Improved Ground-Based Monocular Visual Odometry Estimation using Inertially-Aided Convolutional Neural Networks

Josiah D. Watson

IMPROVED GROUND-BASED MONOCULAR VISUAL ODOMETRY ESTIMATION USING INERTIALLY-AIDED CONVOLUTIONAL NEURAL NETWORKS

THESIS

Josiah D Watson

AFIT-ENG-MS-20-M-072

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-20-M-072

Improved Ground-Based Monocular Visual Odometry Estimation using Inertially-Aided Convolutional Neural Networks

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Electrical Engineering

Josiah D Watson, B.S.Cp.E.

March 26, 2020

AFIT-ENG-MS-20-M-072

Improved Ground-Based Monocular Visual Odometry Estimation using Inertially-Aided Convolutional Neural Networks

THESIS

Josiah D Watson, B.S.Cp.E.

Committee Membership:

John F Raquet, Ph.D
Chair

Joseph A Curro, Ph.D
Member

Robert C Leishman, Ph.D
Member

AFIT-ENG-MS-20-M-072

Abstract

Visual Odometry (VO) is the estimation of a camera's motion between sequential image pairs. Many traditional and Deep Learning (DL) VO methods have already been researched, including the use of Convolutional Neural Networks (CNNs) for VO. Monocular, ground-based VO is of particular importance for environments where satellite-based navigation systems are not available. While CNNs can estimate frame-to-frame (F2F) motion even with monocular images, additional inputs can improve VO predictions. In this thesis, a FlowNetS-based [1] CNN architecture estimates VO using sequential images from the KITTI Odometry dataset [2]. For each of three output types (full six degrees of freedom (6-DoF), Cartesian translation, and transitional scale), a baseline network with only image pair input is compared with a nearly identical architecture that is also given an additional rotation estimate such as from an Inertial Navigation System (INS).

The inertially-aided networks show an order of magnitude improvement over the baseline networks when predicting rotation. The y component rotation prediction errors decrease from a baseline Root Mean Squared Error (RMSE) of 0.09058 deg to an INS-aided RMSE of 0.01447 deg. However, the aided RMSE is still worse than the INS input RMSE of 0.00268 deg. INS-aiding does not necessarily help the translation predictions either with a maximum z translation RMSE improvement of only 0.003 meters on the test set. A full-trajectory analysis gives similar results for the rotation prediction errors, but the translation errors are much larger. When predicting translation, the baseline networks actually perform better than the INS-aided networks in most cases. The INS-aided neural networks are also tested for sensitivity to angular random walk (ARW) and bias errors in the sensor measurements.

Acknowledgements

Many people have made this thesis possible through their guidance and support over the past 18 months. As my research advisor, Dr. Raquet has provided timely input and direction that greatly added to my understanding of navigation and visual odometry concepts. I appreciate the time he has invested in me and this project and his ability to point me in the right direction. Capt Curro served as my second advisor and has been an invaluable resource on neural networks and their applications. I appreciate the ways he has challenged me to think through each research decision. I also am grateful for Dr. Leishman's helpful input and for his willingness to be on my committee in the midst of his many other responsibilities. Beyond those on my committee, the love and support my family has given me during this time has been extraordinary. They have listened well when I tried to explain what I was working on, motivated me when I would lose focus, and cheered me on every step of the way. Most importantly, my Lord and Savior Jesus Christ has been with me throughout this entire process. He has placed these people around me and given me the ability to understand and apply these concepts. Without Him, none of this would have been possible.

Josiah D Watson

Table of Contents

Page

Abstract iv

Acknowledgements v

List of Figures viii

List of Tables ix

I. Introduction 1

1.1 Problem Background 1
1.2 Research Objectives 3
1.3 Assumptions 3
1.4 Document Overview 4

II. Background and Literature Review 5

2.1 Overview 5
2.2 Visual Odometry (VO) 5
2.2.1 Camera Calibration 6
2.2.2 Scale Estimation 7
2.2.3 Traditional Visual Odometry Methods 9
2.2.4 Deep Learning Visual Odometry Methods 15
2.3 Coordinate Notation 19
2.3.1 Coordinate Points 19
2.3.2 Coordinate Frames 20
2.3.3 Special Euclidean Group Matrices 20
2.3.4 Coordinate Frame Conversion 21
2.3.5 Spherical Coordinates 22
2.4 Inertial Errors 23
2.4.1 Bias Error 23
2.4.2 Angular Random Walk Error 23
2.5 Summary 24

III. Methodology 25

3.1 Overview 25
3.2 KITTI Odometry Dataset and Pre-Processing 25
3.2.1 Dataset Statistics 25
3.2.2 Image Pairs 27
3.2.3 Truth Labels 30
3.2.4 Simulated INS Rotations 36
3.3 Model Architectures 38

Page

3.3.1 CNN Architectures 38
3.3.2 VO Model Architectures 39
3.4 Training 42
3.4.1 Training-Validation-Test Split 42
3.4.2 Training Parameters 43
3.5 Procedure 43
3.6 Summary 44

IV. Results and Analysis 45

4.1 Overview 45
4.2 CNN Selection and Hyper-parameter Tuning 45
4.3 Model Aiding Evaluation 48
4.3.1 Training 48
4.3.2 Evaluation Results 51
4.4 Full Trajectory Evaluation 60
4.5 INS-Aided Model Sensitivity Tests 64
4.5.1 Bias Error Sensitivity 64
4.5.2 ARW Error Sensitivity 68
4.6 Summary 68

V. Conclusions 70

5.1 Significance of Results 70
5.2 Future Work 71
Bibliography 74
Acronyms 82

List of Figures

Figure Page

1 Pinhole camera model diagram taken from [3]. 7

2 KITTI Odometry vehicle setup [4]. 26

3 Representative images in KITTI Odometry dataset [2]. 28

4 Abstract VO model architecture. 40

5 Baseline model architecture diagram for full 6-DoF VO prediction. 41

6 INS-aided model architecture diagram for full 6-DoF VO prediction. 42

7 Training and validation loss history for CNN selection tests. 47

8 Training and validation loss history for the six VO models. 49

9 RMSE of the rotation and translation magnitude for each training epoch of the 6-DoF models. 50

10 Test set rotation errors for both the baseline and INS-aided 6-DoF VO models. 55

11 Test set translation errors for both the baseline and INS-aided 6-DoF VO models. 56

12 Test set errors for both the baseline and INS-aided Cartesian translation VO models. 57

13 Test set errors for both the baseline and INS-aided Scale VO models. 58

15 Rotation error standard deviation sensitivity to IMU errors for the INS-Aided 6-DoF VO models. 65

16 Translation error mean sensitivity to IMU errors for the INS-Aided VO models 66

17 Translation error standard deviation sensitivity to IMU errors for the INS-Aided VO models 67

List of Tables

Table Page

1 Number of frames per camera for the first 11 sequences in the KITTI Odometry dataset [2]. 27

2 Truth data statistics for KITTI Odometry [2] sequences 00-09. 34

3 Truth data statistics for KITTI Odometry [2] sequence 10. 34

4 Geometric data normalization parameters. 36

5 INS data RMSE values for different subsets of the KITTI Odometry dataset [2]. 37

6 FlowNet [1] CNN architecture for a small input image size of $(160 \times 608 \times 2)$ 39

7 Characteristics of the six VO models based on decisions shown in Figure 4 40

8 Comparison of CNN selection model performance on the validation set data. 47

9 Best VO model validation set performance. 51

10 Test set RMSE results when the test set mean values are used as the VO prediction for each output. 52

11 Test set RMSE results for the six VO models. 52

12 Test set error mean and standard deviation results for the six VO models. 53

13 Test set RMSE results on turning and non-turning image pairs for the six VO models. 59

14 Test set error mean and standard deviation results on turning and non-turning image pairs for the six VO models. 60

15 Sequence 10 RMSE results when the sequence 10 mean values are used as the VO prediction for each output. 62

Table Page

16 Sequence 10 RMSE results for the six VO models. 63

17 Sequence 10 RMSE results on turning and non-turning image pairs for the six VO models. 63

Improved Ground-Based Monocular Visual Odometry Estimation using Inertially-Aided Convolutional Neural Networks

I. Introduction

1.1 Problem Background

Visual Odometry (VO) is a vital area of research because it can obtain a navigation estimate even when Global Positioning System (GPS) data is not available. Satellite-based navigation systems transmit position information using radio frequency signals. Because of this, GPS and other Global Navigation Satellite Systems (GNSSs) can be jammed or spoofed. Beyond malicious attacks on GPS, building materials or water can attenuate radio signals making them unobtainable in some urban, indoor, and underwater environments. Mars lacks a satellite constellation entirely. Thus, while satellite-based systems clearly provide the most accurate position information, other navigation methods are necessary for times when GPS is not available.

Vision-based navigation and VO in particular provide such alternatives, albeit with some drawbacks. Most importantly, VO uses cameras instead of relying on receiving radio signals, so it cannot be jammed like GPS. Traditional optical cameras also do not send signals into the environment so they cannot be detected like RADAR, SONAR, or LIDAR sensors. Visual Odometry (VO) is the calculation of a camera's ego-motion from one image frame to the next image frame in a sequence. Many methods to obtain these frame-to-frame (F2F) movements have been developed [5, 6, 7, 8]. Traditional indirect methods extract image features and match them between image frames. These image features can be interesting pixel locations or calculated optical flow vectors that track the movement of pixel intensities [9]. VO algorithms then use image geometry and changes in perspective to calculate the camera's motion. In addition to indirect methods, direct VO methods minimize photometric consistency error in the pixel intensity values to determine the camera's movement between frames [8, 9]. More recently, researchers have used Deep Learning (DL) models like Convolutional Neural Networks (CNNs) [10, 11, 12] and Recurrent Convolutional Neural Networks (RCNNs) [9, 13, 14] to digest sequential images and estimate VO.

While VO methods produce promising results in specific environments, vision-based navigation does have major drawbacks. For instance, sunlight, shadows, weather effects, and seasonal changes can all inhibit or invalidate the VO results. Also, in certain environments like aerial navigation over the ocean, there may be too few features for the VO calculation. Even with these drawbacks, VO is still a useful navigation method.

Beyond these environment-based drawbacks, two-dimensional (2D) images do not include absolute scale information. Traditional VO algorithms using a monocular camera system can fully determine three-dimensional (3D) rotation and the direction of the translation, but not the absolute magnitude (scale) of the translational movement. However, the system can determine absolute scale when it is given some external sense of scale. Stereo camera systems successfully inject absolute scale into a VO system using a known baseline distance between the cameras [15, 16, 17, 18, 19]. However, all stereo systems degenerate into a monocular system when features in the scene are at much greater distances than the baseline separation distance between the cameras. Monocular systems can still learn a sense of scale over time though using methods like stadiometry, parallax, or depth from defocus.

Many researchers have used DL for monocular VO [9, 11, 12, 13], but very few have considered the use of Inertial Navigation System (INS) aiding in neural networks.

Of the ones that have, most use a Recurrent Neural Network (RNN) to interpret the Inertial Measurement Unit (IMU) data before they are combined with the transformed images [14, 20].

1.2 Research Objectives

This thesis considers the use of inertial aiding for CNNs estimating monocular, ground-based VO. The main goal is to determine if aiding a CNN with rotation estimates helps the network obtain a more accurate VO estimate. The system learns to predict the absolute scale of the camera's motion through statistical training of a neural network. Each DL model is trained and evaluated using the KITTI Odometry dataset [2]. Three different types of network outputs are considered: full six degrees of freedom (6-DoF) estimation, 3D Cartesian translation estimation, and translation magnitude (scale) estimation. For each of the three output types, two models are trained: one with INS aiding and one without. These six models are evaluated and compared to determine the effect of INS aiding on VO estimation.

1.3 Assumptions

In order for the neural networks to learn translational scale estimation in this thesis, two constraints are employed. First, the KITTI Odometry dataset includes the true position and orientation of the vehicle. Without this data, supervised learning could not be used to train the neural networks in this thesis to estimate absolute translation. Second, the dataset was collected using the same camera setup throughout. The camera was kept in the same position relative to the vehicle and by extension relative to the ground. This serves as a sort of baseline distance that constrains the relative scale of the images. Since the relative image scale is constrained and the absolute scale is known from the truth data, the absolute scale of this dataset can be inferred from the images using supervised learning. However, because the training relies on this fixed relationship between the height of the camera and the absolute scale of the truth data, the DL models would not necessarily be able to predict VO accurately, without retraining, for a dataset with a different baseline or camera focal length.

1.4 Document Overview

A review of relevant research and background information is given in Chapter II. This includes a survey of VO estimation methods in Section 2.2, and overviews of coordinate notation and INS errors in Sections 2.3 and 2.4, respectively.

Chapter III covers the methodology used for experimental setup and evaluation. An overview of the data contained in the KITTI Odometry dataset can be found in Section 3.2 along with specifics on how the data was pre-processed for this thesis. The architectures of the CNN and VO estimation models are described in Section 3.3. Each model was trained with the parameters described in Section 3.4. The overall procedure for this thesis is summarized sequentially in Section 3.5.

The results of the training and evaluation methods are given in Chapter IV. Section 4.2 shows the training and validation results for the four CNN selection models. The training and evaluation results of the six VO models are displayed in Section 4.3 and the effect of INS aiding is analyzed. A full trajectory evaluation in Section 4.4 is used to visualize the integrated results of the VO estimation. Section 4.5 details the effect of errors in the INS aiding input on the prediction outputs of the trained DL models.

Finally, the conclusions are given in Chapter V. This includes a summary of the results and their significance in Section 5.1 and a recommendation for future work in Section 5.2.

II. Background and Literature Review

2.1 Overview

This chapter provides an brief review of other research literature related to Visual Odometry (VO) (Section 2.2). This includes traditional feature-based VO methods (Section 2.2.3) as well as more recent Deep Learning (DL) methods (Section 2.2.4). In addition to the literature review, this chapter also introduces the background information necessary to understand this thesis. It presents an overview of coordinate frames and notation (Section 2.3), including special Euclidean group (SE(3)) matrices, and a brief explanation of Inertial Navigation System (INS) errors (Section 2.4).

