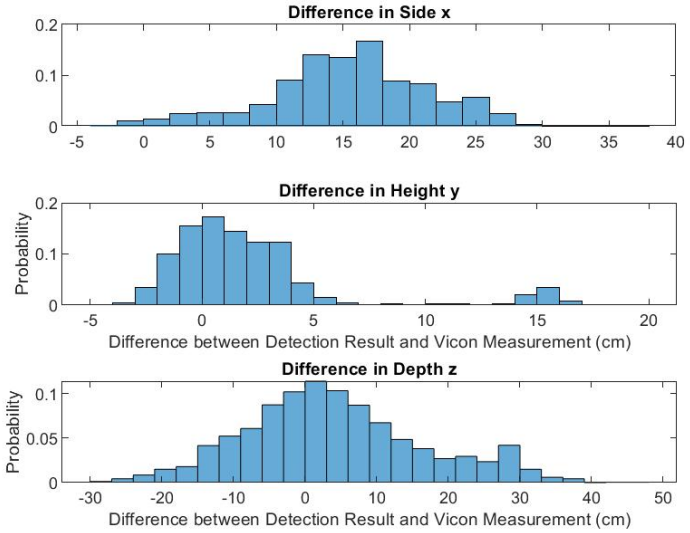

Figure B.160: YOLO v2 Random Flight Pattern Test 2 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

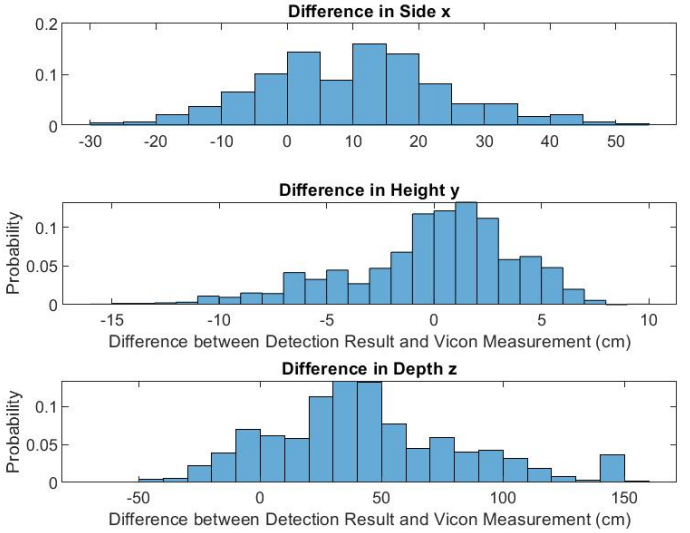

Figure B.161: YOLO v2 Random Flight Pattern Test 2 With Complex Background with Video Unrectified/ Training Unrectified Difference Probability Distribution Histogram

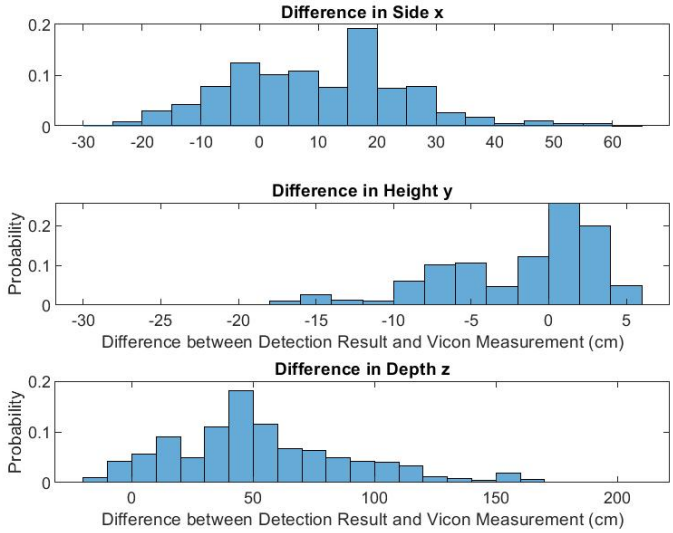

Figure B.162: YOLO v2 Random Flight Pattern Test 2 With Complex Background with Video Rectified/ Training Unrectified Difference Probability Distribution Histogram

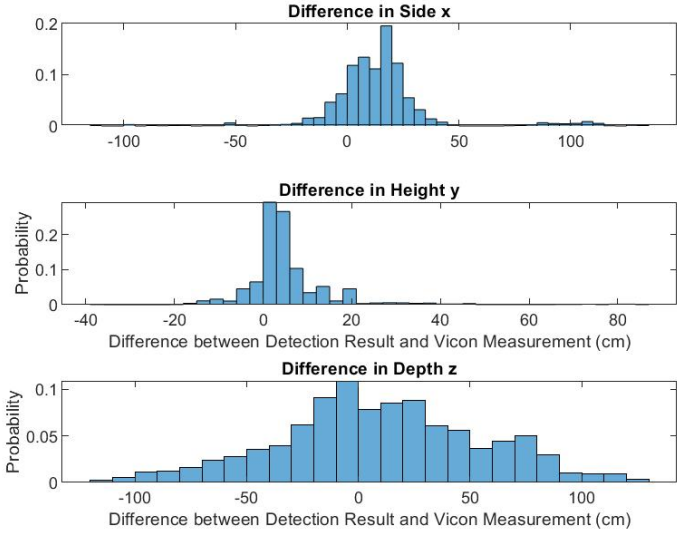

Figure B.163: YOLO v2 Random Flight Pattern Test 2 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

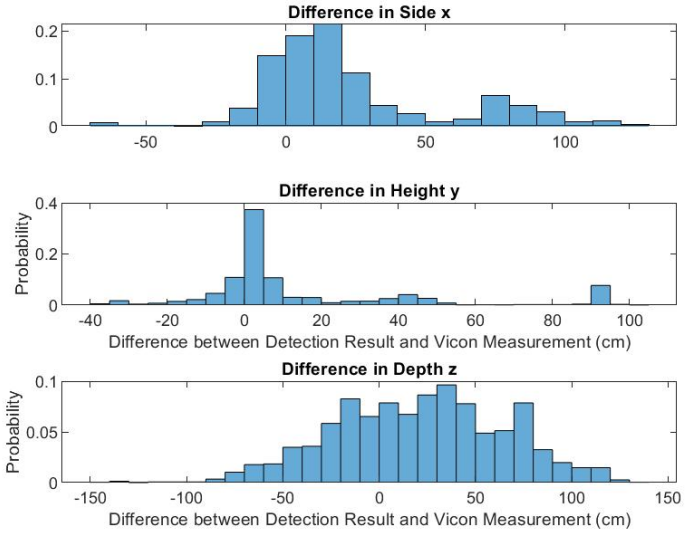

Figure B.164: YOLO v2 Random Flight Pattern Test 2 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

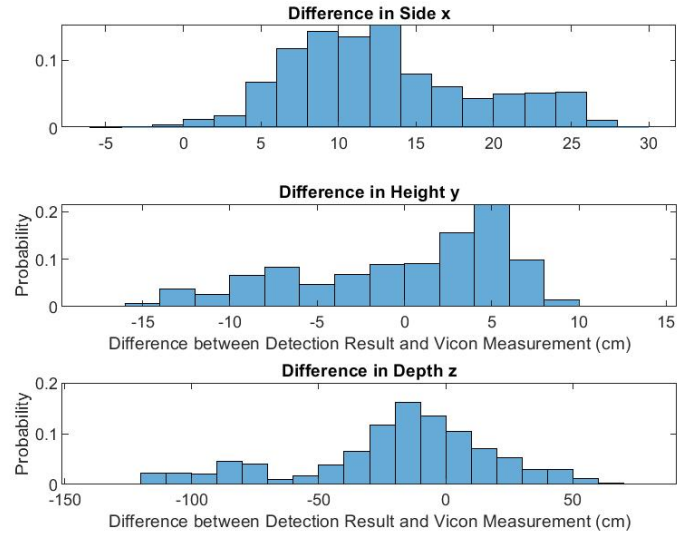

Figure B.165: Tiny YOLO Pure Translation in Side and Height Direction Test 1 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

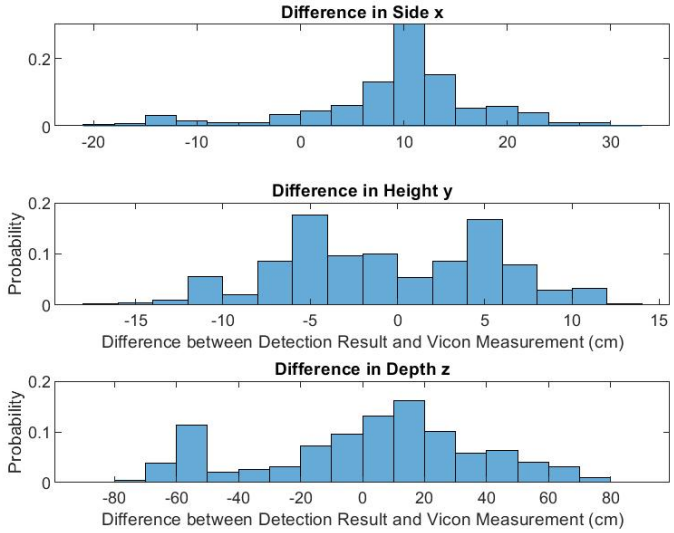

Figure B.166: Tiny YOLO Pure Translation in Side and Height Direction Test 1 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

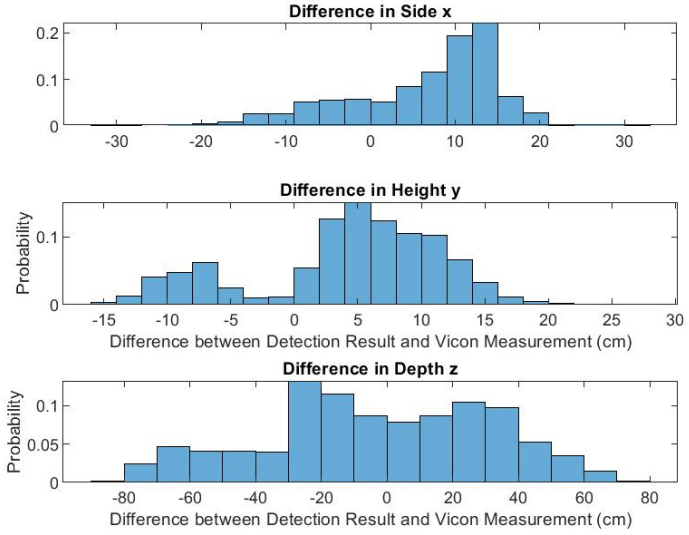

Figure B.167: Tiny YOLO Pure Translation in Side and Height Direction Test 1 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

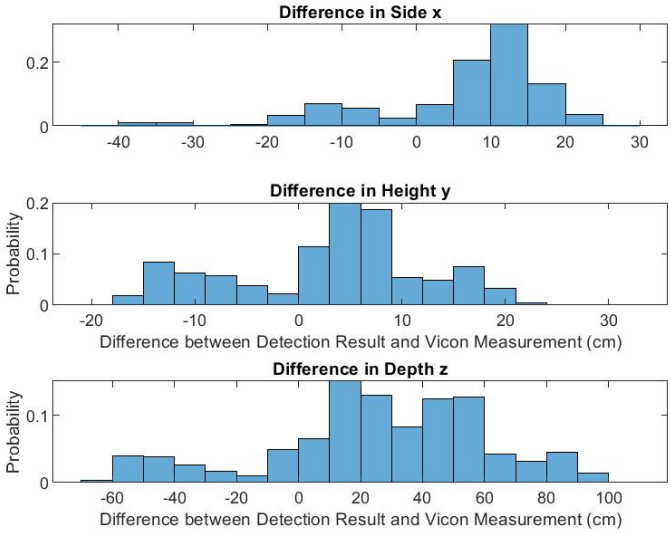

Figure B.168: Tiny YOLO Pure Translation in Side and Height Direction Test 1 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

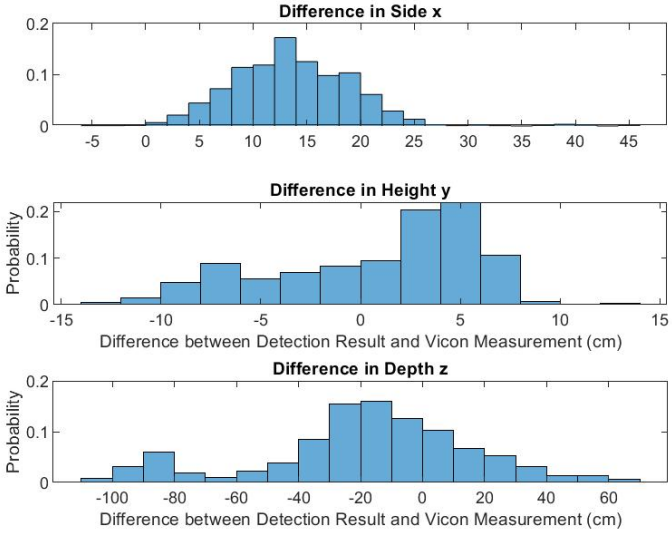

Figure B.169: Tiny YOLO Pure Translation in Side and Height Direction Test 2 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

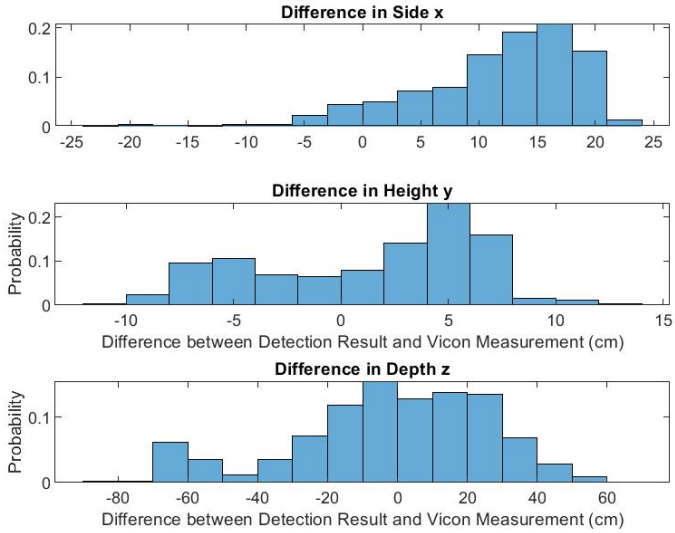

Figure B.170: Tiny YOLO Pure Translation in Side and Height Direction Test 2 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

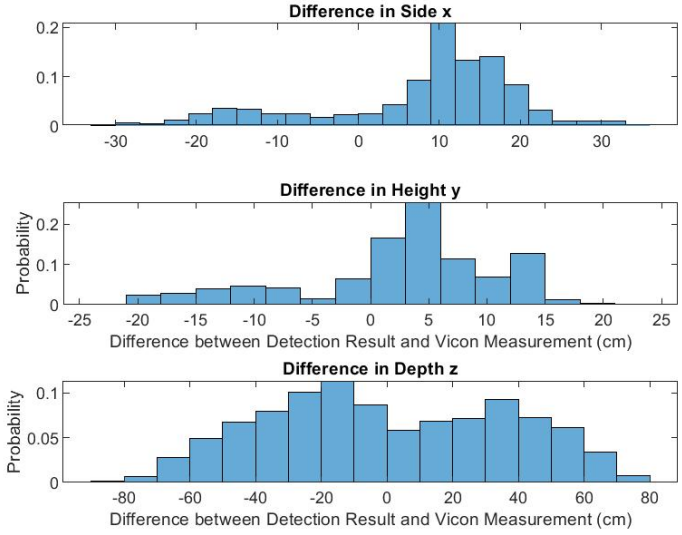

Figure B.171: Tiny YOLO Pure Translation in Side and Height Direction Test 2 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

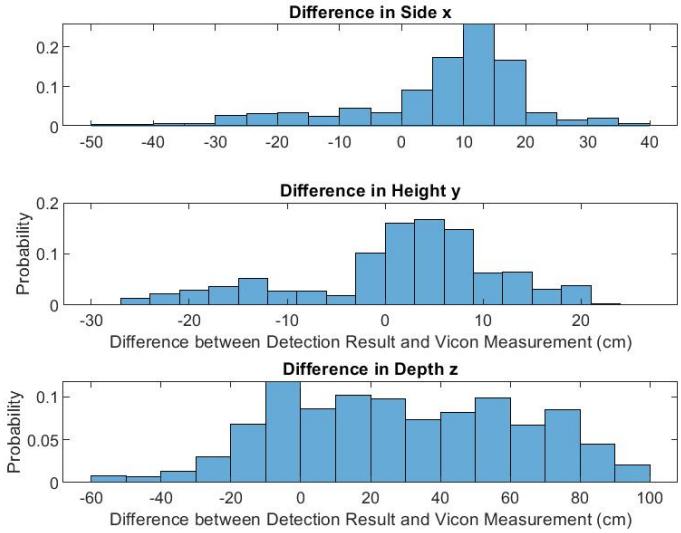

Figure B.172: Tiny YOLO Pure Translation in Side and Height Direction Test 2 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

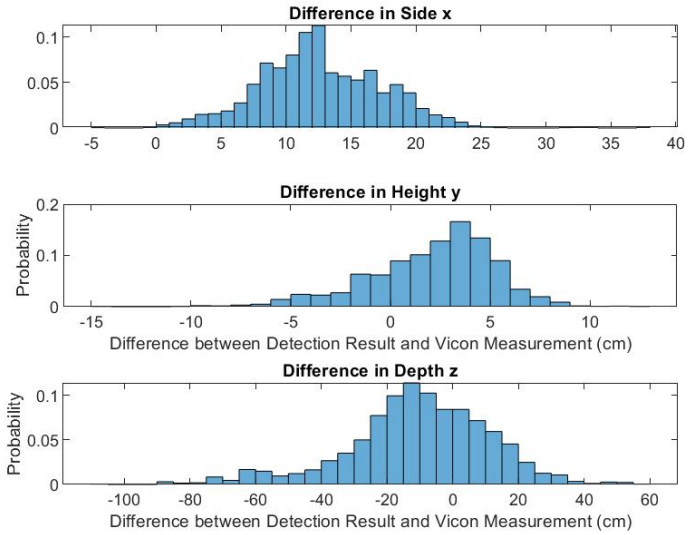

Figure B.173: Tiny YOLO Pure Translation in Depth Direction Test 1 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

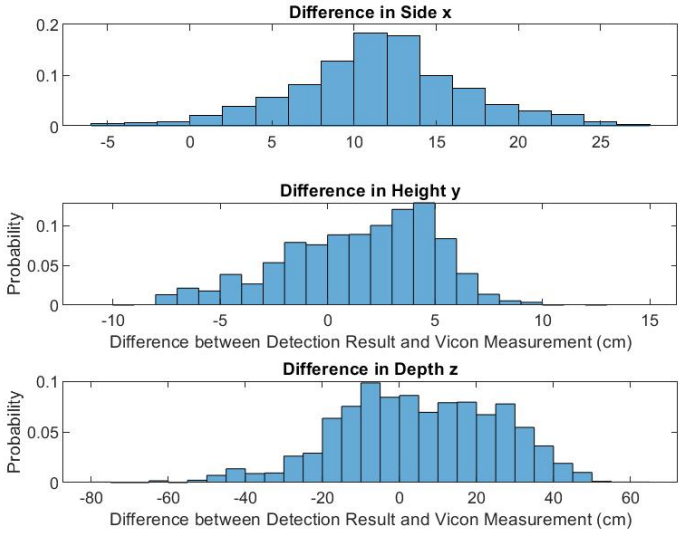

Figure B.174: Tiny YOLO Pure Translation in Depth Direction Test 1 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