2.2 Visual Odometry (VO)

The term ”Visual Odometry” was popularized by [16]. Wheel odometry uses the number of wheel rotations and the known radius of the wheel to determine the velocity of a ground vehicle [6]. Similarly, VO uses sequential images from a camera to determine the camera's change in position. When the camera or cameras used for this task are fixed to an object, the VO can be used to describe the movement of the object.

Compared to other navigation methods, VO has many advantages. The main advantage over GPS is that VO does not rely on radio signals to obtain a navigation solution. This means that VO cannot be jammed or spoofed like satellite-based methods. The camera's used for VO also do not emit any detectable energy, unlike sound or LIDAR sensors. VO has an advantage over wheel odometry because its accuracy is not degraded when travelling over rough terrain [6]. The relative cost of cameras is also much less than some navigation sensors [7, 8]. In addition, given the necessary configuration and conditions, a VO system can be used to get a full six degrees of freedom (6-DoF) navigation solution that describes both the rotation and translation of the camera's movement.

While VO has many advantages, it does have some important drawbacks. Optical cameras used for VO require adequate lighting in the scene, mostly stationary objects in the scene, and sufficient overlap in the field of view from one image to the next [6]. Because of these effects, weather or other visibility distance restrictions can reduce the accuracy of or completely inhibit the VO calculation. In addition, the frame rate of the camera must be high enough to ensure that there is adequate overlap between images even when the camera is moving at its fastest or when it is turning. The trade-off is that a higher number of frames increases the amount of drift error in the overall trajectory, making a larger distance between images preferable. This must be balanced with the necessity of overlap in the field of view.

2.2.1 Camera Calibration

The type of camera calibration depends on the mathematical camera model being used. The most common camera model, and the one used in this thesis, is the pinhole model (shown in Figure 1). A discussion of the catadioptric projection and spherical camera models can be found in [6]. Camera calibration is used to the remove distortion effects of the camera lens, so that the correspondence between the two-dimensional (2D) image plane and the three-dimensional (3D) environment is accurate. The camera calibration can be used to remap pixels from a distorted image to pixels in an undistorted image. This allows for accurate geometric calculations to be made based on the idea that the camera is a single point looking out from behind the image plane. Under this assumption, each pixel value in an image can be considered a measurement of the amount of light entering the camera at a specific range of angles relative to the principle point. The principle point is the place usually

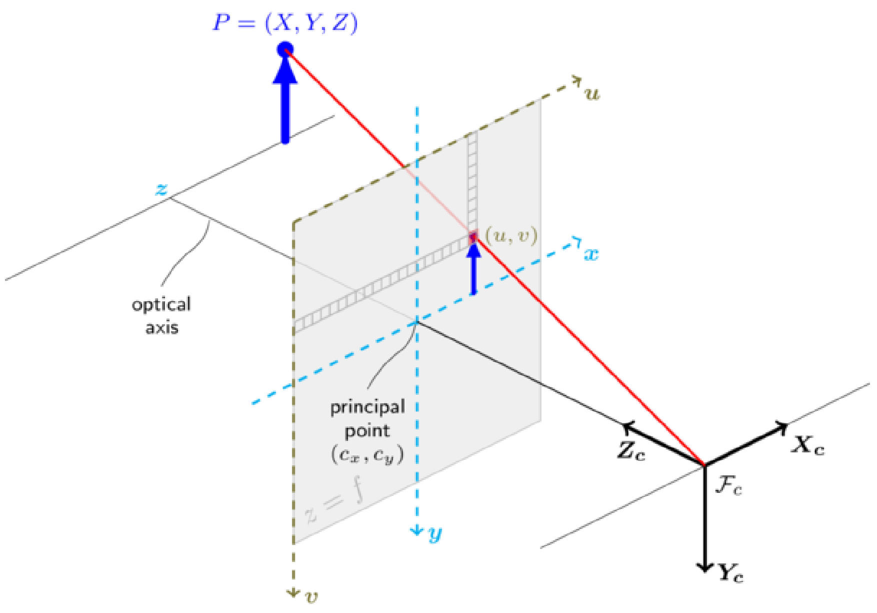


Figure 1: Pinhole camera model diagram taken from [3]. The camera coordinate frame shown on the right side of the diagram is facing the scene and looking through the image frame. This coordinate system is the same one used to express the pose coordinate frames in the KITTI Odometry dataset [2]

in the center of the image plane where the light coming into the camera is at both a vertical and horizontal angle of 0. This camera calibration is also necessary for using epipolar geometry and the essential matrix method for computing camera movement [6].

For stereo systems, the images also need to be rectified. According to [7], image rectification remaps the pixels in the stereo image pair so the epipolar lines of the left and right image are horizontal and aligned with one another. This makes feature correspondences between the stereo image pair easier to find [7].

2.2.2 Scale Estimation

While 2D images do not have absolute scale, they do include relative scale information. Images can be used to determine the 3D rotation and direction of translation. However, the magnitude of the translation cannot be calculated without an external source.

Many researchers use stereo camera systems to solve the scale estimation issue [15, 16, 17, 18, 19]. In this case, two cameras are placed at a fixed baseline distance apart from one another. Since these cameras are facing the same direction, the distance to features that appear in both cameras can be determined proportional to the baseline separation distance between cameras. Since this baseline distance is known, the absolute scale of the camera's motion can be fully determined. While stereo camera systems can produce good results, they do require precise timing to synchronize the two cameras and calibration and rectification to compare camera perspectives. In addition, when the features in the scene are much further from the cameras than the baseline distance between the cameras, the stereo system degenerates into a monocular system. This occurs because the perspective differences between the stereo images are indistinguishable.

Because of this, monocular VO is another important area of study. Monocular camera systems are cheaper and easier to use than stereo systems. They still require adequate camera calibration to ensure that image features are geometrically accurate regardless of where they appear in the image.

Monocular systems also require an adequate scale insertion method. Scale can be determined in many ways. This is observable with humans as well as camera systems. Even when a person only has one eye, absolute scale can be inferred from depth from defocus, parallax, and stadiometry. Depth from defocus (or depth from focus) is the relative increased blurriness of objects as they get further from the focal point of the image. Parallax is the relative shift of objects due to change in perspective. Objects that are further away will appear to shift at a slower rate than objects that are close. Like stereo systems, parallax also degenerates when the relative shift changes of features in the scene are indistinguishable because they are too far away. Stadiometry uses objects present in the images that have a known absolute scale. For instance, if a tree with a known height is completely visible in an image, the absolute scale of the image can be determined including the camera's distance from the tree. With ground vehicles, the camera is at a fixed height above the ground plane. This reduces the amount of perspective change from frame to frame making the scale estimation easier to calculate. Beyond these image-based methods, scale can also be inserted into the system using data from an external sensor like a Global Positioning System (GPS) receiver or an Inertial Measurement Unit (IMU). The main drawback of using an additional sensor is that precise timing is required to synchronize measurements from the external sensor with the camera images.

2.2.3 Traditional Visual Odometry Methods

Traditional VO methods can be categorized as feature based or appearance based. Feature-based methods use the movement of interest keypoints in the image to determine camera motion. Appearance-based methods use the movement of pixel intensity values. See [6, 7, 8] for more information on both feature-based and appearance-based methods.

2.2.3.1 Feature-Based VO Methods

Feature-based indirect methods were first proposed in the early 1980s [6, 8]. These methods rely on the extraction of features from the images to determine the camera's motion. This greatly reduces the complexity and computation time that would be needed to analyze the entire image [6]. However, these methods can fail when feature correspondences between images cannot be computed. For instance, this can occur when the camera moves or rotates too much between frames.

Feature-Based VO Pipeline Over time the traditional VO pipeline has developed. Here is a summary of the pipeline taken from [9]:

1. Camera Calibration
2. Image Feature Extraction
3. Image Feature Matching
4. Outlier Rejection
5. Motion Estimation
6. Scale Estimation
7. Optimization

Camera Calibration See Section 2.2.1.

Sparse Feature Extraction Feature-based VO relies on feature detectors and descriptors to extract and match points of interest between images. Feature extraction identifies interesting aspects of an image. The identified features are usually corners, blobs (large regions with similar color or brightness), or some other recognizable element [7]. Extensive research has been done to ensure that extracted feature keypoints (pixel locations) are useful for the VO process. Because features need to be matched between images to recognize and track objects, good features should be unique and identifiable regardless of perspective changes [7]. Some common feature extractors are SIFT [21], SURF [22], BRISK [23], Harris-Laplace [24], FAST [25], and MSER [26] [27]. Certain detectors work better in particular environments. For instance, in urban environments where corners are prevalent, a blob detector like SIFT may not work as well as a corner detector like FAST [7].

Feature Matching In order to match features, interest point descriptors mathematically represent the region around a feature keypoint. These descriptors are then compared and said to identify the same feature if the two descriptions are close enough. In this way, aspects of objects can be identified from multiple perspectives. There are two main types of feature descriptors: binary and non-binary. Some common feature descriptors are SIFT, SURF, BRISK, BRIEF [28] and FREAK [29]. Oriented FAST and Rotated BRIEF (ORB) [30] is a detector/descriptor method that uses improved versions of FAST and BRIEF together. Some descriptors are specifically designed to work better with blob or corner detectors as well [7]. In general, the best descriptor depends on the balance between accuracy and computational time. For instance, the SIFT descriptor takes a long time to compute, but produces some of the most stable feature descriptions for matching [7]. In addition to these traditional feature detectors and descriptors, Deep Learning (DL) has also been used to identify and describe feature keypoints [31, 32].

Common matching algorithms include brute-force, K-nearest neighbors, and FLANN matching. Feature descriptions are compared using mathematical distance measures. The higher the similarity between features, the lower the distance measure between their descriptions will be. Common distance measures include Euclidean distance (L2-Norm) and Manhattan distance (L1-Norm) for non-binary descriptors and Hamming distance for binary descriptors.

Outlier Rejection The ratio test [21] can reject some mismatched features using their descriptors. Relative locations of matched features can also be used to reject mismatches. Matched feature keypoints can be used collectively to determine image homography and estimate where a particular keypoint in one image should appear in the other. If the relative locations of a matched keypoint pair vary greatly from the majority, this pair can be rejected as an outlier. Random sample consensus (RANSAC) [33] is a method that uses image homography for robust outlier rejection.

Motion Estimation With traditional VO methods, there are three main ways that the camera's motion is determined: 3D-to-3D, 3D-to-2D, and 2D-to-2D. These are summarized in [6].

3D-to-3D This method uses triangulated feature points from two pairs of images, typically two stereo pairs. Each pair of stereo images is used to triangulate the location of feature points. The two sets of 3D feature point locations are then used to determine the camera pose transformation (rotation and translation) that minimizes the amount of error between one set of feature points and the transformed set of other feature points. This calculated transform is then used as the VO estimate. In this method, the relative translational scale is inherent in the pose calculation because 3D feature locations are used.

3D-to-2D This method can be used by either stereo or monocular systems. To use this method, feature matches between three different images are necessary. Two of the images, which can be from either a stereo image pair or a sequential monocular image pair, are used to triangulate the 3D location of feature points. These 3D feature locations are then reprojected onto the frame of the third image and compared with the actual feature keypoints in the third image. The transformation that minimizes the reprojection error (the difference between the reprojected pixel locations and the actual pixel locations) is used as the VO estimate. This method is also called perspective from n points (PnP) because it uses a given number (n) of points for the reprojection error calculation. According to [16], this method is more accurate than the 3D-to-3D method.

2D-to-2D This method calculates the essential matrix to determine the amount of rotation and the direction of translation for the camera. The essential matrix describes the geometric relationship between two images taken with a calibrated camera. Using the epipolar constraint, a feature that appears in one image perspective can be projected to another image perspective. The essential matrix is calculated using this epipolar constraint. The essential matrix can then be used to estimate the rotation and translation from one image to the next. The translation from this calculation is the normalized direction of change in position. The relative scale of the translation can then be calculated by triangulating 3D points that appear in both images. The 3D distance between a pair of points in one image is divided by the distance between the same points in the other image to determine the relative scale. To improve the relative scale estimate, multiple scale values can be computed for different pairs of points and the mean or median of these values can be taken.

Scale Estimation See Section 2.2.2.

Optimization Optimization methods are ways to overcome the drift errors in scale estimation. One of the ways to do this is with a bundle adjustment which minimizes reprojection error over a subset of the trajectory [6]. Another optimization method is Simultaneous Localization and Mapping (SLAM). SLAM methods store features of the environment in a map and adjusting the scale drift when a previously identified object or feature is recognized (known as a revisit or loop closure) [6].

2.2.3.2 Appearance-Based VO Methods

Appearance-based VO methods consider the actual intensity values of the images instead of extracting and matching feature keypoints [8].

Optical Flow According to [11], optical flow is a method of computing how far a pixel moves from one image to the next. Dense optical flow determines the magnitude and direction of movement for every single pixel in the images. Sparse optical flow only does this calculation for a subset of the pixels. Some optical flow methods return the average flow for certain regions in the images. These optical flow calculations can then be used to calculate the camera's motion between perspectives [8].

Traditional Optical Flow Calculations Many algorithms have been developed for computing optical flow in images. Some common ones are Horn and Schunck [34], DeepFlow [35], EpicFlow [36], and DeepMatching [37]. One of the difficulties when relying on optical flow images for VO is the high computation time needed to determine the pixel movements from normal image pairs. Because of this, Deep Learning (DL) methods have been developed to accurately and quickly estimate the optical flow values.

Optical Flow Estimation with CNNs Multiple papers have used DL for optical flow estimation [1, 38, 39, 40]. In FlowNet [1], the authors developed two Convolutional Neural Network (CNN) architectures. These architectures were designed to produce an optical flow image from a pair of input images. According to [38], FlowNet was a paradigm shift because up to this point no one had tried using a simple CNN to do optical flow estimation. Both models in FlowNet used multiple convolutional layers such that the image size decreased and the channel size increased as the network progressed. FlowNetSimple (FlowNetS) used a single path that took a stacked image pair as input. FlowNetCorr (FlowNetC) started with two separate paths in a siamese structure that each took one of the images in the pair. Toward the middle of FlowNetC, the two paths combined into a single path that led to the output. In both networks, the convolutional part that gradually reduced the image size was followed by an up-pooling section that increased the resolution of the optical flow image based on the learned motion representation. FlowNet 2.0 [38] improved upon the FlowNet designs using a different training schedule, a stacked architecture that included multiple layers of FlowNet models, and a small displacement network. This architecture was able to greatly decrease the estimation error while running only slightly slower than the original FlowNet models.

Direct Methods Direct VO methods rely on photometric consistency error between the current image and the previous image to calculate the camera's ego-motion [8]. Photometric consistency uses the correlation of pixel intensity values to determine the movement of subregions in the two images, and thus the overall movement of the camera. Because direct methods do not extract features from the images, they are able to take advantage of more available information. However, this comes with a larger computational cost.

2.2.4 Deep Learning Visual Odometry Methods

In addition to the traditional methods for computing frame-to-frame (F2F) movement, the power of Deep Learning (DL) has been applied to Visual Odometry (VO) as well.

2.2.4.1 Ego-motion with CNNs

Early on in DL research for VO estimation, the problem was framed as a classification instead of a regression [10, 41]. Konda and Memisevic [10] proposed one of the first DL methods for VO. They used two separate Convolutional Neural Networks (CNNs) to process a small sequence of stereo images and output discretized estimates of the velocity or the change in direction. The authors also tried to use a linear regression, but found that the results were better when they used classification. This may have been due to the relatively small size of the CNN used [11]. While discretizing the predictions helps to reduce the necessary size of the CNN, it severely limits the versatility and precision of the predictions.

In [42], the authors proposed a siamese CNN architecture that they used to estimate the 2D motion of a ground-based robot from depth images computed using a light detection and ranging (LIDAR) 3D point cloud. The network consisted of two streams of alternating convolutional and pooling layers that were combined using dense (fully-connected) layers before making the actual VO predictions. The authors tried many different variations on the architecture and the hyper-parameters.

Optical Flow Based In [43], three different CNN sub-architectures were compared. The first CNN learned VO using a dense optical flow image. The second CNN learned VO by considering each quadrant of the optical flow image individually. The third CNN, also known as P-CNN, used both of the other two CNNs and combined the feature vectors before making a VO prediction.

In Flowdometry [11], the authors used FlowNetS to predict optical flow, then fed that prediction to a separate FlowNet-based architecture that predicted the camera's motion in the stacked image pair. Because optical flow calculations can be computationally intensive, Flowdometry was an attempt to use deep learning to estimate optical flow more efficiently for the purpose of VO.

The DeMoN network [39] used a stacked CNN architecture to predict depth, camera ego-motion, optical flow, surface normals, and confidence matching. The stacked CNN used an iterative process to improve each of its predictions. This network took an image pair as input, but unlike VO networks, it did not require the images to be taken at a specific interval or from a particular perspective. It avoided this constraint because the depth was estimated in conjunction with the ego-motion. The main network building block consisted of two stacked encoder/decoder convolutional networks inspired by FlowNet [1]. The first network predicted optical flow based on the previous depth and camera motion predictions. The second network predicted the depth map and camera motion based on that optical flow prediction. These building blocks were configured with an initial prediction network block followed by an identical iterative network block that reprocessed the predictions three times.

In [44], the authors used a CNN based on FlowNet 2.0 to determine the optical flow between subsequent images then passed this data to another CNN to estimate the camera's motion between images. The network also learned an explainability mask so that objects or features that could throw off the VO prediction would not be included in the VO calculation.

ImageNet/ResNet Based The VLocNet [12] used a siamese CNN architecture for VO estimation. This architecture was based on a ResNet-50 architecture [45]. In addition to the odometry network, an additional single-path CNN was used to predict the global pose. The first part of the global pose network shared network parameter weights with one of the streams in the VO network. This allowed for multi-task learning of the camera's motion at both a local image pair level and a global trajectory level. This network predicted the full 6-DoF of the camera's ego-motion. The authors used a geometric consistency loss function to facilitate learning both the rotation and translation, which have different units and scales. The network tuned the loss function's parameters during the training process. These parameters weighted the rotation and translation loss components separately so that they had a similar scale. VLocNet++ [46] built on the VLocNet architecture by adding a semantic segmentation network. Because the semantic segmentation network shared some parameters with the global pose network, parts of the image could be deemed more useful or less useful for determining pose regression.

2.2.4.2 End-to-End VO with Recurrent Convolutional Neural Networks (RCNNs)

The authors in [9] developed an end-to-end architecture that took in a sequence of monocular images and used a combination of a CNN and a Recurrent Neural Network (RNN) to predict the absolute pose of the camera in the sequence. The CNN they used was a pre-trained FlowNetS [1] architecture without the up-pooling layers. The network then fed the result from the CNN to a Long Short-Term Memory (LSTM) RNN layer. This allowed the network to learn overall sequence (trajectory) information in addition to the individual image pair movement determined by the CNN. In addition to the pose estimation, the network also predicted estimation uncertainty terms using unsupervised learning. In [13], the authors developed a RCNN architecture based on VLocNet [12]. By adding LSTM layers in between the convolutional layers and the dense layers, the network could better account for overall trajectory information before predicting the camera's pose.

2.2.4.3 Other Sensor Data and DL

While most VO with DL papers only use image-based inputs, some have tried using inputs from other sensors as well [14, 20]. In VINet [14], the authors used a RCNN structure that included a CNN architecture based on a pre-trained version of FlowNetC [1]. In addition to image pairs that were fed to the CNN, the network also took in raw IMU data. Because the IMU data was not synchronized with the camera's frame rate, an LSTM layer was used to condense the IMU information into an inertial feature vector that was then concatenated with the visual feature vector from the CNN. This combined visual/inertial feature vector was fed into another LSTM RNN layer that produced a VO pose prediction for the image sequence. The authors in [20], built on VINet [14] by exploring a different way of fusing the visual and the inertial feature vectors. Instead of a simple concatenation, they attempted to train the network to learn a better feature vector combination method by inserting fusion layers into the architecture.

2.2.4.4 Alternative Structures

Beyond CNNs or RCNNs, other DL structures have also been used for VO. In [47], a Variational Auto-Encoder (VAE) was used to predict VO results. They found that the VAE could be retrained to predict VO for different camera optic models. In [48], multiple Generative Adversarial Network (GAN) layers were used to estimate both depth and ego-motion.

2.3 Coordinate Notation

2.3.1 Coordinate Points

Coordinates are defined in reference to a specific set of axes. In a typical 3D coordinate frame, each of the three axes are defined as orthogonal to each of the other two axes. Thus, a point **p** expressed in the 3D origin frame (0) as the distance from the origin to point **p** ($0 \rightarrow \mathbf{p}$) can be expressed as a vector $\mathbf{t}^0_{0\rightarrow\mathbf{p}}$ of three numbers (one for each axis x, y, z)

$$\mathbf{t}^0_{0\rightarrow\mathbf{p}} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}. \tag{1}$$

The distance from point **p** to the origin can be expressed as the negative vector.

$$\mathbf{t}^0_{\mathbf{p}\rightarrow 0} = -\mathbf{t}^0_{0\rightarrow\mathbf{p}} \tag{2}$$

2.3.2 Coordinate Frames

A coordinate frame is defined by a position and orientation relative to another frame. If two coordinate frames $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are located at the same position, but their relative orientations are different, a point in $\boldsymbol{\alpha}$ can be expressed in $\boldsymbol{\beta}$'s frame using a Direction Cosine Matrix (DCM) that transforms points from $\boldsymbol{\alpha}$ to $\boldsymbol{\beta}$. For 3D coordinate frames, a DCM is expressed as a 3×3 matrix ($\mathbf{R}_{\alpha}^{\beta} \in \mathbb{R}^3$) where each row identifies the weight of each original frame ($\boldsymbol{\alpha}$) axis value in the new frame ($\boldsymbol{\beta}$) axis.

In addition, a DCM is orthogonal, meaning that its transpose is the same as its inverse. Thus, given a DCM $\mathbf{R}_{\alpha}^{0}$ that transforms points from the $\boldsymbol{\alpha}$ frame to the origin frame, the DCM $\mathbf{R}_{0}^{\alpha}$ that transforms points in the opposite direction is,

$$\mathbf{R}_{0}^{\alpha} = (\mathbf{R}_{\alpha}^{0})^{-1} = (\mathbf{R}_{\alpha}^{0})^{T}. \tag{3}$$

The point $\mathbf{p}$, expressed in the origin frame ($\mathbf{t}_{0\to\mathbf{p}}^{0}$), can be expressed in the $\boldsymbol{\alpha}$ frame given the relative rotation and translation to the origin axis $\mathbf{R}_{0}^{\alpha}$ and $\mathbf{t}_{\alpha\to 0}^{\alpha}$, respectively.

$$\mathbf{t}_{\alpha\to\mathbf{p}}^{\alpha} = \mathbf{R}_{0}^{\alpha}\mathbf{t}_{0\to\mathbf{p}}^{0} + \mathbf{t}_{\alpha\to 0}^{\alpha} \tag{4}$$

$$= \mathbf{t}_{0\to\mathbf{p}}^{\alpha} + \mathbf{t}_{\alpha\to 0}^{\alpha} \tag{5}$$

2.3.3 Special Euclidean Group Matrices

Special Euclidean group (SE(3)) matrices are used to facilitate these coordinate frame transformations. A SE(3) matrix takes the rotation and translation that define a coordinate frame transformation and puts them together in a single square matrix with an added lower row. This allows the resulting matrix to be used to transform homogeneous coordinate points with a single matrix multiplication. For example, given the relative rotation and translation of the $\boldsymbol{\alpha}$ frame to the origin frame $\mathbf{R}_0^{\alpha}$ and $\mathbf{t}_{\alpha\to 0}^{\alpha}$, respectively, the SE(3) matrix $\mathbf{T}_0^{\alpha}$ that translates points from the origin frame to the $\boldsymbol{\alpha}$ frame is,

$$\mathbf{T}_0^{\alpha} = \begin{bmatrix} \mathbf{R}_0^{\alpha} & \mathbf{t}_{\alpha\to 0}^{\alpha} \\ \mathbf{0_{1\times 3}} & 1 \end{bmatrix}. \tag{6}$$

2.3.4 Coordinate Frame Conversion

The SE(3) matrix $\mathbf{T}_0^{\alpha}$ can be used to transform the point $\mathbf{p}$ to the $\boldsymbol{\alpha}$ frame using homogeneous coordinates $\underline{\mathbf{t}}_{0\to\mathbf{p}}^{0}$.

$$\underline{\mathbf{t}}_{0\to\mathbf{p}}^{0} = \begin{bmatrix} \mathbf{t}_{0\to\mathbf{p}}^{0} \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{7}$$

$$\begin{aligned}
\underline{\mathbf{t}}_{\alpha\to\mathbf{p}}^{\alpha} &= \mathbf{T}_0^{\alpha}\underline{\mathbf{t}}_{0\to\mathbf{p}}^{0} && (8)\\
&= \begin{bmatrix} \mathbf{R}_0^{\alpha} & \mathbf{t}_{\alpha\to 0}^{\alpha} \\ \mathbf{0_{1\times 3}} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{t}_{0\to\mathbf{p}}^{0} \\ 1 \end{bmatrix} && (9)\\
&= \begin{bmatrix} \mathbf{R}_0^{\alpha}\mathbf{t}_{0\to\mathbf{p}}^{0} + \mathbf{t}_{\alpha\to\mathbf{0}}^{\alpha} \\ 1 \end{bmatrix} && (10)\\
&= \begin{bmatrix} \mathbf{t}_{\alpha\to\mathbf{p}}^{\alpha} \\ 1 \end{bmatrix} && (11)
\end{aligned}$$

SE(3) matrices can also be multiplied to combine frame transformations. Given another SE(3) matrix $\mathbf{T}_{\alpha}^{\beta}$ that transforms points from the $\boldsymbol{\alpha}$ frame to the $\boldsymbol{\beta}$ frame,

$$\begin{aligned}
\mathbf{T}_0^{\beta} &= \mathbf{T}_{\alpha}^{\beta}\mathbf{T}_0^{\alpha} && (12)\\
&= \begin{bmatrix} \mathbf{R}_{\alpha}^{\beta} & \mathbf{t}_{\beta\to\alpha}^{\beta} \\ \mathbf{0_{1\times 3}} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{0}^{\alpha} & \mathbf{t}_{\alpha\to 0}^{\alpha} \\ \mathbf{0_{1\times 3}} & 1 \end{bmatrix} && (13)\\
&= \begin{bmatrix} \mathbf{R}_{\alpha}^{\beta}\mathbf{R}_{0}^{\alpha} & \mathbf{R}_{\alpha}^{\beta}\mathbf{t}_{\alpha\to 0}^{\alpha} + \mathbf{t}_{\beta\to\alpha}^{\beta} \\ \mathbf{0_{1\times 3}} & 1 \end{bmatrix} && (14)\\
&= \begin{bmatrix} \mathbf{R}_{0}^{\beta} & \mathbf{t}_{\beta\to 0}^{\beta} \\ \mathbf{0_{1\times 3}} & 1 \end{bmatrix} && (15)
\end{aligned}$$

In addition, the inverse of a SE(3) matrix can be used to transform coordinates in reverse. However, unlike DCMs the SE(3) matrix inverse is not the same as its transpose.

$$\mathbf{T}_{\beta}^{0} = (\mathbf{T}_0^{\beta})^{-1} \neq (\mathbf{T}_0^{\beta})^{T} \qquad (16)$$

2.3.5 Spherical Coordinates

Spherical coordinates can be calculated from Cartesian coordinates using Equations (17), (18) and (19) [49]:

$$\begin{aligned}
r &= \sqrt{x^2 + y^2 + z^2} && (17)\\
\theta &= \arccos\left(\frac{z}{r}\right) && (18)\\
\phi &= \text{arctan2}\left(\frac{y}{x}\right) && (19)
\end{aligned}$$

where r is the radius, θ is the inclination angle, and ϕ is the azimuth angle.