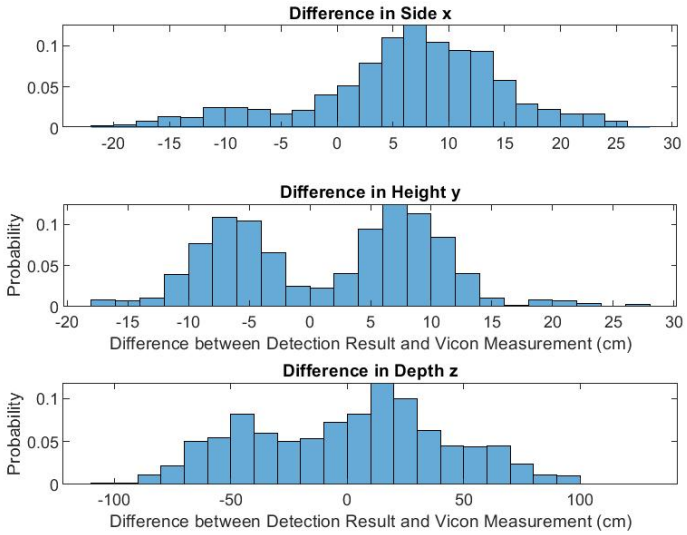

Figure B.175: Tiny YOLO Pure Translation in Depth Direction Test 1 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

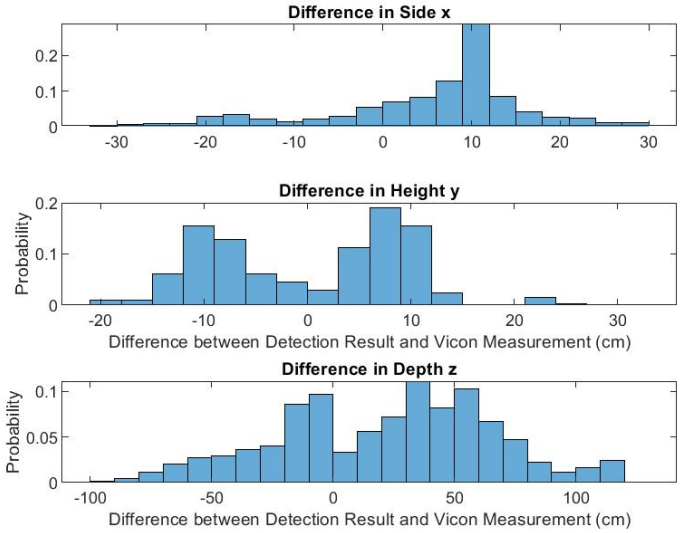

Figure B.176: Tiny YOLO Pure Translation in Depth Direction Test 1 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

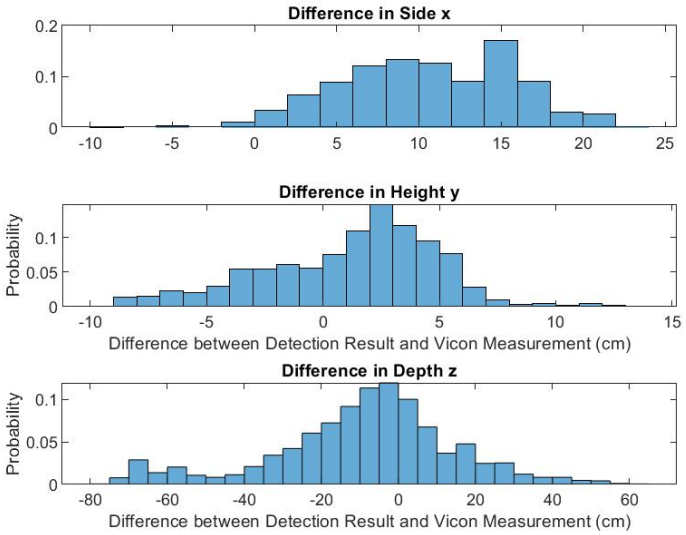

Figure B.177: Tiny YOLO Pure Translation in Depth Direction Test 2 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

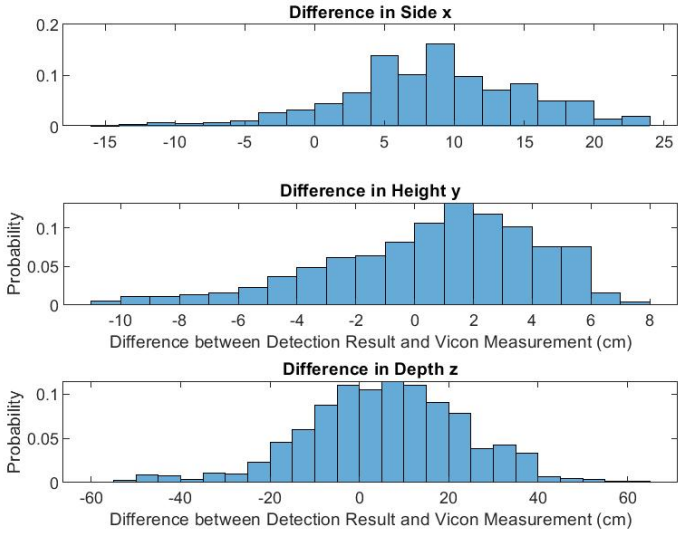

Figure B.178: Tiny YOLO Pure Translation in Depth Direction Test 2 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

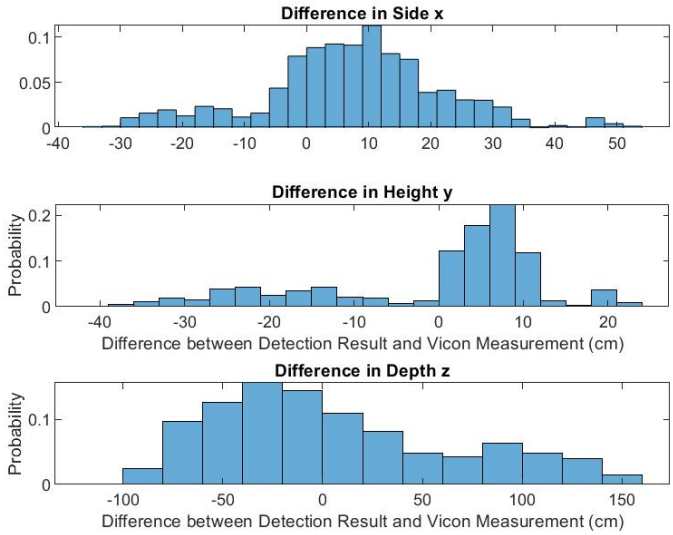

Figure B.179: Tiny YOLO Pure Translation in Depth Direction Test 2 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

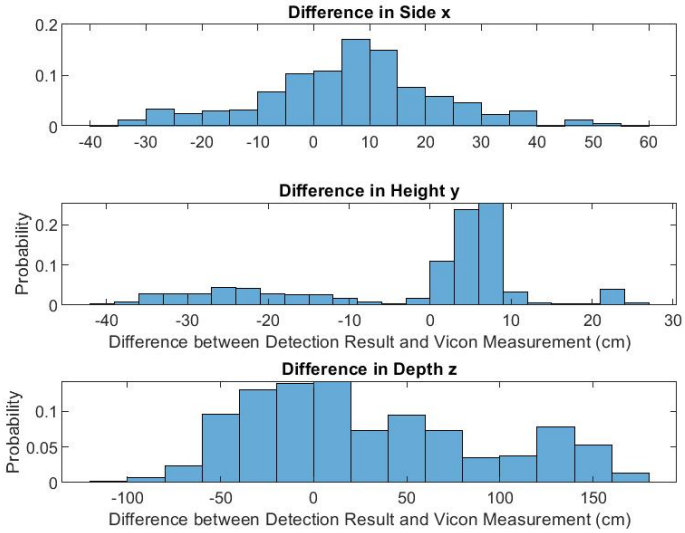

Figure B.180: Tiny YOLO Pure Translation in Depth Direction Test 2 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

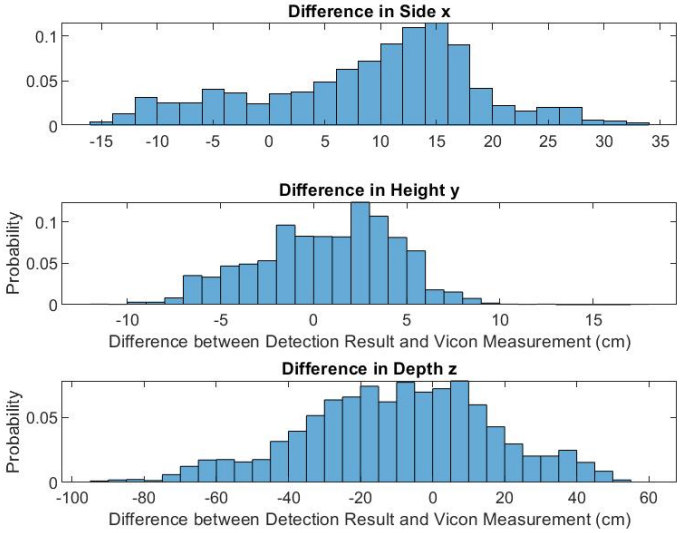

Figure B.181: Tiny YOLO Pure Rotation Test 1 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

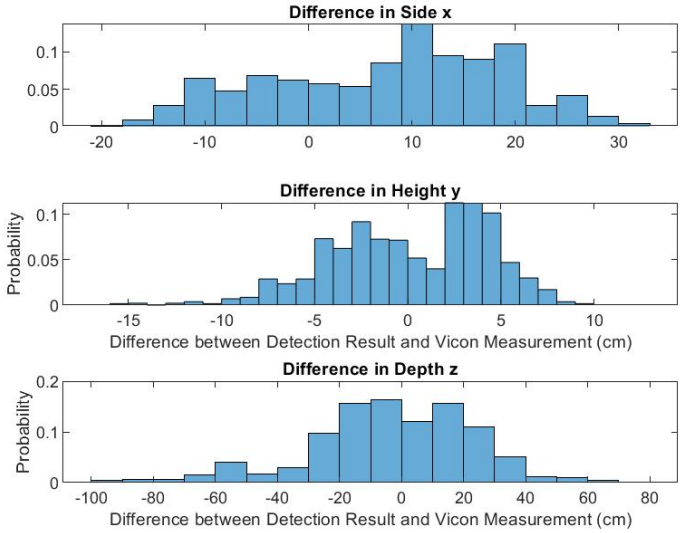

Figure B.182: Tiny YOLO Pure Rotation Test 1 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