Cartesian coordinates can be calculated from spherical coordinates using Equations (20), (21) and (22) [49]:

$$x = r \sin\theta \cos\phi \tag{20}$$

$$y = r \sin\theta \sin\phi \tag{21}$$

$$z = r \cos\theta \tag{22}$$

2.4 Inertial Errors

2.4.1 Bias Error

Angular bias error is a constant angular offset ϵ_{bias} in the gyroscope measurements of an IMU. This is the average output of the gyroscope when it is not undergoing any rotation [50]. Common units for angular bias error are degrees per hour ($°/hr$). Constant gyro bias errors cause an angular error θ which grows linearly with time when integrated.

$$\theta(t) = \epsilon_{bias} \cdot t \tag{23}$$

2.4.2 Angular Random Walk Error

Angular random walk (ARW) or Thermo-Mechanical White Noise is the error response due to white noise in the gyro measurements [50]. White noise with a particular standard deviation σ_{arw} causes the standard deviation of the angular error σ_θ to grow proportional to the square root of time.

$$\sigma_\theta(t) = \sigma_{arw} \cdot \sqrt{\delta t \cdot t} \tag{24}$$

where δt is the time in between samples.

Common units for the ARW term are degrees per square root hour ($°/\sqrt{hr}$). The standard deviation (strength) of the white noise causing the ARW error can be calculated using the random walk noise term ϵ_{arw} and the sampling rate f_{samp}.

$$\sigma_{arw} = \frac{\epsilon_{arw}}{\sqrt{f_{samp}}} \tag{25}$$

2.5 Summary

This chapter explained the necessary background and literature review information to understand this thesis. This included a review of research in VO, an overview of coordinate frame transformations, and an explanation of INS errors. The next chapter will explain the experimental setup and procedures.

III. Methodology

3.1 Overview

The main task of this thesis is to analyze the effect of additional input information given to a Convolutional Neural Network (CNN) estimating frame-to-frame (F2F) Visual Odometry (VO). The KITTI Odometry dataset [2] is used for training and evaluation. It includes undistorted and rectified image frame sequences taken from a vehicle driving along roadways. These image frames are labeled with the true pose (orientation and position) of the vehicle. A detailed explanation of the dataset and pre-processing is given in Section 3.2.

For each of three VO output types, two Deep Neural Networks (DNNs) are considered: a baseline model and an Inertial Navigation System (INS)-aided model. Each of the six models use the same CNN portion. The baseline models only take in image pair inputs. The INS-aided models also have a dense (fully-connected) layer branch that takes in rotation estimate inputs. The only differences between the three models within each type is the number of VO outputs. Section 3.3 describes the neural network architectures used in this thesis. The training parameters used in the networks are given in Section 3.4. Finally, Section 3.5 outlines the procedure for training and evaluation.

3.2 KITTI Odometry Dataset and Pre-Processing

3.2.1 Dataset Statistics

The KITTI Odometry dataset [2] includes 22 sequences of images. Four cameras (two gray-scale, two color) were fixed to the top of a small car that was driven in both urban and highway environments. The cameras were mounted such that the pair of gray-scale cameras were side by side approximately 54 cm apart and facing

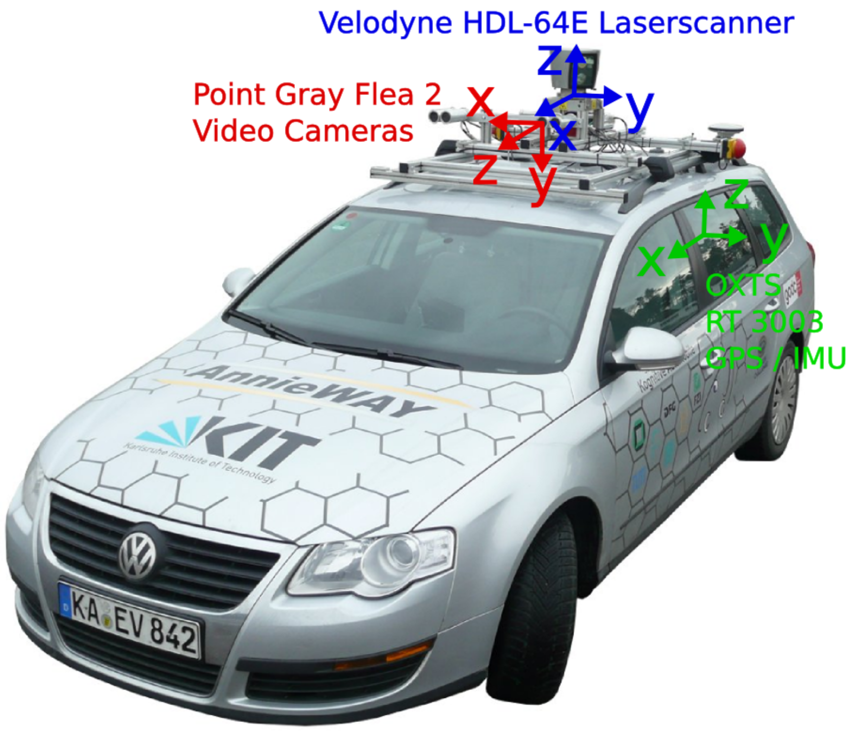


Figure 2: Image taken from [4] of the vehicle setup used for the collection of the KITTI Odometry dataset [2]. The diagram shows the locations of the GPS/IMU sensors, four cameras, and Velodyne LIDAR. The left gray-scale camera axis (camera 0) is shown in red.

the front of the vehicle. Thus, the KITTI dataset could also be used with stereo VO techniques, but this thesis only uses images from a single gray-scale camera. The setup also included a Velodyne light detection and ranging (LIDAR) scanner. A GPS receiver and an Inertial Measurement Unit (IMU) mounted to the vehicle were used to determine the actual pose of the car. The vehicle setup for the KITTI Odometry dataset is shown in Figure 2. The dataset also provides the camera calibration matrix for each sequence.

Of the 22 data sequences, only the first 11 include the associated truth pose of the vehicle. The truth poses of the other 11 sequences are not provided because they are used to benchmark methods submitted to the KITTI Odometry administrators. The number of image frames for the first 11 sequences are shown in Table 1. Of these 11,

Table 1: Number of frames per camera for the first 11 sequences in the KITTI Odometry dataset [2].

Sequence	**Number of Frames**
00	4541
01	1101
02	4661
03	801
04	271
05	2761
06	1101
07	1101
08	4071
09	1591
10	1201

the first ten sequences (00-09) were used for training and evaluation in this thesis. The 11^{th} sequence (10), which has 1201 frames, was used for full trajectory evaluation and visualization. Since images were only used in pairs of sequential frames, the number of observations in each sequence is one less than the number of frames in that sequence. In addition, only images from the left gray-scale camera (camera 0) were used. Thus, there were a total of 21,990 observations. Each observation included an image pair, a truth label defining the amount of rotation and/or translation between the two images, and a simulated INS rotation estimate.

3.2.2 Image Pairs

3.2.2.1 Sequence Characteristics

The KITTI Odometry dataset consists of image sequences taken in urban, suburban, and highway environments. A representative sample of the dataset images is shown in Figure 3. The vehicle was driving forward the majority of the time, but many sequences include significant turns as well. While there are slight changes in the roll and pitch of the vehicle, the vast majority of the rotation changes in the dataset are in the yaw (heading). Similarly, the majority of the translational movement is in the forward direction.

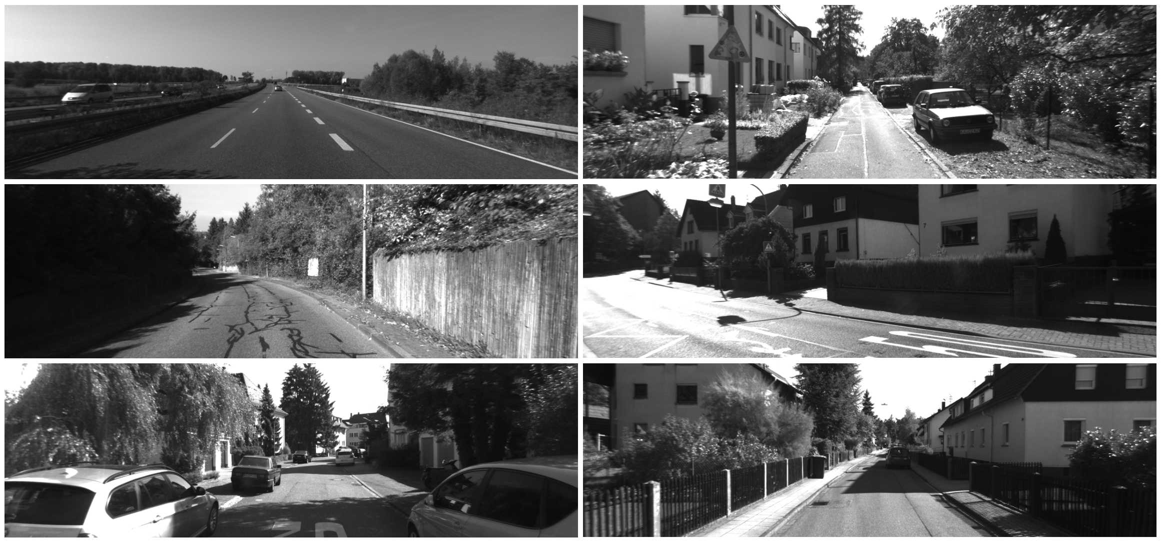


Figure 3: Respresentative sample of the images taken by Camera 0 in the KITTI Odometry dataset [2].

3.2.2.2 KITTI Format

The images in KITTI Odometry dataset were collected at a 10 Hz framerate. Each image is already undistorted and rectified. The pixel intensity values range from 0 to 255. The images given in the dataset are cropped versions of the raw collected images. The image sizes are consistent within each individual sequence, but they vary slightly from sequence to sequence. The image widths range from 1226 pixels to 1242 pixels. The image heights range from 370 pixels to 376 pixels. This is most likely due to the image rectification.

3.2.2.3 Image Standardization

The images needed to be downsampled to reduce the number of calculations for the neural network. However, because the image sizes between sequences were inconsistent due to the rectification process, downsampling could cause the principle point to be inconsistent from sequence to sequence. This could affect the VO estimation because the geometric relationships between pixels would be inconsistent. Thus, the image sizes needed to be standardized first.

To get a consistent size, the image size for each sequence was converted to normalized image coordinates using the sequence's camera calibration matrix. This allowed the field of view for each sequence to be compared with the other sequences using a consistent principle point. Equation (26) was used to convert a pixel coordinate $\mathbf{p} = [x_p \; y_p]^T$ to a normalized coordinate $\mathbf{n} = [x_n \; y_n]^T$.

$$\underline{\mathbf{n}} = \mathbf{K}^{-1}\underline{\mathbf{p}} \tag{26}$$

where $\mathbf{K}$ is the 3×3 camera calibration matrix and $\underline{\mathbf{n}} = [x_n \; y_n \; 1]^T$ and $\underline{\mathbf{p}} = [x_p \; y_p \; 1]^T$ are the homogeneous normalized and pixel image coordinates, respectively.

For each image size, the corner pixel values were converted to normalized image coordinates. These four corners created a rectangular region that represented the field of view for that sequence. The field of view rectangles were plotted on the same graph, and the middle region, where the rectangles overlapped, was chosen as the standard image rectangle.

The standard rectangle corners were then converted back to pixel coordinates for each sequence using Equation (27). These pixel coordinates were then used to map the original image pixels to the standardized image size. The standardized images had a height and width of 370×1226 pixels. This size was selected using the minimum width and height across all of the sequences.

$$\underline{\mathbf{p}} = \mathbf{K}\underline{\mathbf{n}} \tag{27}$$

3.2.2.4 Image Downsampling

The standardized images were then downsampled using the Open-Source Computer Vision (OpenCV) library [51] `resize` function with area interpolation. When downsampling, area interpolation calculates the pixel values for the new image using the weighted average of the corresponding region of pixels in the original image.

Two sizes of images were considered. The large size had a height and width of 320×1216 pixels. This was selected by taking the closest multiple of 64, as in [9], that was less than the standard image size of 370×1226. The small size with height and width of 160×608 pixels was selected by halving each dimension of the large image size.

3.2.2.5 Pixel Normalization

The pixels values in each image were normalized by compressing the original integer range from 0 to 255 into a decimal range from 0 to 1 [52, pp. 101-102]. This was done to aid in the training process of the neural network models because the pixel distribution would not also have to be learned [53, 54].

3.2.2.6 Image Pair Composition

The image pair tensor was composed by taking two sequential single-channel grayscale image frames from a sequence and stacking them in the channels dimension. Thus, for the smaller image size, the final tensor shape was $(160 \times 608 \times 2)$.

3.2.3 Truth Labels

3.2.3.1 KITTI Format

The truth poses in the KITTI Odometry dataset were calculated using measurements from a GPS/IMU system. They are given in a text file for each sequence. Each row corresponds to an image frame in the sequence. The 12 values in each row correspond to the top 12 values in a special Euclidean group (SE(3)) matrix (from left to right and top to bottom). The SE(3) matrix $\mathbf{T}_i^0$ in row i of a particular sequence's text file will transform a position from the i^{th} image's frame to first (0^{th}) image's coordinate frame (i.e. the coordinate frame at the sequence's starting position). This SE(3) matrix is shown in Equation (28).

$$\mathbf{T}_i^0 = \begin{bmatrix} \mathbf{R}_i^0 & \mathbf{t}_{0\to i}^0 \\ \mathbf{0_{1\times 3}} & 1 \end{bmatrix} \tag{28}$$

3.2.3.2 Vehicle/Camera Coordinate Frame

All coordinate frames are defined in camera coordinates. A camera coordinate frame is located at the camera's position with the z axis pointing out of the camera directly toward the image principle point, the y axis pointing down toward the bottom of the image perpendicular to the z axis, and the x axis pointing toward the right side of the camera perpendicular to both the y and z axes. This is illustrated in Figure 1 and in the red coordinate frame in Figure 2. For the KITTI Odometry dataset, the truth poses given are the pose of the left gray-scale camera (camera 0).