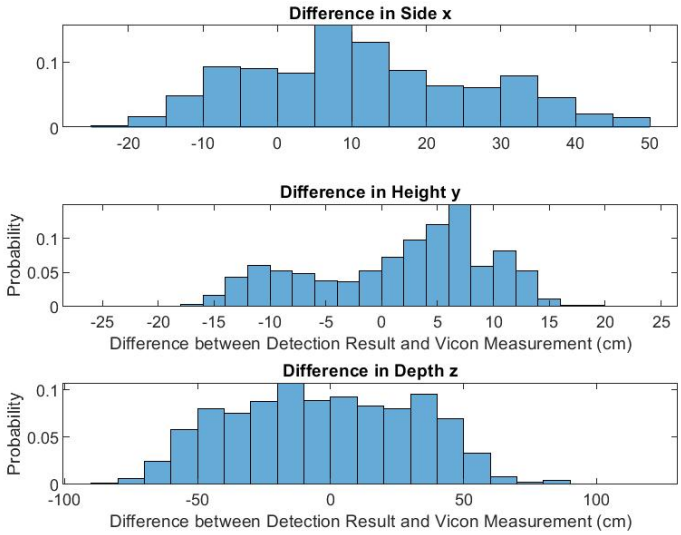

Figure B.183: Tiny YOLO Pure Rotation Test 1 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

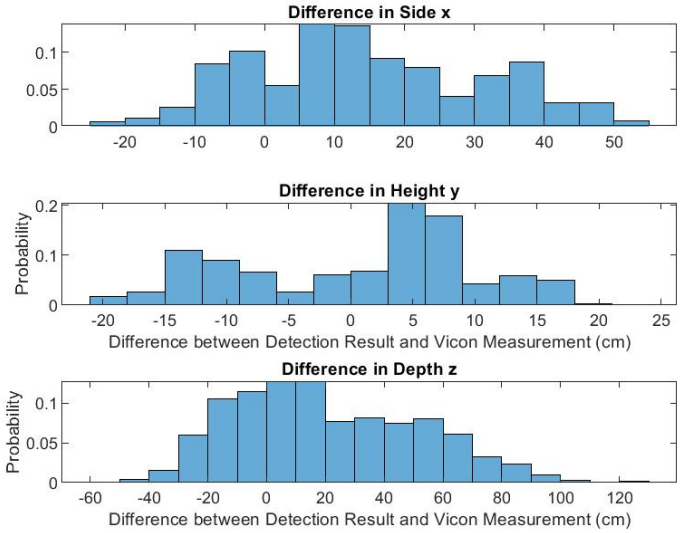

Figure B.184: Tiny YOLO Pure Rotation Test 1 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

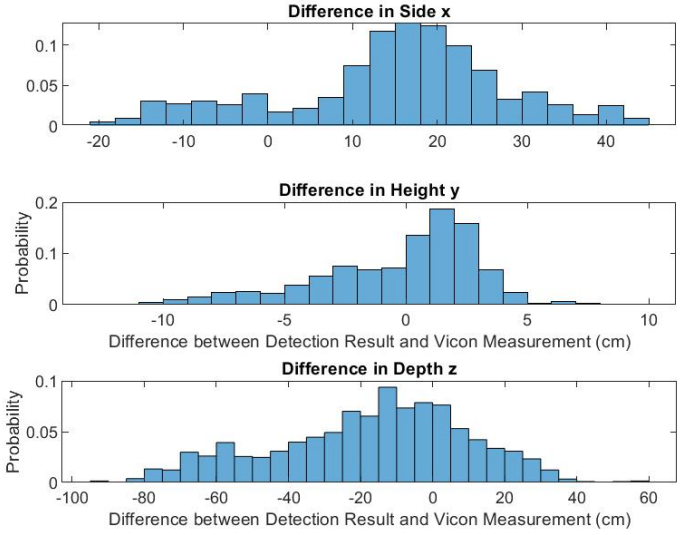

Figure B.185: Tiny YOLO Pure Rotation Test 2 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

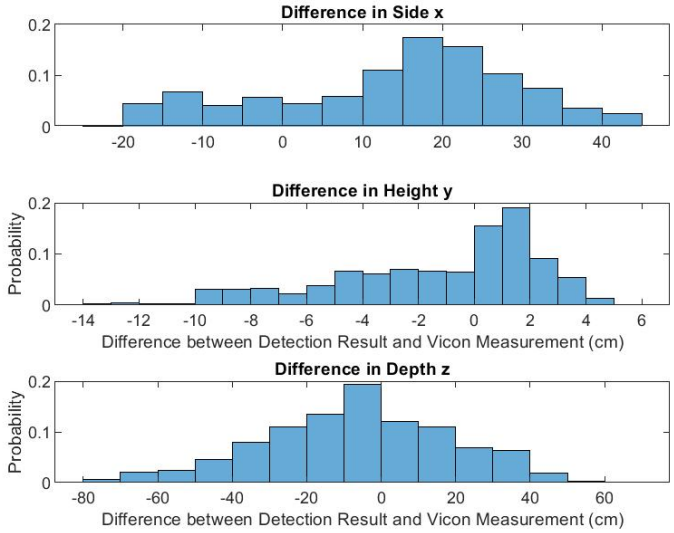

Figure B.186: Tiny YOLO Pure Rotation Test 2 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

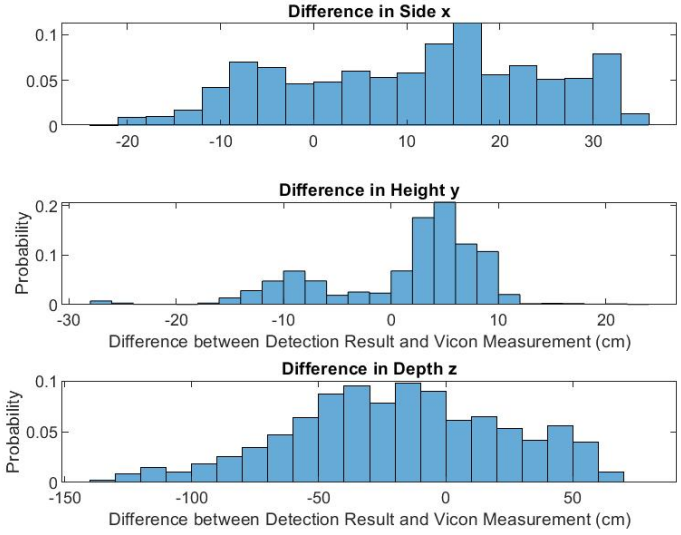

Figure B.187: Tiny YOLO Pure Rotation Test 2 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

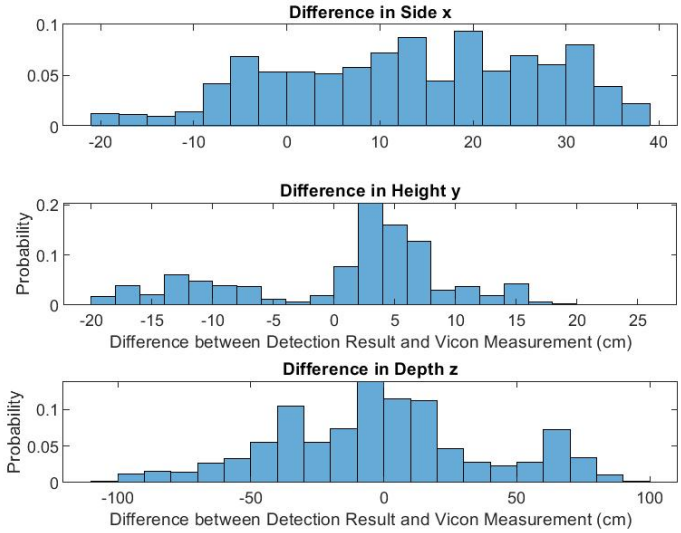

Figure B.188: Tiny YOLO Pure Rotation Test 2 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

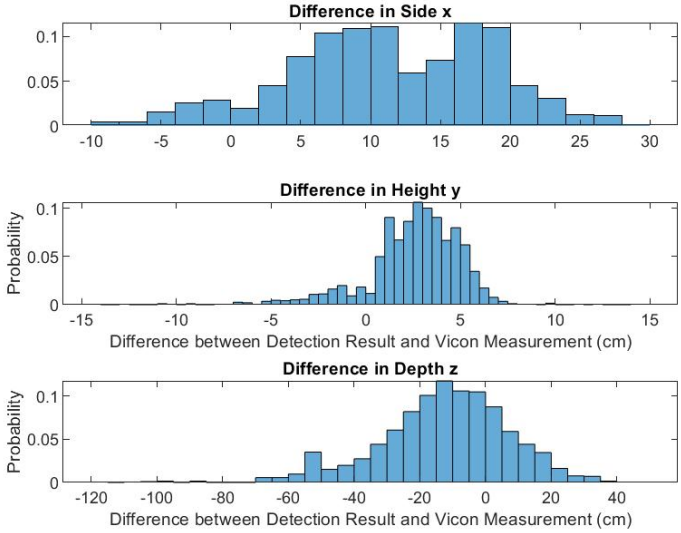

Figure B.189: Tiny YOLO Random Flight Pattern Test 1 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

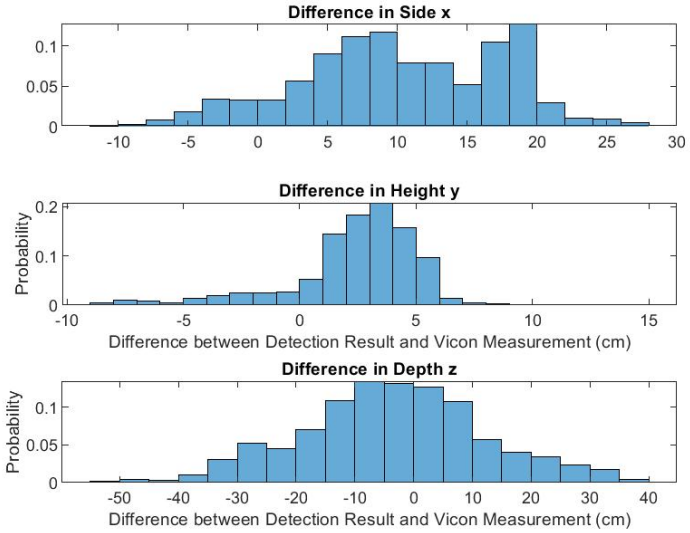

Figure B.190: Tiny YOLO Random Flight Pattern Test 1 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

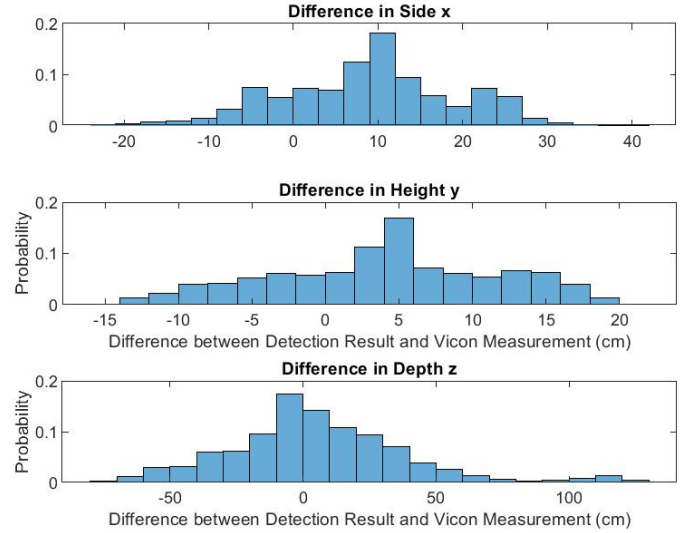

Figure B.191: Tiny YOLO Random Flight Pattern Test 1 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

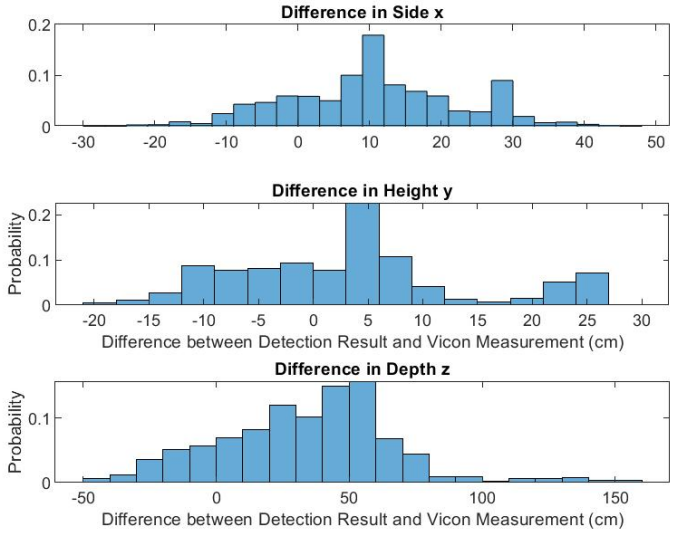

Figure B.192: Tiny YOLO Random Flight Pattern Test 1 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

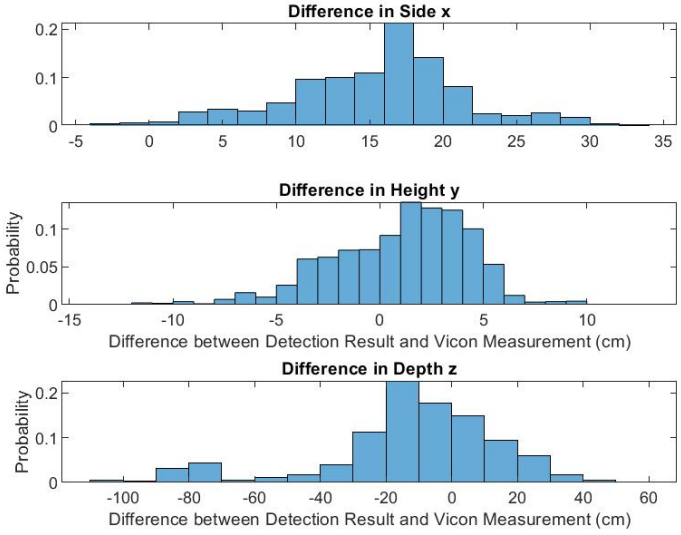

Figure B.193: Tiny YOLO Random Flight Pattern Test 2 With White Curtain with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

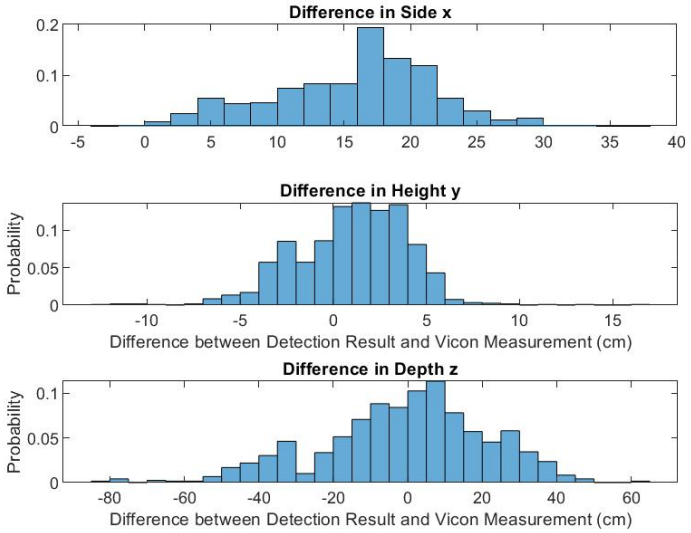

Figure B.194: Tiny YOLO Random Flight Pattern Test 2 With White Curtain with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

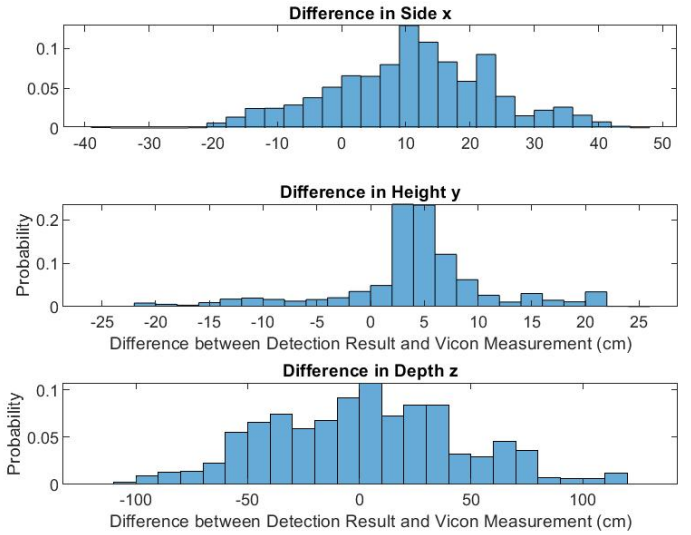

Figure B.195: Tiny YOLO Random Flight Pattern Test 2 With Complex Background with Video Unrectified/ Training Rectified Difference Probability Distribution Histogram

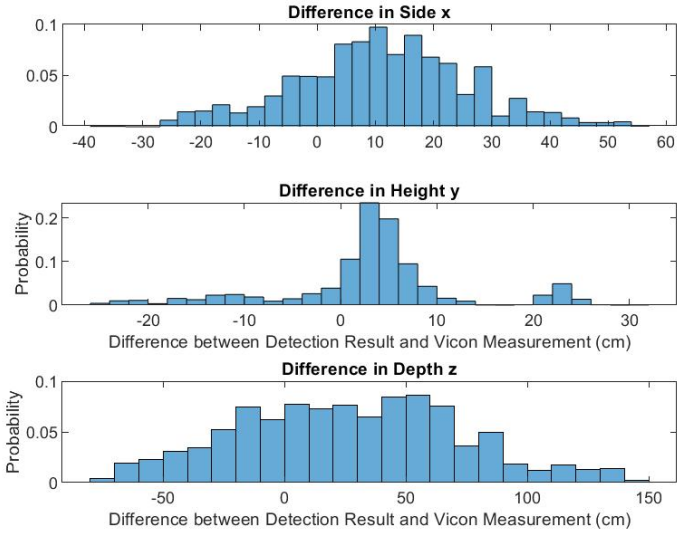

Figure B.196: Tiny YOLO Random Flight Pattern Test 2 With Complex Background with Video Rectified/ Training Rectified Difference Probability Distribution Histogram

B.2.2 Root Mean Square Error

B.2.2.1 SSD MobileNet v1 Root Mean Square Error

Table B.2: Accuracy of SSD MobileNet v1, Pure Translation in Side and Height

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	10.94	10.45	39.52	20.30
WWC/VR/TU/1	13.30	10.89	46.90	23.70
OWC/VU/TU/1	8.24	14.85	18.39	13.83
OWC/VR/TU/1	5.37	21.07	46.63	24.35
WWC/VU/TU/2	13.21	9.22	34.90	19.11
WWC/VR/TU/2	14.89	11.19	53.10	26.39
OWC/VU/TU/2	15.13	17.02	19.38	17.18
OWC/VR/TU/2	14.14	18.87	34.67	22.56

Table B.3: Accuracy of SSD MobileNet v1, Pure Translation in Depth

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	14.07	6.51	30.80	17.13
WWC/VR/TU/1	15.60	6.01	40.25	20.62
OWC/VU/TU/1	9.97	14.99	19.34	14.76
OWC/VR/TU/1	11.05	16.79	35.95	21.26
WWC/VU/TU/2	11.65	7.20	30.08	16.31
WWC/VR/TU/2	12.88	8.91	45.23	22.34
OWC/VU/TU/2	5.32	17.37	26.07	16.25
OWC/VR/TU/2	7.06	16.00	25.08	16.05

Table B.4: Accuracy of SSD MobileNet v1, Pure Rotation

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	12.65	6.87	31.34	16.95
WWC/VR/TU/1	12.92	7.32	35.12	18.45
OWC/VU/TU/1	12.56	13.51	11.07	12.38
OWC/VR/TU/1	14.87	16.88	33.65	21.80
WWC/VU/TU/2	20.11	7.17	31.52	19.60
WWC/VR/TU/2	20.32	7.76	35.17	21.09
OWC/VU/TU/2	16.67	16.41	19.38	17.49
OWC/VR/TU/2	17.73	19.15	36.43	24.44