3.2.3.3 Convert from Absolute SE(3) to Relative Rotation, Translation, or Scale

Equation (29) was used to convert the given absolute frame transformations into relative frame transformations. The SE(3) matrix $\mathbf{T}_i^{i+1}$ transforms coordinates from the current frame (i) to the next frame ($i + 1$).

$$\begin{aligned}\mathbf{T}_i^{i+1} &= (\mathbf{T}_{i+1}^0)^{-1}\mathbf{T}_i^0 &(29)\\ &= \mathbf{T}_0^{i+1}\mathbf{T}_i^0 &(30)\\ &= \begin{bmatrix}\mathbf{R}_i^{i+1} & \mathbf{t}_{i+1\to i}^{i+1}\\ \mathbf{0_{1\times 3}} & 1\end{bmatrix} &(31)\end{aligned}$$

Rotation To use the rotation $\mathbf{R}_i^{i+1}$ as a truth label, it was converted from a Direction Cosine Matrix (DCM) to a rotation vector representation using the OpenCV [51] `Rodrigues` function [3]. The Rodrigues formula transforms a 3×3 DCM into a 3×1 rotation vector. A rotation vector is a combination of an axis-angle representation of rotation. Axis-angle is expressed as an amount of rotation (angle) about a particular unit vector (axis). The rotation vector is calculated by multiplying the angle value and the unit vector axis. Thus, the rotation vector's magnitude is the same as the angle value, and its direction is the same as the axis vector. Because of the nature of the `Rodrigues` function in OpenCV, the inverse rotation was required in order to produce the desired result. This is shown in Equation (32).

$$\boldsymbol{\phi}_i^{i+1} = \texttt{Rodrigues}((\mathbf{R}_i^{i+1})^{-1}) = \begin{bmatrix}\phi_x\\ \phi_y\\ \phi_z\end{bmatrix} \tag{32}$$

where ϕ_x, ϕ_y, and ϕ_z represent the respective axis components of the camera's orientation change in radians.

Translation The Cartesian translation $\mathbf{t}_{i+1\to i}^{i+1}$ was used directly in the truth label. This is shown in Equation (33).

$$\mathbf{t}^{i+1}_{i+1\to i} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{33}$$

where x, y, and z represent the camera's translation along each axis.

Scale The scale of the translation was determined by calculating the vector magnitude of each Cartesian translation vector. The translation vector magnitude is equivalent to the radius r value, as calculated in Equation (17), when the translation is expressed in spherical coordinates (Section 2.3.5).

3.2.3.4 Truth Label Composition

There were three different neural network output types used in this thesis. With the full six degrees of freedom (6-DoF) output, the truth label consisted of the three rotation vector values $\boldsymbol{\phi}^{i+1}_{i}$ calculated using Rodrigues [3] followed by the three Cartesian translation $\mathbf{t}^{i+1}_{i+1\to i}$ values. This is shown in Equation (34).

$$\boldsymbol{\gamma}_{\text{6DoF}} = \begin{bmatrix} \phi_x & \phi_y & \phi_z & x & y & z \end{bmatrix} \tag{34}$$

With the Cartesian translation output, the truth label consisted of the three values in the translation vector $\mathbf{t}^{i+1}_{i+1\to i}$. This is shown in Equation (35).

$$\boldsymbol{\gamma}_{\text{trans}} = \begin{bmatrix} x & y & z \end{bmatrix} \tag{35}$$

With the scale output, the truth label consisted of only the translation magnitude r, as calculated in Equation (17). This is shown in Equation (36).

$$\boldsymbol{\gamma}_{\text{scale}} = \begin{bmatrix} r \end{bmatrix} \tag{36}$$

The truth data statistics are summarized in Table 2 for sequences 00-09 and Table 3 for sequence 10.

3.2.3.5 Turns

Through visual inspection of the images, it was determined that a y rotation component value greater than 1 degree produced a noticeable rotation change in the images. In sequences 00-09 of the dataset, 4051 of the 21990 image pairs (18.4%) have y rotation changes of greater than 1 degree. For sequence 10 of the dataset, 165

Table 2: Truth data statistics for KITTI Odometry [2] sequences 00-09. This includes the mean, standard deviation, minimum and maximum values, and normalized minimum and maximum values for each output type. The rotation values presented are the individual axis components of a rotation vector. The normalization of the data is explained in Section 3.2.3.6.

Output Type	**Mean**	**Std Dev**	**Real Min**	**Real Max**	**Norm Min**	**Norm Max**
Rot X (deg/1000)	-3.605	170.472	-1291.390	1324.998	-7.547	7.804
Rot Y (deg/1000)	16.669	996.844	-4775.337	4257.612	-4.826	4.274
Rot Z (deg/1000)	0.272	149.822	-1071.533	1146.135	-7.227	7.727
Trans X (mm)	1.123	17.808	-239.388	299.083	-13.848	17.155
Trans Y (mm)	17.828	17.754	-196.512	199.258	-12.105	10.262
Trans Z (mm)	-965.973	438.729	-2738.502	17.445	-4.064	2.248
Trans Scale (mm)	966.810	437.969	0.542	2738.905	-2.212	4.070

Table 3: Truth data statistics for KITTI Odometry [2] sequence 10. This includes the mean, standard deviation, minimum and maximum values, and normalized minimum and maximum values for each output type. The rotation values presented are the individual axis components of a rotation vector. The normalization of the data is explained in Section 3.2.3.6.

Output Type	**Mean**	**Std Dev**	**Real Min**	**Real Max**	**Norm Min**	**Norm Max**
Rot X (deg/1000)	7.967	212.244	-936.705	1501.438	-5.466	8.839
Rot Y (deg/1000)	184.032	848.388	-1511.065	3897.035	-1.538	3.911
Rot Z (deg/1000)	-0.605	166.840	-843.556	533.731	-5.690	3.598
Trans X (mm)	3.325	15.667	-127.401	66.611	-7.400	3.770
Trans Y (mm)	14.555	11.367	-21.764	103.889	-2.229	4.872
Trans Z (mm)	-765.799	334.646	-1525.177	-11.567	-1.285	2.181
Trans Scale (mm)	766.265	334.471	12.147	1525.580	-2.186	1.286

of the 1200 image pairs (13.8%) are turning observations.

3.2.3.6 Normalization

The goal of the output normalization was to scale the data such that each component had a mean of 0 and a standard deviation of 1 [53, 54]. Because the evaluation data could not be used for this normalization calculation, only values from the training set were used. Thus, the normalization parameters were the training set mean and standard deviation of each output feature.

For example, the Cartesian translation output has three features: x, y, and z. Thus, the normalization parameters included three mean ($\boldsymbol{\pi}$) and three standard deviation ($\boldsymbol{\sigma}$) values. This is shown in Equations (37) and (38).

$$\boldsymbol{\pi}_{\text{trans}} = \begin{bmatrix} \pi_x & \pi_y & \pi_z \end{bmatrix} \tag{37}$$

$$\boldsymbol{\sigma}_{\text{trans}} = \begin{bmatrix} \sigma_x & \sigma_y & \sigma_z \end{bmatrix} \tag{38}$$

where $\boldsymbol{\pi}_{\text{trans}}$ and $\boldsymbol{\sigma}_{\text{trans}}$ are the training mean and standard deviation vectors, respectively.

The normalization parameters for both the truth and INS data are shown in Table 4. These normalization parameters were applied to the truth labels using Equation (39) and undone using Equation (40).

$$\boldsymbol{\gamma}_{norm} = \frac{\boldsymbol{\gamma}_{orig} - \boldsymbol{\pi}}{\boldsymbol{\sigma}} \tag{39}$$

$$\boldsymbol{\gamma}_{orig} = (\boldsymbol{\gamma}_{norm} \odot \boldsymbol{\sigma}) + \boldsymbol{\pi} \tag{40}$$

Table 4: Geometric data normalization parameters calculated by taking the mean and standard deviation of all training set values for each output feature. This allows for easier training because the normalized input and output data has an approximate a mean of 0 and a standard deviation of 1. The rotation values presented are the individual axis components of a rotation vector.

Output Type	**Mean**	**Std Dev**
INS Rot X (deg/1000)	-5.138	170.478
INS Rot Y (deg/1000)	15.174	992.711
INS Rot Z (deg/1000)	0.198	148.333
True Rot X (deg/1000)	-5.106	170.441
True Rot Y (deg/1000)	15.161	992.657
True Rot Z (deg/1000)	0.209	148.302
True Trans X (mm)	1.131	17.368
True Trans Y (mm)	17.681	17.695
True Trans Z (mm)	-964.118	436.659
True Trans Scale (mm)	964.942	435.901

where $\boldsymbol{\gamma}_{norm}$ is the normalized truth label, $\boldsymbol{\gamma}_{orig}$ is the original truth label, $\boldsymbol{\pi}$ is the training mean vector, $\boldsymbol{\sigma}$ is the training standard deviation vector, and $\odot$ is an element-wise multiplication.

3.2.4 Simulated INS Rotations

The KITTI dataset does include raw IMU measurements, but they are not time synchronized with the images. Thus, to reduce complexity, simulated INS rotations were created by adding Gaussian errors to the true rotations. First, a typical angular random walk (ARW) error ($\epsilon_{arw} = 0.5\ deg/\sqrt{hr}$) was taken from the NovAtel HG1700 IMU datasheet [55]. This, along with the KITTI sampling rate of $f_{samp} = 10\ Hz$ [2], was used to determine the strength (σ_{arw}) of Gaussian noise needed using Equation (25) in Section 2.4.

$$\sigma_{arw} = \frac{\epsilon_{arw}}{\sqrt{3600 \cdot f_{samp}}} \approx 0.002635 \text{ deg} \tag{41}$$

To create the simulated INS rotations, this strength value was added to each of the truth rotation vectors ($\boldsymbol{\phi}_i^{i+1}$) according to Equation (42).

$$\text{INS}_i^{i+1} = \boldsymbol{\phi}_i^{i+1} + \frac{\pi}{180}\sigma_{arw}\mathbf{n} \tag{42}$$

where $\boldsymbol{\phi}_i^{i+1}$ is the true rotation vector from frame i to frame $i+1$, σ_{arw} is the ARW strength computed in Equation (41), and $\mathbf{n}$ is a 3×1 sample of white Gaussian noise.

These INS values were normalized in the same way as the truth labels, as described in Section 3.2.3.6. The normalization parameters for the INS data are shown in Table 4.

The statistics for the INS rotations are very similar to the truth rotation statistics shown in Tables 2 and 3. The RMSE values for the INS data compared to the truth data for various subsets of the KITTI Odometry dataset are shown in Table 5. These RMSE values can be used to determine if the INS-aided neural network models predict more accurate rotation values than the input INS rotations they are given.

Table 5: INS data Root Mean Squared Error (RMSE) values for different subsets of the KITTI Odometry dataset [2]. The rotation values presented are the individual axis components of a rotation vector.

Output Type	**Train Set**	**Val Set**	**Test Set**	**Seqs 00-09**	**Seq 10**
Rot X RMSE (deg/1000)	2.644	2.621	2.633	2.638	2.700
Rot Y RMSE (deg/1000)	2.646	2.625	2.682	2.647	2.641
Rot Z RMSE (deg/1000)	2.645	2.650	2.648	2.647	2.636

3.3 Model Architectures

The main machine learning task for this thesis is to use supervised learning of input images and rotation estimates to regress a camera's movement (whether full 6-DoF, just Cartesian translation, or translational scale). CNNs were used to account for the spatial nature of the images. The CNN design was taken from [1, 9].

Recurrent Convolutional Neural Networks (RCNNs) have been demonstrated to produce good results for VO because they can consider global trajectory information [9, 13]. However, it is not yet clear how to use RCNNs with other sensor fusion techniques like Kalman filters. Thus, this thesis focuses on CNN-based techniques that consider one image pair at a time.

3.3.1 CNN Architectures

Two different CNN architectures were considered along with two image sizes for each. Both of these architectures were inspired by the convolutional portion of the FlowNetSimple CNN [1] as used in [9]. Both have the same structure, and the only difference is the channel size for each layer. The models were implemented using TensorFlow 1.14.0 [56] in the Keras 2.3.1 framework [57, 52].

3.3.1.1 FlowNet

The FlowNet CNN architecture is based on the convolutional portion of FlowNet-Simple [1]. It uses nine convolutional layers each followed by a Rectified Linear Unit (ReLU) activation, except the last. The kernel size starts at 7×7 and gradually decreases to 3×3. Zero padding is used so that the image size in the current layer is not reduced unless a stride operation is used [9]. Certain layers include a stride 2 which reduces the image size by half in each dimension. The first layer has a channel size of 64. Every time a stride 2 is used, the channel size in the next layer is doubled. Oth-

Table 6: FlowNet [1] CNN architecture for a small input image size of $(160 \times 608 \times 2)$

Layer	**Activation**	**Kernel Size**	**Zero Padding**	**Stride**	**Image Size**	**Number of Channels**
Input					160×608	2
Conv1	ReLU	7×7	3	2	80×304	64
Conv2	ReLU	5×5	2	2	40×152	128
Conv3	ReLU	5×5	2	2	20×76	256
Conv3_1	ReLU	3×3	1	1	20×76	256
Conv4	ReLU	3×3	1	2	10×38	512
Conv4_1	ReLU	3×3	1	1	10×38	512
Conv5	ReLU	3×3	1	2	5×19	512
Conv5_1	ReLU	3×3	1	1	5×19	512
Conv6	Linear	3×3	1	2	3×10	1024
Dense1	ReLU				3×10	128

erwise, the channel size stays the same. The nine convolutional layers are followed by a dense (fully-connected) layer with 128 channels and a ReLU activation. The total number of trainable parameters for the FlowNet model is 14,731,200. With the larger images, the final dense layer has a shape of $(5 \times 19 \times 128)$. With the small images, the final dense layer has a shape of $(3 \times 10 \times 128)$. This information is summarized in Table 6.

3.3.1.2 FlowNet-Half

The FlowNet-Half architecture is exactly the same as the FlowNet architecture given in Table 6 except with half the number of channels for each layer, including the dense layer. The total number of trainable parameters for the FlowNet-Half model is 3,685,344.