Table B.5: Accuracy of SSD MobileNet v1, Complex Flight

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	12.60	4.71	26.73	14.68
WWC/VR/TU/1	12.86	7.49	34.34	18.23
OWC/VU/TU/1	12.48	15.89	17.34	15.24
OWC/VR/TU/1	12.90	23.52	61.87	32.76
WWC/VU/TU/2	16.78	6.63	31.10	18.17
WWC/VR/TU/2	17.11	7.56	35.52	20.06
OWC/VU/TU/2	14.76	17.13	20.49	17.46
OWC/VR/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected

B.2.2.2 SSD Inception v2 Root Mean Square Error

Table B.6: Accuracy of SSD Inception v2, Pure Translation in Side and Height

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	10.30	7.33	21.46	13.03
WWC/VR/TU/1	12.03	9.04	37.59	19.55
OWC/VU/TU/1	9.53	13.69	10.63	11.28
OWC/VR/TU/1	8.31	16.59	23.03	15.98
WWC/VU/TU/2	12.55	8.66	25.68	15.63
WWC/VR/TU/2	13.59	10.00	46.45	23.34
OWC/VU/TU/2	11.71	15.71	6.53	11.32
OWC/VR/TU/2	11.17	17.41	26.57	18.38

Table B.7: Accuracy of SSD Inception v2, Pure Translation in Depth

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	13.72	4.14	22.46	13.44
WWC/VR/TU/1	14.93	5.17	37.83	19.31
OWC/VU/TU/1	9.19	11.28	12.59	11.02
OWC/VR/TU/1	10.32	13.07	11.67	11.69
WWC/VU/TU/2	11.33	5.43	19.58	12.11
WWC/VR/TU/2	12.62	7.04	35.84	18.50
OWC/VU/TU/2	5.81	14.91	21.72	14.15
OWC/VR/TU/2	7.51	14.82	24.02	15.45

Table B.8: Accuracy of SSD Inception v2, Pure Rotation

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	12.82	5.31	18.94	12.36
WWC/VR/TU/1	12.68	5.71	23.12	13.84
OWC/VU/TU/1	12.12	10.89	23.00	15.34
OWC/VR/TU/1	10.18	16.43	34.57	20.39
WWC/VU/TU/2	20.16	5.12	21.63	15.64
WWC/VR/TU/2	21.05	6.79	24.78	17.54
OWC/VU/TU/2	15.28	13.03	15.90	14.74
OWC/VR/TU/2	17.43	14.97	23.35	18.59

Table B.9: Accuracy of SSD Inception v2, Complex Flight

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	11.91	4.82	20.91	12.55
WWC/VR/TU/1	11.07	7.81	33.05	17.31
OWC/VU/TU/1	12.18	16.82	21.89	16.96
OWC/VR/TU/1	7.37	13.39	61.47	27.41
WWC/VU/TU/2	15.66	5.43	22.45	14.51
WWC/VR/TU/2	16.21	7.57	31.31	18.37
OWC/VU/TU/2	14.18	21.52	39.09	24.93
OWC/VR/TU/2	18.94	39.58	51.20	36.57

B.2.2.3 Faster RCNN Inception v2 Root Mean Square Error

Table B.10: Accuracy of Faster RCNN Inception v2, Pure Translation in Side and Height

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	10.88	7.67	19.35	12.63
WWC/VR/TU/1	11.63	8.06	27.70	15.79
OWC/VU/TU/1	14.75	8.70	34.37	19.27
OWC/VR/TU/1	73.60	9.77	58.49	47.28
WWC/VU/TU/2	11.78	7.85	20.57	13.40
WWC/VR/TU/2	12.38	8.82	31.20	17.47
OWC/VU/TU/2	18.87	9.58	30.44	19.63
OWC/VR/TU/2	101.88	11.96	82.81	65.55

Table B.11: Accuracy of Faster RCNN Inception v2, Pure Translation in Depth

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	12.65	5.05	16.11	11.27
WWC/VR/TU/1	13.73	5.46	25.71	14.97
OWC/VU/TU/1	9.24	6.67	19.89	11.93
OWC/VR/TU/1	50.90	7.96	46.75	35.20
WWC/VU/TU/2	10.57	5.79	18.35	11.57
WWC/VR/TU/2	11.05	6.87	28.64	15.52
OWC/VU/TU/2	13.42	7.75	31.50	17.56
OWC/VR/TU/2	39.74	8.71	43.29	30.58

Table B.12: Accuracy of Faster RCNN Inception v2, Pure Rotation

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	11.75	5.30	14.10	10.39
WWC/VR/TU/1	11.88	6.19	20.36	12.81
OWC/VU/TU/1	18.33	6.72	31.74	18.93
OWC/VR/TU/1	32.85	7.95	37.72	26.17
WWC/VU/TU/2	18.87	4.53	24.15	15.85
WWC/VR/TU/2	19.24	5.24	22.54	15.67
OWC/VU/TU/2	19.46	7.29	35.94	20.90
OWC/VR/TU/2	36.60	8.71	54.76	33.36

Table B.13: Accuracy of Faster RCNN Inception v2, Complex Flight

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	11.90	6.09	15.50	11.17
WWC/VR/TU/1	11.82	6.78	21.55	13.39
OWC/VU/TU/1	16.05	7.92	34.09	19.36
OWC/VR/TU/1	42.27	9.70	52.96	34.98
WWC/VU/TU/2	15.62	5.78	21.19	14.20
WWC/VR/TU/2	16.28	5.84	22.82	14.98
OWC/VU/TU/2	19.30	7.58	34.41	20.43
OWC/VR/TU/2	54.87	9.18	55.45	39.83

B.2.2.4 YOLO v2 Root Mean Square Error

Table B.14: Accuracy of YOLO v2, Pure Translation in Side and Height

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	11.77	6.22	18.05	12.01
WWC/VR/TU/1	12.57	7.98	31.07	17.21
WWC/VU/TR/1	11.10	5.91	15.89	10.97
WWC/VR/TR/1	11.32	6.95	19.50	12.59
OWC/VU/TU/1	14.64	10.82	41.40	22.29
OWC/VR/TU/1	12.41	14.11	55.10	27.21
OWC/VU/TR/1	38.72	11.28	39.28	29.76
OWC/VR/TR/1	47.87	20.12	37.09	35.02
WWC/VU/TU/2	12.97	6.73	20.64	13.45
WWC/VR/TU/2	13.80	8.32	35.52	19.21
WWC/VU/TR/2	12.49	5.99	17.06	11.85
WWC/VR/TR/2	12.35	7.18	22.14	13.89
OWC/VU/TU/2	15.39	11.95	38.98	22.11
OWC/VR/TU/2	15.48	15.49	59.16	30.05
OWC/VU/TR/2	21.76	10.40	39.69	23.95
OWC/VR/TR/2	47.57	19.53	37.96	35.02

Table B.15: Accuracy of YOLO v2, Pure Translation in Depth

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	14.04	4.03	16.80	11.62
WWC/VR/TU/1	15.34	4.80	30.49	16.87
WWC/VU/TR/1	12.94	4.31	16.23	11.16
WWC/VR/TR/1	13.13	4.64	17.97	11.91
OWC/VU/TU/1	10.04	8.88	28.55	15.82
OWC/VR/TU/1	11.34	8.70	36.61	18.88
OWC/VU/TR/1	25.53	12.77	35.61	24.64
OWC/VR/TR/1	34.59	20.02	32.18	28.93
WWC/VU/TU/2	10.85	4.53	17.71	11.03
WWC/VR/TU/2	11.14	6.38	30.88	16.14
WWC/VU/TR/2	11.04	5.43	15.38	10.62
WWC/VR/TR/2	10.96	5.26	19.81	12.01
OWC/VU/TU/2	20.30	11.52	46.78	26.20
OWC/VR/TU/2	23.71	14.52	70.63	36.29
OWC/VU/TR/2	27.82	12.86	49.89	30.19
OWC/VR/TR/2	44.87	28.08	50.01	40.99

Table B.16: Accuracy of YOLO v2, Pure Rotation

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	12.09	4.63	16.36	11.03
WWC/VR/TU/1	12.23	5.65	26.58	14.82
WWC/VU/TR/1	12.15	3.98	12.21	9.45
WWC/VR/TR/1	12.14	4.83	15.16	10.71
OWC/VU/TU/1	20.62	10.00	46.75	25.79
OWC/VR/TU/1	22.99	12.16	46.84	27.33
OWC/VU/TR/1	25.99	8.05	29.86	21.30
OWC/VR/TR/1	35.08	11.90	31.43	26.14
WWC/VU/TU/2	19.05	3.98	21.39	14.81
WWC/VR/TU/2	19.45	4.93	22.36	15.58
WWC/VU/TR/2	19.35	3.89	21.97	15.07
WWC/VR/TR/2	19.62	4.57	18.66	14.28
OWC/VU/TU/2	21.07	8.70	27.81	19.19
OWC/VR/TU/2	21.97	9.38	36.38	22.58
OWC/VU/TR/2	26.74	8.79	42.23	25.92
OWC/VR/TR/2	33.96	11.69	44.17	29.94

Table B.17: Accuracy of YOLO v2, Complex Flight

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	12.50	4.30	12.32	9.71
WWC/VR/TU/1	12.49	5.33	18.61	12.15
WWC/VU/TR/1	12.48	4.87	13.98	10.44
WWC/VR/TR/1	12.31	6.21	16.72	11.75
OWC/VU/TU/1	14.15	10.16	45.17	23.16
OWC/VR/TU/1	13.25	5.08	40.64	19.66
OWC/VU/TR/1	27.45	12.30	36.66	25.47
OWC/VR/TR/1	47.36	27.32	40.45	38.38
WWC/VU/TU/2	17.08	3.54	17.68	12.77
WWC/VR/TU/2	18.11	4.30	23.68	15.36
WWC/VU/TR/2	16.21	4.06	15.89	12.06
WWC/VR/TR/2	16.62	4.64	13.20	11.49
OWC/VU/TU/2	17.62	3.96	59.59	27.06
OWC/VR/TU/2	18.31	5.32	65.50	29.71
OWC/VU/TR/2	27.82	11.82	46.67	28.77
OWC/VR/TR/2	41.82	32.65	48.90	41.12

B.2.2.5 Tiny YOLO Root Mean Square Error

Table B.18: Accuracy of Tiny YOLO, Pure Translation in Side and Height

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VR/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VU/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VR/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VU/TR/1	14.28	5.94	43.23	21.15
WWC/VR/TR/1	12.84	6.07	35.40	18.11
OWC/VU/TR/1	11.01	8.52	34.67	18.06
OWC/VR/TR/1	13.88	9.89	43.76	22.51
WWC/VU/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VR/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VU/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VR/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VU/TR/2	14.80	5.13	38.74	19.56
WWC/VR/TR/2	13.51	5.21	28.98	15.90
OWC/VU/TR/2	14.67	8.82	36.38	19.96
OWC/VR/TR/2	16.04	10.40	44.53	23.66

Table B.19: Accuracy of Tiny YOLO, Pure Translation in Depth

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VR/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VU/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VR/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VU/TR/1	13.49	3.77	24.46	13.91
WWC/VR/TR/1	12.75	3.84	21.43	12.67
OWC/VU/TR/1	10.73	8.55	42.21	20.50
OWC/VR/TR/1	12.28	9.36	49.26	23.63
WWC/VU/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VR/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VU/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VR/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VU/TR/2	11.84	3.93	25.84	13.87
WWC/VR/TR/2	10.85	3.67	19.12	11.21
OWC/VU/TR/2	16.48	13.35	60.34	30.06
OWC/VR/TR/2	17.81	15.10	70.05	34.32

Table B.20: Accuracy of Tiny YOLO, Pure Rotation

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VR/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VU/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VR/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VU/TR/1	13.39	3.78	28.18	15.12
WWC/VR/TR/1	13.42	4.27	26.88	14.86
OWC/VU/TR/1	19.51	8.16	34.27	20.65
OWC/VR/TR/1	22.27	9.54	37.65	23.15
WWC/VU/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VR/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VU/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VR/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VU/TR/2	19.99	3.33	32.07	18.46
WWC/VR/TR/2	20.89	3.89	26.00	16.93
OWC/VU/TR/2	17.43	7.61	46.32	23.79
OWC/VR/TR/2	19.21	8.87	40.38	22.82

Table B.21: Accuracy of Tiny YOLO, Complex Flight

Test Configuration/Trial	**RMS Error in Side x (cm)**	**RMS Error in Height y (cm)**	**RMS Error in Depth z (cm)**	**Average RMS Error (cm)**
WWC/VU/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VR/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VU/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VR/TU/1	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VU/TR/1	13.50	3.61	23.96	13.69
WWC/VR/TR/1	12.53	3.66	16.13	10.78
OWC/VU/TR/1	13.63	8.61	34.15	18.80
OWC/VR/TR/1	16.20	11.12	47.37	24.90
WWC/VU/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VR/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VU/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
OWC/VR/TU/2	Nothing Detected	Nothing Detected	Nothing Detected	Nothing Detected
WWC/VU/TR/2	16.51	3.49	30.15	16.72
WWC/VR/TR/2	16.94	3.19	22.51	14.21
OWC/VU/TR/2	16.75	8.53	43.44	22.91
OWC/VR/TR/2	18.64	9.71	53.25	27.20

B.2.3 Mean Average Precision

B.2.3.1 SSD MobileNet v1 mAP

Table B.22: Consistency of SSD MobileNet v1, Pure Translation in Side and Height

Test Configuration/Trial	Average Precision
WWC/VU/TU/1	0.8442
WWC/VR/TU/1	0.8888
OWC/VU/TU/1	0.8226
OWC/VR/TU/1	0.9488
WWC/VU/TU/2	0.9096
WWC/VR/TU/2	0.9329
OWC/VU/TU/2	0.0444
OWC/VR/TU/2	0.0688

Table B.23: Consistency of SSD MobileNet v1, Pure Translation in Depth

Test Configuration/Trial	Average Precision
WWC/VU/TU/1	0.9277
WWC/VR/TU/1	0.7954
OWC/VU/TU/1	0.6229
OWC/VR/TU/1	0.7078
WWC/VU/TU/2	0.9572
WWC/VR/TU/2	0.9449
OWC/VU/TU/2	0.6230
OWC/VR/TU/2	0.0265

Table B.24: Consistency of SSD MobileNet v1, Pure Rotation

Test Configuration/Trial	Average Precision
WWC/VU/TU/1	0.9130
WWC/VR/TU/1	0.9409
OWC/VU/TU/1	0.9353
OWC/VR/TU/1	0.7953
WWC/VU/TU/2	0.8445
WWC/VR/TU/2	0.8827
OWC/VU/TU/2	0.5000
OWC/VR/TU/2	0.5150

Table B.25: Consistency of SSD MobileNet v1, Complex Flight

Test Configuration/Trial	Average Precision
WWC/VU/TU/1	0.7921
WWC/VR/TU/1	0.9205
OWC/VU/TU/1	0.8088
OWC/VR/TU/1	0.9732
WWC/VU/TU/2	0.8507
WWC/VR/TU/2	0.8295
OWC/VU/TU/2	0.0278
OWC/VR/TU/2	0

B.2.3.2 SSD Inception v2 mAP

Table B.26: Consistency of SSD Inception v2, Pure Translation in Side and Height

Test Configuration/Trial	Average Precision
WWC/VU/TU/1	0.8045
WWC/VR/TU/1	0.7671
OWC/VU/TU/1	0.7259
OWC/VR/TU/1	0.7098
WWC/VU/TU/2	0.8045
WWC/VR/TU/2	0.8046
OWC/VU/TU/2	0.6027
OWC/VR/TU/2	0.6453

Table B.27: Consistency of SSD Inception v2, Pure Translation in Depth

Test Configuration/Trial	Average Precision
WWC/VU/TU/1	0.6485
WWC/VR/TU/1	0.5247
OWC/VU/TU/1	0.5726
OWC/VR/TU/1	0.9440
WWC/VU/TU/2	0.8541
WWC/VR/TU/2	0.8227
OWC/VU/TU/2	0.4941
OWC/VR/TU/2	0.0264

Table B.28: Consistency of SSD Inception v2, Pure Rotation

Test Configuration/Trial	Average Precision
WWC/VU/TU/1	0.8477
WWC/VR/TU/1	0.8252
OWC/VU/TU/1	0.8160
OWC/VR/TU/1	0.7948
WWC/VU/TU/2	0.7467
WWC/VR/TU/2	0.6309
OWC/VU/TU/2	0.4715
OWC/VR/TU/2	0.5034

Table B.29: Consistency of SSD Inception v2, Complex Flight

Test Configuration/Trial	Average Precision
WWC/VU/TU/1	0.9040
WWC/VR/TU/1	0.7949
OWC/VU/TU/1	0.9137
OWC/VR/TU/1	0.1496
WWC/VU/TU/2	0.7358
WWC/VR/TU/2	0.5800
OWC/VU/TU/2	0.0294
OWC/VR/TU/2	0.1269

B.2.3.3 Faster RCNN Inception v2 mAP

Table B.30: Consistency of Faster RCNN Inception v2 , Pure Translation in Side and Height

Test Configuration/Trial	Average Precision
WWC/VU/TU/1	1.0000
WWC/VR/TU/1	1.0000
OWC/VU/TU/1	0.9555
OWC/VR/TU/1	0.9800
WWC/VU/TU/2	1.0000
WWC/VR/TU/2	1.0000
OWC/VU/TU/2	0.7797
OWC/VR/TU/2	0.8741

Table B.31: Consistency of Faster RCNN Inception v2 , Pure Translation in Depth

Test Configuration/Trial	**Average Precision**
WWC/VU/TU/1	1.0000
WWC/VR/TU/1	1.0000
OWC/VU/TU/1	0.9739
OWC/VR/TU/1	0.9866
WWC/VU/TU/2	1.0000
WWC/VR/TU/2	1.0000
OWC/VU/TU/2	0.9827
OWC/VR/TU/2	0.9775

Table B.32: Consistency of Faster RCNN Inception v2 , Pure Rotation

Test Configuration/Trial	**Average Precision**
WWC/VU/TU/1	1.0000
WWC/VR/TU/1	1.0000
OWC/VU/TU/1	0.9810
OWC/VR/TU/1	0.9890
WWC/VU/TU/2	1.0000
WWC/VR/TU/2	1.0000
OWC/VU/TU/2	0.9840
OWC/VR/TU/2	0.9710

Table B.33: Consistency of Faster RCNN Inception v2 , Complex Flight

Test Configuration/Trial	**Average Precision**
WWC/VU/TU/1	1.0000
WWC/VR/TU/1	1.0000
OWC/VU/TU/1	0.9520
OWC/VR/TU/1	0.9691
WWC/VU/TU/2	1.0000
WWC/VR/TU/2	1.0000
OWC/VU/TU/2	0.9255
OWC/VR/TU/2	0.9440

B.2.3.4 YOLO v2 mAP

Table B.34: Consistency of YOLO v2, Pure Translation in Side and Height

Test Configuration/Trial	Average Precision
WWC/VU/TU/1	1.0000
WWC/VR/TU/1	0.9987
OWC/VU/TU/1	0.5151
OWC/VR/TU/1	0.4585
WWC/VU/TR/1	1.0000
WWC/VR/TR/1	1.0000
OWC/VU/TR/1	0.8512
OWC/VR/TR/1	0.9525
WWC/VU/TU/2	1.0000
WWC/VR/TU/2	0.9998
OWC/VU/TU/2	0.6190
OWC/VR/TU/2	0.5632
WWC/VU/TR/2	1.0000
WWC/VR/TR/2	1.0000
OWC/VU/TR/2	0.8283
OWC/VR/TR/2	0.9727