3.3.2 VO Model Architectures

Six different VO architectures are compared in this thesis—two for each of the three output types. Other than the number of outputs and an extra dense (fully-connected) layer in the INS aiding case, the two different types of architectures do

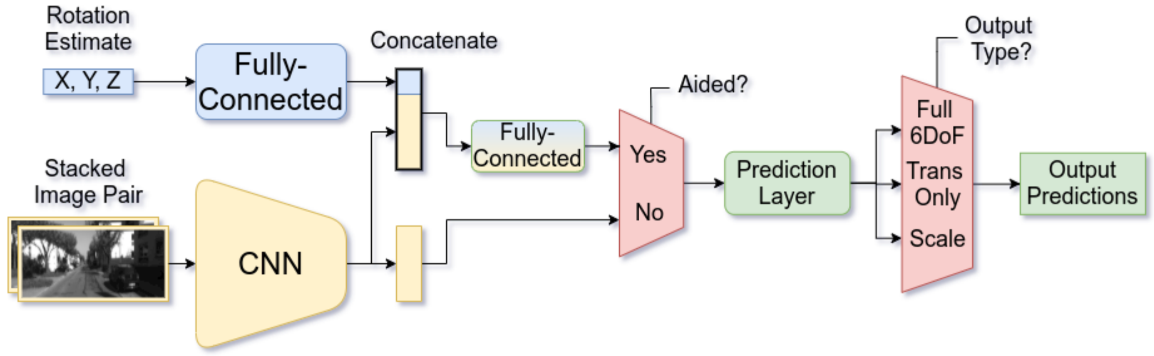


Figure 4: Abstract design and flow of data for all VO networks explored in this thesis. Decisions that separate cases (whether INS aiding is used and the type of output) are shown with red trapezoids.

Table 7: Characteristics of the six VO models based on decisions shown in Figure 4

Decisions		**Model Characteristics**		
Aided?	**Output Type?**	**Name**	**Inputs**	**Outputs**
No	Full 6DoF	6DoF-Baseline	Image Pairs Only	3D Rotation & 3D Translation
	Trans Only	Trans-Baseline		3D Cartesian Translation
	Scale	Scale-Baseline		Magnitude of Translation
Yes	Full 6DoF	6DoF-INS-Aided	Image Pairs & INS Rotations	3D Rotation & 3D Translation
	Trans Only	Trans-INS-Aided		3D Cartesian Translation
	Scale	Scale-INS-Aided		Magnitude of Translation

not differ. The abstract overall design of the networks used in this thesis is shown in Figure 4.

3.3.2.1 Baseline Architecture

The baseline architecture only takes in image pair input. The image pairs are fed into the CNN architecture. The output from the CNN is put through a global average pooling layer that condenses the information in each channel into a single dimension. This result is then fed into a dense layer that performs linear regression for each of the outputs. Thus, the size of this layer is the same as the number of outputs. An example of this architecture for the full 6-DoF output with the small image size is shown in Figure 5.

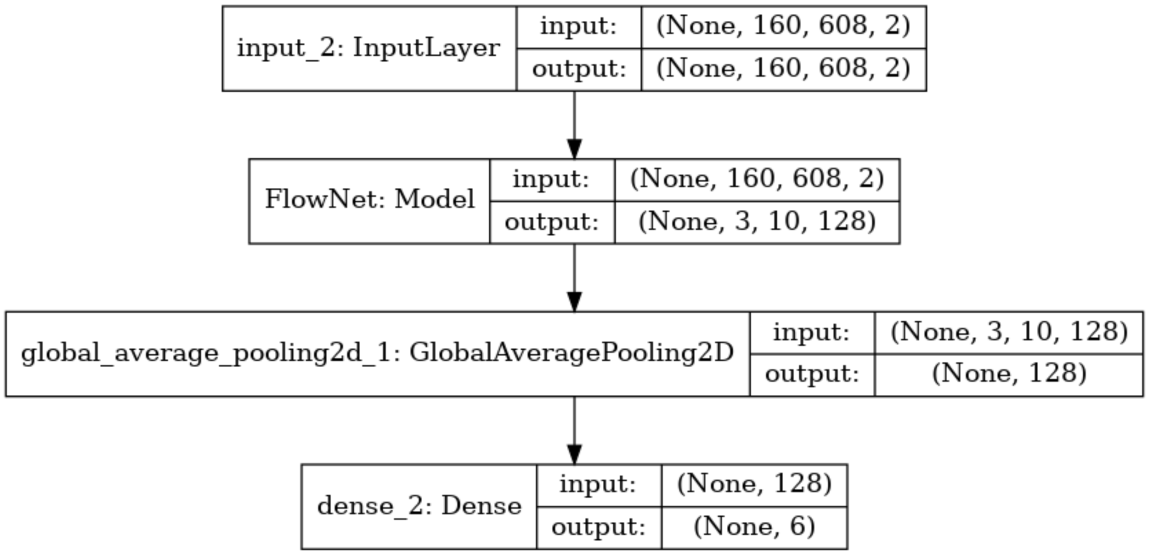


Figure 5: Baseline model architecture diagram for full 6-DoF VO prediction.

With the FlowNet CNN and small image input, the total number of trainable parameters for the baseline networks is 14,731,974 for the full 6-DoF model, 14,731,587 for the Cartesian translation model, and 14,731,329 for the translational scale model.

3.3.2.2 INS-Aided Architecture

Like the baseline architecture, the INS-aided architecture also takes in the image pairs and feeds them to the CNN architecture. In addition to this, it takes in the three rotation vector components of the rotation estimate and feeds them to a dense layer with a size of 16 and a ReLU activation. The output of this layer is then concatenated with the global average pooled output from the CNN and passed through another dense layer with 128 channels. This extra dense layer is added to allow the network to determine the fusion of the CNN vector and the INS-aiding vector. The result of this dense layer is then passed to the regression layer that estimates the output values. An example of this architecture for the full 6-DoF output with the small image size is shown in Figure 6.

With the FlowNet CNN and small image input, the total number of trainable

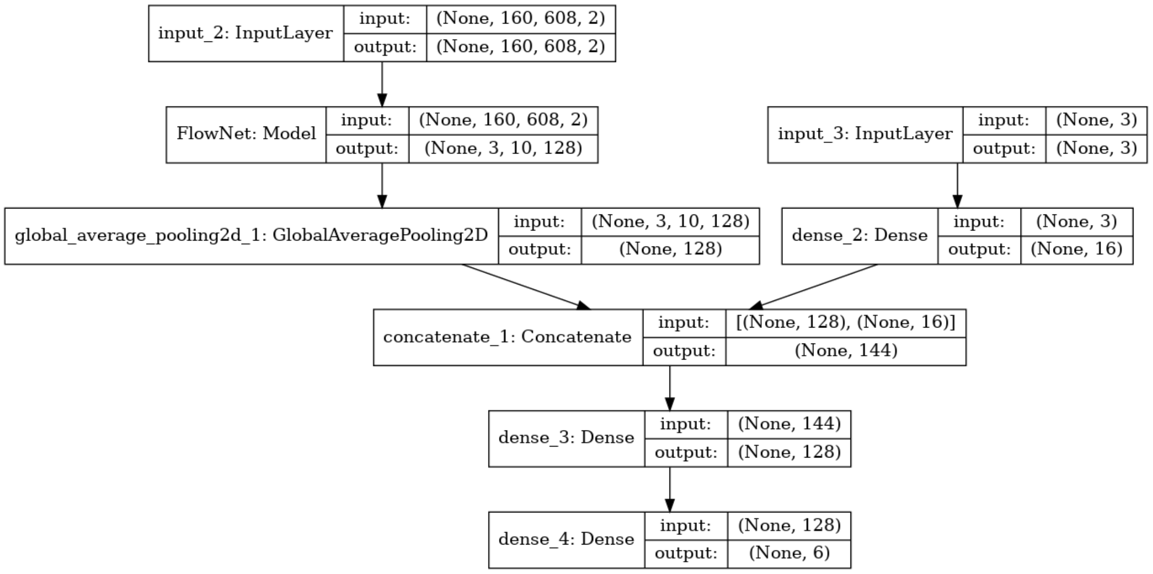


Figure 6: INS-aided model architecture diagram for full 6-DoF VO prediction.

parameters for the aided networks is 14,750,598 for the full 6-DoF model, 14,750,211 for the Cartesian translation model, and 14,749,953 for the scale model.

3.4 Training

3.4.1 Training-Validation-Test Split

To facilitate proper training and evaluation procedures, the 21,990 observations were randomly divided into a test and non-test set. The non-test set was then randomly divided into a training set and validation set. Of the total observations, 3,298 (15%) were used as the test set, 4,398 (20%) were used as the validation set, and the remaining 14,294 (65%) were used as the training set. The training set was used to fit each neural network model to the data, the validation set was used for model selection and hyper-parameter tuning, and the test set was used for final model comparison and evaluation.

3.4.2 Training Parameters

Loss Function The loss function used was Mean Squared Error (MSE). This increasingly penalizes incorrect values the further away they get from the truth. The scale and Cartesian translation models simply used the MSE as the loss function. In the 6-DoF models, the loss function calculated the translation and rotation MSEs separately because of the difference in units. Some have found that scaling the two MSE values to have a similar magnitude can produce good results [58, 12]. Since in this thesis, the MSE values were already scaled because of the normalization of the inputs and outputs (Section 3.2.3.6), the two MSE values were simply added together.

Optimizer The optimizer used for all training was the RMSProp [59] optimizer implemented in the Keras framework [57]. RMSProp is an implementation of stochastic gradient descent that also includes a variable learning rate.

Learning Rate The initial learning rate for each model was 0.0001. In addition, the learning rate was reduced by a factor of 0.1 after the training loss did not change more than 0.0001 for 10 epochs.

Initialization All of the model parameters were initialized to the default values in the Keras framework [57]. The convolutional and dense (fully-connected) layer kernels were initialized with Glorot Uniform [60] data. The convolutional and dense layer biases were initialized with zeros.

3.5 Procedure

The procedure followed in this thesis began by pre-processing the KITTI Odometry dataset [2] to obtain image pairs and truth labels as described in Sections 3.2.2 and 3.2.3. The INS rotations were then simulated by adding ARW errors to the true rotations as described in Section 3.2.4. In the final pre-processing step, the image pairs, truth labels, and INS rotations were normalized according to the procedures discussed in Sections 3.2.2.5 and 3.2.3.6. These dataset observations were divided into a training, validation, and test set according to Section 3.4.1.

After the data was prepared, the CNN architecture that would be used in each of the VO prediction models needed to be selected. Each VO model used the same CNN so that their performance could be compared. Four CNN architectures were examined: two architecture sizes and two input image sizes. These are explained in detail in Section 3.3.1. The four models were trained to fit the training set observations. For each of the four model types, the training epoch with the lowest validation loss was used for CNN selection. Using these CNN selection epochs, the CNN architecture with the lowest validation RMSE was used in the subsequent VO models. The CNN selection process and results are shown in Section 4.2.

Six VO models were used to analyze the performance effect of INS aiding. The differences are explained in Section 3.3.2 and summarized in Figure 4 and Table 7. These models were trained to fit the training set. Similar to the CNN selection, the best epoch was selected for each VO model type based on the lowest validation loss. These best epochs were used in each of the evaluation comparison methods. The evaluations in Sections 4.3.2 and 4.5 used the test set observations. The evaluations in Section 4.4 used the sequence 10 observations.

3.6 Summary

This chapter covered the experimental setup, KITTI Odometry dataset characteristics, and neural network model designs. It also explained the training scheme, dataset split, and model hyper-parameters. The next chapter will explain the model evaluation methods and show the training and evaluation results.

IV. Results and Analysis

4.1 Overview

This chapter begins by explaining how the Convolutional Neural Network (CNN) used in the Visual Odometry (VO) models was selected based on training and validation set results (Section 4.2). The training of the six VO model types is then detailed along with the selection of the best model epochs for evaluation (Section 4.3.1). The best epochs are then compared using results from the test set observations (Section 4.3.2). The overall test set errors are analyzed using Root Mean Squared Error (RMSE), mean and standard deviation, histograms, and error value plots. The error values on turning observations are also considered and compared with non-turning observations. A full trajectory (sequence 10) is then used to visualize the results of integrating the predictions from each model (Section 4.4). The RMSE values are also analyzed for sequence 10. Finally, the sensitivity of the Inertial Navigation System (INS)-aided networks to INS quality is shown (Section 4.5).

4.2 CNN Selection and Hyper-parameter Tuning

Four models were trained and analyzed to determine the appropriate CNN size for the VO models. Two CNN types were considered: the full FlowNet [1] model and a modified version of FlowNet with half the number of channels (FlowNet-Half). Each CNN type was tested with large images (320×1216) and small images (160×608). These differences are described in Sections 3.2.2.4 and 3.3.1. No INS aiding was used for these tests, so each model only used a stacked image pair as input. The models were trained using the RMSProp optimizer [59], an initial learning rate of 0.0001, and a training and validation batch size of 20. While learning rate reduction was active during training, the learning rate was not reduced for any of the models. The lack of learning rate reduction may indicate that further training or a larger initial learning rate could have improved results. However, this was not noticed until after the time for further training had passed. Multiple factors were considered to select the best architecture: the number of epochs until the minimum validation loss was reached, the lowest validation loss, and the lowest RMSE values for both rotation and translation.

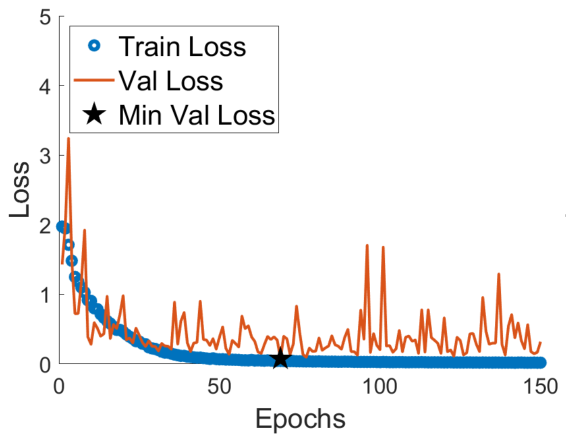
In addition to the four CNN selection models, other experiments were run with initial learning rates of 0.00001 and 0.001. The initial learning rate of 0.0001 was used because it was able to achieve a lower validation loss and reach the minimum validation loss faster than the other learning rates considered. The Adam optimizer [61] was tried as well, but the RMSProp optimizer performed better.