Table B.35: Consistency of YOLO v2, Pure Translation in Depth

Test Configuration/Trial	Average Precision
WWC/VU/TU/1	0.9945
WWC/VR/TU/1	0.9685
OWC/VU/TU/1	0.6578
OWC/VR/TU/1	0.5714
WWC/VU/TR/1	1.0000
WWC/VR/TR/1	1.0000
OWC/VU/TR/1	0.8036
OWC/VR/TR/1	0.9738
WWC/VU/TU/2	0.9925
WWC/VR/TU/2	0.9755
OWC/VU/TU/2	0.4207
OWC/VR/TU/2	0.2210
WWC/VU/TR/2	1.0000
WWC/VR/TR/2	1.0000
OWC/VU/TR/2	0.6906
OWC/VR/TR/2	0.9132

Table B.36: Consistency of YOLO v2, Pure Rotation

Test Configuration/Trial	**Average Precision**
WWC/VU/TU/1	1.0000
WWC/VR/TU/1	1.0000
OWC/VU/TU/1	0.4610
OWC/VR/TU/1	0.3554
WWC/VU/TR/1	1.0000
WWC/VR/TR/1	1.0000
OWC/VU/TR/1	0.8396
OWC/VR/TR/1	0.9726
WWC/VU/TU/2	1.0000
WWC/VR/TU/2	0.9937
OWC/VU/TU/2	0.6445
OWC/VR/TU/2	0.5268
WWC/VU/TR/2	1.0000
WWC/VR/TR/2	1.0000
OWC/VU/TR/2	0.8779
OWC/VR/TR/2	0.9924

Table B.37: Consistency of YOLO v2, Complex Flight

Test Configuration/Trial	**Average Precision**
WWC/VU/TU/1	0.9992
WWC/VR/TU/1	0.9987
OWC/VU/TU/1	0.4643
OWC/VR/TU/1	0.2002
WWC/VU/TR/1	1.0000
WWC/VR/TR/1	1.0000
OWC/VU/TR/1	0.6967
OWC/VR/TR/1	0.9302
WWC/VU/TU/2	0.9653
WWC/VR/TU/2	0.9022
OWC/VU/TU/2	0.3164
OWC/VR/TU/2	0.2145
WWC/VU/TR/2	1.0000
WWC/VR/TR/2	1.0000
OWC/VU/TR/2	0.7445
OWC/VR/TR/2	0.9302

B.2.3.5 Tiny YOLO mAP

Table B.38: Consistency of Tiny YOLO, Pure Translation in Side and Height

Test Configuration/Trial	**Average Precision**
WWC/VU/TU/1	0
WWC/VR/TU/1	0
OWC/VU/TU/1	0
OWC/VR/TU/1	0
WWC/VU/TR/1	0.7731
WWC/VR/TR/1	0.5559
OWC/VU/TR/1	0.7192
OWC/VR/TR/1	0.7097
WWC/VU/TU/2	0
WWC/VR/TU/2	0
OWC/VU/TU/2	0
OWC/VR/TU/2	0
WWC/VU/TR/2	0.7669
WWC/VR/TR/2	0.3773
OWC/VU/TR/2	0.9058
OWC/VR/TR/2	0.8952

Table B.39: Consistency of Tiny YOLO, Pure Translation in Depth

Test Configuration/Trial	**Average Precision**
WWC/VU/TU/1	0
WWC/VR/TU/1	0
OWC/VU/TU/1	0
OWC/VR/TU/1	0
WWC/VU/TR/1	0.7680
WWC/VR/TR/1	0.3933
OWC/VU/TR/1	0.8501
OWC/VR/TR/1	0.8603
WWC/VU/TU/2	0
WWC/VR/TU/2	0
OWC/VU/TU/2	0
OWC/VR/TU/2	0
WWC/VU/TR/2	0.8012
WWC/VR/TR/2	0.4419
OWC/VU/TR/2	0.8267
OWC/VR/TR/2	0.8162

Table B.40: Consistency of Tiny YOLO, Pure Rotation

Test Configuration/Trial	**Average Precision**
WWC/VU/TU/1	0
WWC/VR/TU/1	0
OWC/VU/TU/1	0
OWC/VR/TU/1	0
WWC/VU/TR/1	0.9222
WWC/VR/TR/1	0.7626
OWC/VU/TR/1	0.8860
OWC/VR/TR/1	0.8270
WWC/VU/TU/2	0
WWC/VR/TU/2	0
OWC/VU/TU/2	0
OWC/VR/TU/2	0
WWC/VU/TR/2	0.8953
WWC/VR/TR/2	0.6536
OWC/VU/TR/2	0.9486
OWC/VR/TR/2	0.9436

Table B.41: Consistency of Tiny YOLO, Complex Flight

Test Configuration/Trial	**Average Precision**
WWC/VU/TU/1	0
WWC/VR/TU/1	0
OWC/VU/TU/1	0
OWC/VR/TU/1	0
WWC/VU/TR/1	0.9227
WWC/VR/TR/1	0.7034
OWC/VU/TR/1	0.7183
OWC/VR/TR/1	0.7274
WWC/VU/TU/2	0
WWC/VR/TU/2	0
OWC/VU/TU/2	0
OWC/VR/TU/2	0
WWC/VU/TR/2	0.6576
WWC/VR/TR/2	0.5121
OWC/VU/TR/2	0.8783
OWC/VR/TR/2	0.7855

B.3 Camera Calibration Files

In this Section, 11 trials of camera calibrations are conducted and the calibration files are listed below.

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
  data: [565.6125151546354, 0, 317.3316782430719, 0,
      ↪ 564.8481170587216, 184.2906650074466, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.5926584270957175, 0.3603663705471096,
      ↪ -0.001293012518993791, 0.008236529105444059, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [447.2406005859375, 0, 326.1898806961472, 0, 0,
      ↪ 528.8070068359375, 184.6186261898401, 0, 0, 0, 1, 0]
```

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
  data: [527.859194549529, 0, 323.8411857372338, 0,
      ↪ 528.4889458520696, 189.529178855309, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.5452685319365548, 0.3169366695208474,
      ↪ 0.002525160632601806, 0.0002668957084609045, 0]
rectification_matrix:
  rows: 3
```

```
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [416.145263671875, 0, 325.4994652538298, 0, 0,
     ↪ 492.4153747558594, 191.5844334897729, 0, 0, 0, 1, 0]
```

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
  data: [533.4164868159106, 0, 320.6058446431378, 0,
     ↪ 533.3226937158951, 183.7123271397635, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.5504663011264753, 0.3004454342326453,
     ↪ -0.0004465554446928154, -0.001662564721472756, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [415.9227294921875, 0, 318.6531988442439, 0, 0,
     ↪ 497.6212463378906, 184.1703457528929, 0, 0, 0, 1, 0]
```

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
  data: [559.0388159999543, 0, 309.4420691600483, 0,
     ↪ 561.0016581328583, 179.1097742137247, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
```

```
  data: [-0.5680798050660655, 0.3816376063256253,
    ↪ 0.004027969342063515, 0.003458879115659362, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [454.368408203125, 0, 309.9864112689684, 0, 0,
    ↪ 526.6734619140625, 179.8893351986335, 0, 0, 0, 1, 0]
```

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
  data: [556.022707285772, 0, 330.0925071971059, 0,
    ↪ 554.8050302889992, 184.6446954326166, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.5567127982042523, 0.3318254627238606,
    ↪ 0.002253735786806615, -0.001817605270684056, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [445.7396240234375, 0, 331.7946918618909, 0, 0,
    ↪ 520.239501953125, 185.7993614255683, 0, 0, 0, 1, 0]
```

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
  data: [554.3974113785129, 0, 304.2240641633496, 0,
    ↪ 556.2971778990975, 170.839601003777, 0, 0, 1]
distortion_model: plumb_bob
```

```
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.6584872683956912, 0.4991870727128286,
     ↪ 0.001310314818277521, -0.0006481000399672231, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [440.9514465332031, 0, 301.4554650789833, 0, 0,
     ↪ 515.1989135742188, 169.649437021013, 0, 0, 0, 1, 0]
```

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
  data: [553.6658049122539, 0, 326.3733014243585, 0,
     ↪ 552.6879159861015, 179.0311456556649, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.5543740018546095, 0.3114736151678509,
     ↪ 0.0009804161252434315, -0.002206482637588245, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [439.7734985351562, 0, 326.3617064027385, 0, 0,
     ↪ 518.2238159179688, 179.1186808660113, 0, 0, 0, 1, 0]
```

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
```

```
  data: [550.2045894558636, 0, 327.396624085826, 0,
    ↪ 551.5090498256899, 183.267719797952, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.5608616551983711, 0.3542765088398399,
    ↪ -0.000819696270168569, 0.003755492662011094, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [441.7192077636719, 0, 334.0961309924387, 0, 0,
    ↪ 516.7183227539062, 183.5381596549014, 0, 0, 0, 1, 0]
```

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
  data: [552.8298230303856, 0, 333.5446031149884, 0,
    ↪ 552.8470541636079, 186.6372832667655, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.5519367747616147, 0.3248271119450988,
    ↪ -0.001951074094604523, -0.004173892436812423, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [443.1218872070312, 0, 333.7817476913369, 0, 0,
    ↪ 518.5982055664062, 187.1101233664667, 0, 0, 0, 1, 0]
```

```
image_width: 640
image_height: 360
camera_name: ardrone_front
```

```
camera_matrix:
  rows: 3
  cols: 3
  data: [563.0418985537083, 0, 311.9145325150032, 0,
     ↪ 562.4914915941853, 178.7635884646983, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.654118619133408, 0.438262486045934,
     ↪ -0.008352828143861415, -0.002656636159456569, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [436.4547729492188, 0, 306.0777826667618, 0, 0,
     ↪ 521.868896484375, 176.6082298605397, 0, 0, 0, 1, 0]
```

```
image_width: 640
image_height: 360
camera_name: ardrone_front
camera_matrix:
  rows: 3
  cols: 3
  data: [539.633067245398, 0, 323.9711190650898, 0,
     ↪ 539.2165660769166, 187.4086724558219, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.5354407767265371, 0.2801137753188095,
     ↪ 0.0006743725518513881, -0.002628061711864213, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [424.4468994140625, 0, 322.315947960371, 0, 0,
     ↪ 504.7111511230469, 188.633070810738, 0, 0, 0, 1, 0]
```