The training history for each of the four models is shown in Figure 7. This shows that the full FlowNet model with small images reaches the minimum validation loss faster than the other models. In addition, the validation loss is less noisy when small images are used. This is probably because the number of input parameters is reduced. The spikes in validation loss are lower when FlowNet is used. The FlowNet-Half models probably did not perform as well because of the reduced capacity compared to the full FlowNet models.

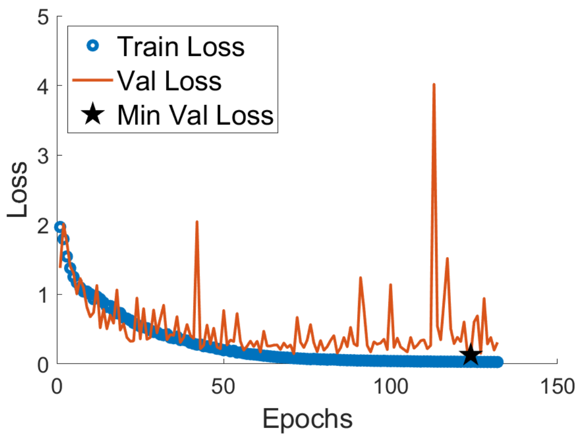
The four CNN model types were compared using their training epoch with the lowest validation loss. The results on the validation set for each model are shown in Table 8. The RMSE values for the rotation outputs are computed separately from the translation outputs. The RMSE is calculated using Equations (43) and (44).

$$\text{L2Norm}(\boldsymbol{\gamma}_{true}, \boldsymbol{\gamma}_{pred}) = \sqrt{\sum_{i=1}^{N} (\gamma_{true,i} - \gamma_{pred,i})^2} \tag{43}$$

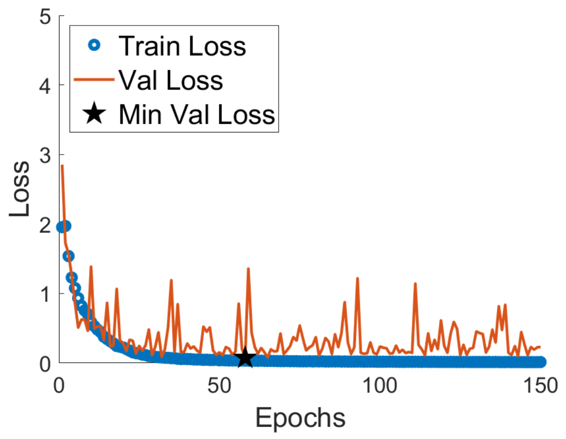
where $\boldsymbol{\gamma}_{true}$ is the true output vector, and $\boldsymbol{\gamma}_{pred}$ is the predicted output vector, and N is the length of the output vectors.



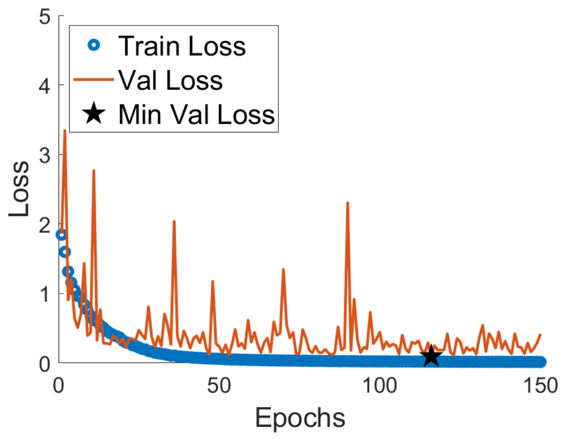
(a) FlowNet, Large Images



(b) FlowNet-Half, Large Images



(c) FlowNet, Small Images



(d) FlowNet-Half, Small Images

Figure 7: Training and validation loss history for CNN selection tests.

Table 8: Comparison of CNN selection model performance on the validation set data. Smaller values are better.

Comparison Value	**FlowNet**		**FlowNet-Half**	
	Large Images	**Small Images**	**Large Images**	**Small Images**
Rotation RMSE (deg/1000)	180.59	**144.73**	175.62	155.65
Translation RMSE (mm)	74.96	**63.67**	73.03	69.50
Min. Validation Epoch	69	**58**	124	116
Min. Validation Loss	$\mathbf{7.520 \cdot 10^{-2}}$	$7.544 \cdot 10^{-2}$	$12.913 \cdot 10^{-2}$	$9.744 \cdot 10^{-2}$

$$\text{RMSE} = \sqrt{\frac{1}{M}\sum_{j=1}^{M}\left(\text{L2Norm}(\mathbf{Y}_{true,j}, \mathbf{Y}_{pred,j})\right)^2} \tag{44}$$

where M is the number of observations, $\mathbf{Y}_{true,j}$ is a row of the true output $M \times N$

matrix, and $\mathbf{Y}_{pred,j}$ is a row of the predicted output $M \times N$ matrix.

In addition, the minimum validation loss values are shown along with the epoch number at which they appear. These are also indicated in Figure 7. The results clearly show that the FlowNet model with small image inputs trains faster and achieves more accurate prediction values. Thus, this CNN architecture is used in subsequent tests.

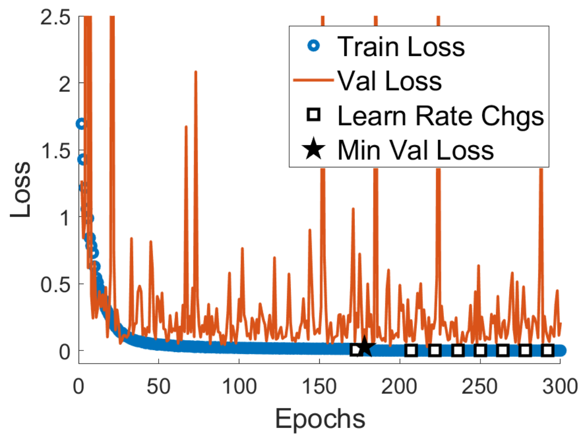
4.3 Model Aiding Evaluation

Each neural network model uses the FlowNet CNN (Section 3.3.1.1) to extract motion information from the images. The baseline and INS-aided architectures are described in Section 3.3.2.

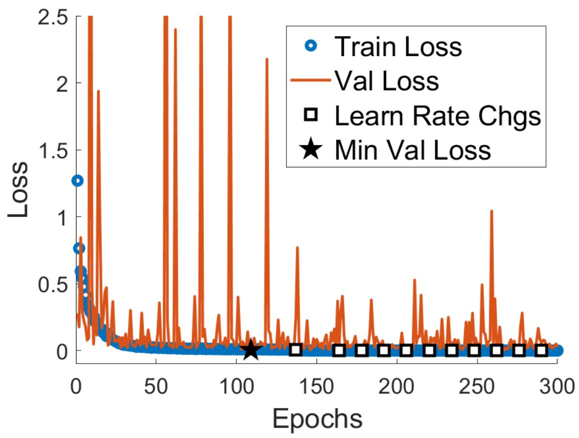
4.3.1 Training

Both of the six degrees of freedom (6-DoF) models were trained for 300 epochs with a training batch size of 20 and a validation batch size of 5. The translation and scale output models were each trained for 200 epochs with the same batch size parameters as the 6-DoF models. These epoch values were chosen to give each model the opportunity to tune the parameters to the training set. The results for the model training according to Section 3.4 are shown in Figure 8. While the training loss decreases steadily, the validation loss is noisy and spikes frequently even toward the end of training. This makes it difficult to know which training epoch will be best for further comparison. The validation loss spikes are especially high for the baseline 6-DoF model shown in Figure 8a.

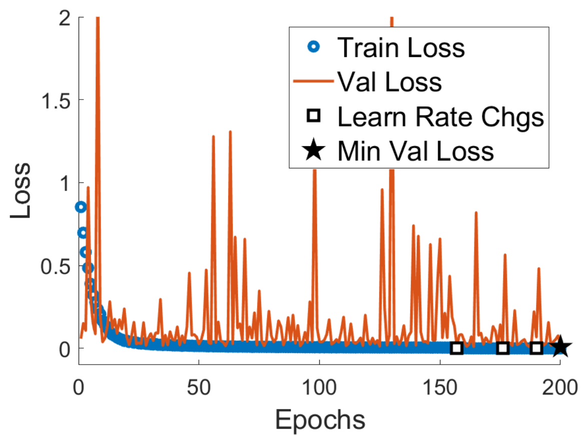
The 6-DoF models have the highest validation losses followed by the translation and scale models, respectively. This is due in part to the fact that the 6-DoF models use a different loss function than the normal Mean Squared Error (MSE) loss of the translation and scale models, as explained in Section 3.4.2.



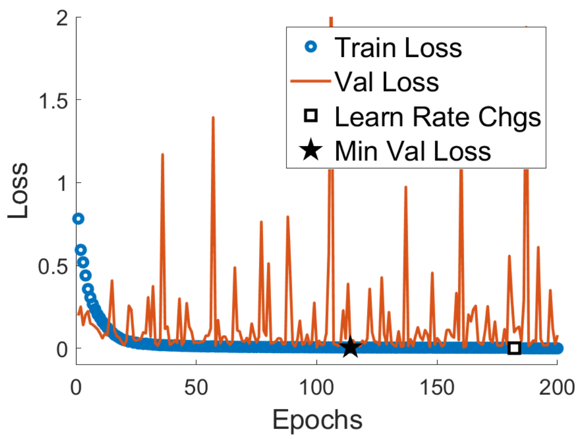
(a) 6-DoF Baseline



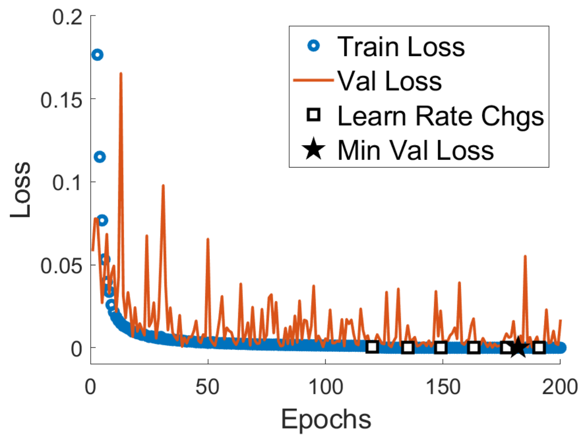
(b) 6-DoF INS-Aided



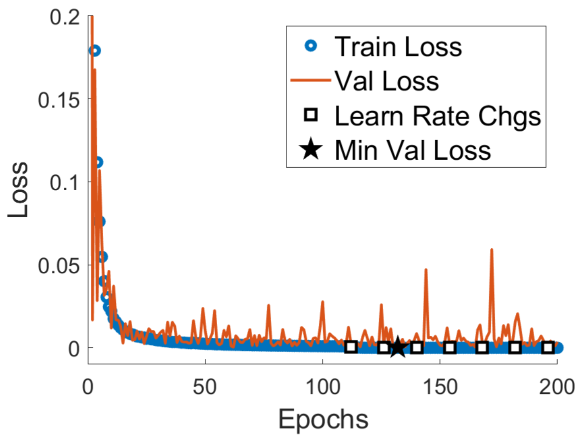
(c) Trans Baseline



(d) Trans INS-Aided

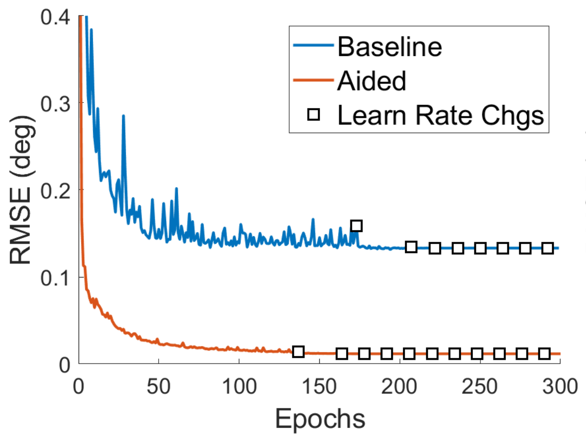


(e) Scale Baseline

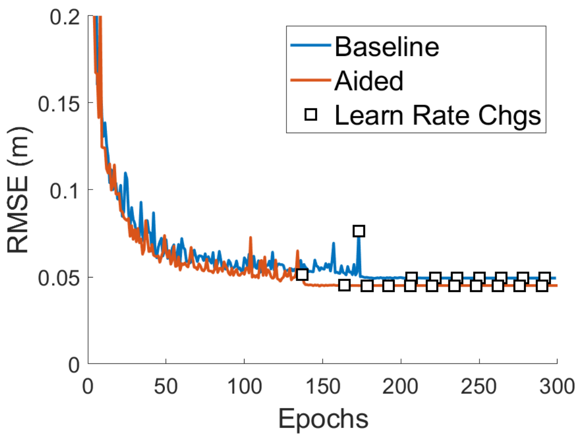


(f) Scale INS-Aided

Figure 8: Training and validation loss history for the six VO models. Each learning rate change decreases the learning rate by a factor of 0.1.



(a) Rotation



(b) Translation

Figure 9: RMSE of the rotation and translation magnitude for each training epoch of the 6-DoF models. Each learning rate change decreases the learning rate by a factor of 0.1.

The epochs where the learning rate changed are also shown in Figure 8. Figure 9 shows the actual RMSE values as training progresses for the 6-DoF models. Based on the actual RMSE, it appears that the learning rate reduction factor of 0.1 was too drastic. When the first learning rate reduction is applied, the noise in the RMSE decreases dramatically, but the model does not improve much further. The plots for the translation and scale output models show a similar effect.

In addition, the validation loss changes dramatically from epoch to epoch, but the actual RMSE in Figure 9 is not as noisy. The noise in the RMSE also decreases as training progresses, which is only slightly apparent in the corresponding validation losses given in Figures 8a and 8b. This indicates that the noise in the loss values is due to the MSE calculation on the normalized values. The input and output feature values are normalized such that the mean and standard deviation of each component over the entire training set are 0 and 1, respectively. For instance, the actual value of the y component of rotation is much larger than the x component of rotation. However, these values are normalized to be approximately the same scale in the loss function. In fact, the normalized minimum and maximum values shown in Table 2 of

Section 3.2.3.4 are larger for the axes with a smaller range. Thus, a relatively small change in the x rotation appears large in the loss calculation.

After training, the model epochs with the lowest validation set loss were chosen for evaluation on the test set. These epoch choices are shown in Table 9. Interestingly, the INS-aided models all reach their minimum validation loss earlier than their baseline counterparts. Typically, the INS-aided models have a lower validation loss as well, except in the case of the scale output model.

4.3.2 Evaluation Results

Once the best VO model in each category was selected based on the minimum validation loss, the six models were compared with one another and in pairs, i.e. each baseline model was compared with the corresponding INS-aided model that produced the same output type. In addition, all six models were compared using the scale of the translation.

4.3.2.1 Overall Error

An average prediction value was considered for comparison to the VO model predictions. First, the mean of all the test set observations was calculated for each output feature (e.g. x rotation). Next, these mean values were used as a simplistic guess of the camera's motion in place of the neural network predictions. Finally, the

Table 9: Best VO model validation set performance. Smaller values are better.

Test	**Min. Validation Epoch**	**Min. Validation Loss**
6DoF-Baseline	179	$26.511 \cdot 10^{-3}$
6DoF-INS-Aided	**109**	$\mathbf{4.112 \cdot 10^{-3}}$
Trans-Baseline	200	$8.656 \cdot 10^{-3}$
Trans-INS-Aided	**114**	$\mathbf{5.720 \cdot 10^{-3}}$
Scale-Baseline	182	$\mathbf{8.24 \cdot 10^{-5}}$
Scale-INS-Aided	**132**	$14.50 \cdot 10^{-5}$

errors shown in Table 10 were determined by comparing these mean value predictions with the true motion value for all test set observations. The RMSE in millimeters was used for the translation outputs and the RMSE in thousandths of degrees was used for the rotation outputs.

Each VO model epoch selected using the validation set loss was evaluated by estimating the camera's motion on the test set data. Up to this point, the test set was not used for any training or model selection tasks. This was done so that the results would be representative of performance on real-world data that was gathered using a similar approach and environment to the KITTI dataset.

Each test set VO prediction was used to determine the RMSE for that model. The RMSE results for the VO models are shown in Table 11. This includes the RMSE values for each individual output and the combined RMSE magnitude value for both

Table 10: Test set RMSE results when the test set mean values are used as the VO prediction for each output. The rotation values presented are the individual axis components of a rotation vector. Smaller values are better.

Mean Prediction	**X**	**Y**	**Z**	**Mag**
Rot RMSE (deg/1000)	175.72	1010.62	155.85	1037.56
Trans RMSE (mm)	18.31	18.31	430.95	431.73

Table 11: Test set RMSE results for the six VO models. The rotation values presented are the individual axis components of a rotation vector. Smaller values are better.

Test	**Rotation (deg/1000)**			
	X	**Y**	**Z**	**Mag**
6DoF-Baseline	50.22	90.70	81.35	131.78
6DoF-INS-Aided	**3.79**	**14.95**	**3.79**	**15.88**

Test	**Translation (mm)**			
	X	**Y**	**Z**	**Mag**
6DoF-Baseline	10.38	**6.48**	51.10	52.54
6DoF-INS-Aided	**10.00**	6.49	**49.01**	**50.44**
Trans-Baseline	10.81	6.24	47.41	49.02
Trans-INS-Aided	**10.08**	**5.70**	**47.04**	**48.45**
Scale-Baseline				37.92
Scale-INS-Aided				**32.06**

the rotation and the translation outputs. These results show that the rotation RMSE decreases by an order of magnitude when INS-aiding is used, both for the individual components and the magnitude. While these results are positive, the amount of rotation error in the simulated INS data (Table 5 in Section 3.2.4) is approximately $2.65 \cdot 10^{-3}$ deg for each individual component, which is still less than the RMSE of the aided prediction values shown in Table 11. Based on these results, if a similarly accurate INS is available, the INS rotation measurement should be used instead of the neural network rotation prediction.

The translation RMSEs are not as drastically different. In most cases, the networks that use aiding do better than their baseline counterparts, but not by much. In general, the 6-DoF model RMSE values are larger than the translation model RMSE values. The scale model RMSE values are smaller than either of the other two network types. Thus, the network performs better when it only needs to predict fewer

Table 12: Test set error mean and standard deviation results for the six VO models. The rotation values presented are related to the individual axis components of a rotation vector. Mean values closer to 0 and smaller standard deviation values are better.

Test	**Rotation (deg/1000)**					
	X		**Y**		**Z**	
	Mean	**Std Dev**	**Mean**	**Std Dev**	**Mean**	**Std Dev**
6DoF-Baseline	4.855	49.99	4.898	90.58	3.779	81.27
6DoF-INS-Aided	**-0.394**	**3.77**	**-3.768**	**14.47**	**-1.217**	**3.59**

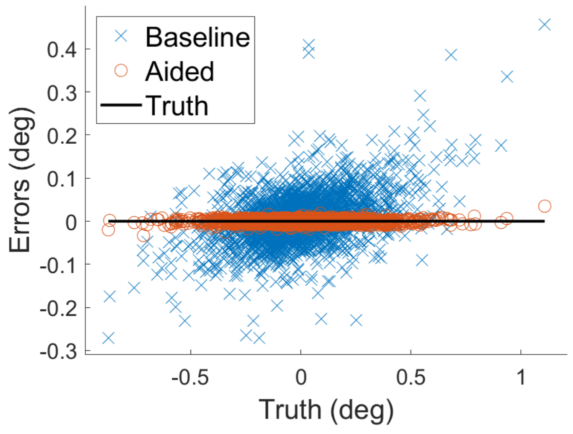
Test	**Translation (mm)**					
	X		**Y**		**Z**	
	Mean	**Std Dev**	**Mean**	**Std Dev**	**Mean**	**Std Dev**
6DoF-Baseline	-0.125	10.38	0.201	**6.48**	**0.494**	51.10
6DoF-INS-Aided	**-0.056**	**10.00**	**-0.006**	6.49	3.514	**48.89**
Trans-Baseline	**-0.165**	10.81	0.336	6.23	**0.039**	47.41
Trans-INS-Aided	0.205	**10.08**	**0.274**	**5.69**	-0.062	**47.05**

Test	**Scale (mm)**	
	Mean	**Std Dev**
Scale-Baseline	-0.687	37.92
Scale-INS-Aided	**0.636**	**32.06**

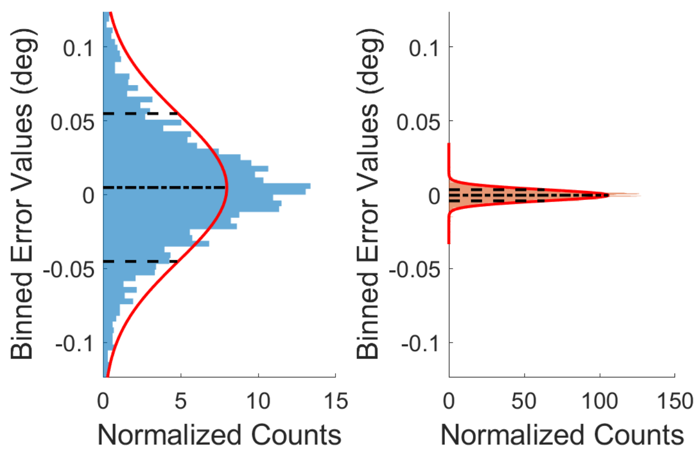
output values.

In addition to the RMSE, the mean and standard deviation of the errors for each output were calculated. These results are shown in Table 12. In general, the INS-aided networks did better than the baseline networks. The standard deviation values correspond directly with the RMSE values shown in Table 11. Thus, in general the standard deviation of the rotation errors decreases by an order of magnitude. Also, the error mean values are generally close to zero and decrease in magnitude when aiding is used. The main outlier to this effect is the z translation which has a larger error mean when aiding is used.

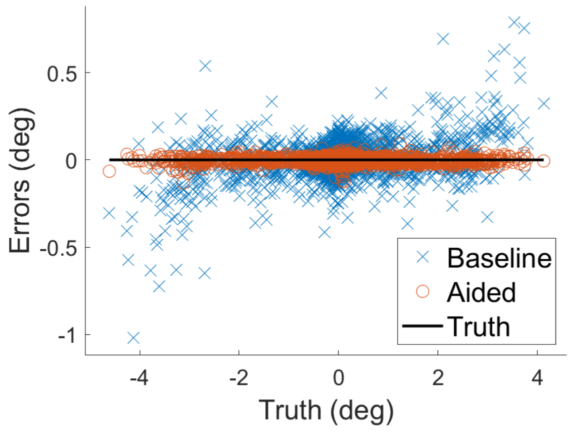
The test set errors for each of the outputs of the 6-DoF networks are shown in Figures 10 and 11. The baseline error distribution is compared with the INS-aided error distribution. Likewise, the translation network outputs are shown in Figure 12 and the scale network output is shown in Figure 13a.



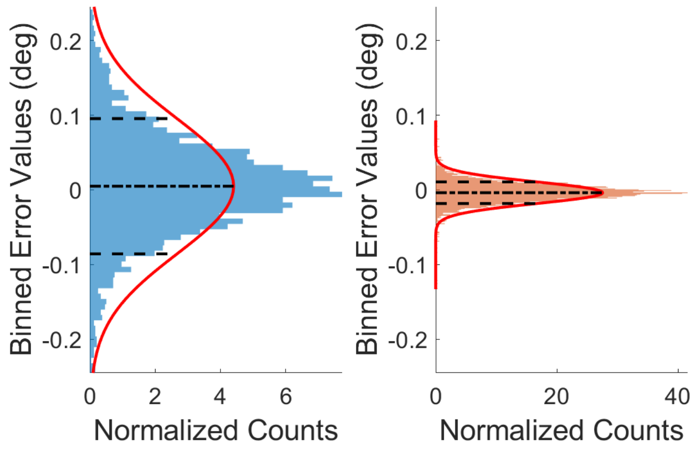
(a) 6-DoF Test Error - Rotation X



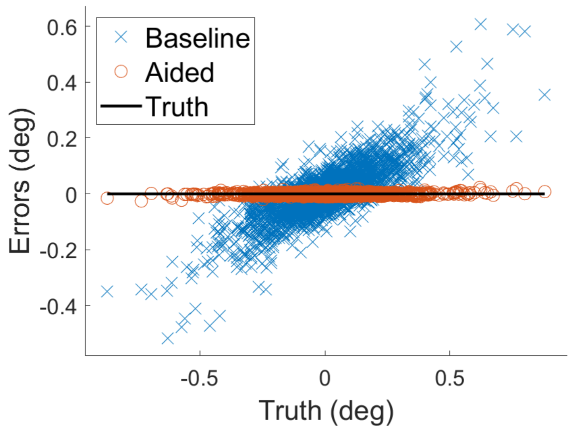
(b) Baseline (left) and Aided (right) Histograms Compared with Gaussian Fit - 6-DoF Rotation X



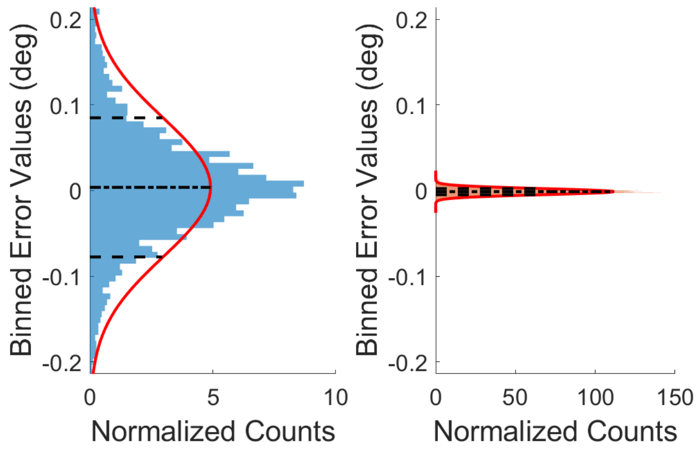
(c) 6-DoF Test Error - Rotation Y



(d) Baseline (left) and Aided (right) Histograms Compared with Gaussian Fit - 6-DoF Rotation Y

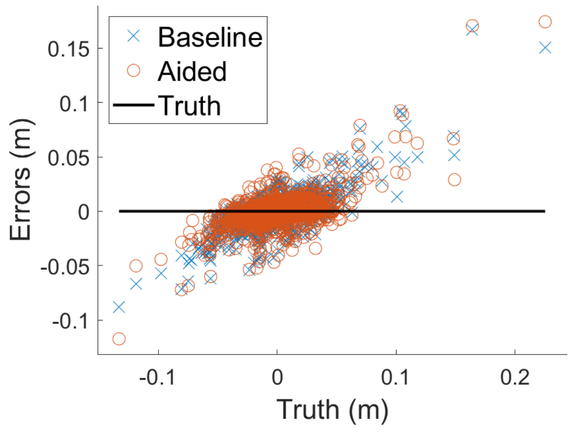


(e) 6-DoF Test Error - Rotation Z

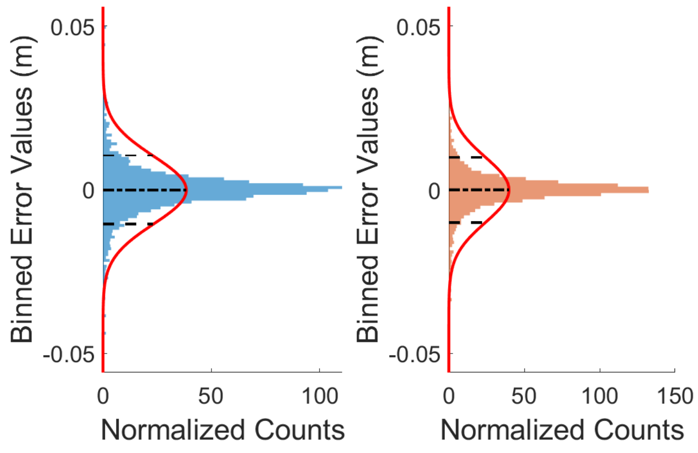


(f) Baseline (left) and Aided (right) Histograms Compared with Gaussian Fit - 6-DoF Rotation Z

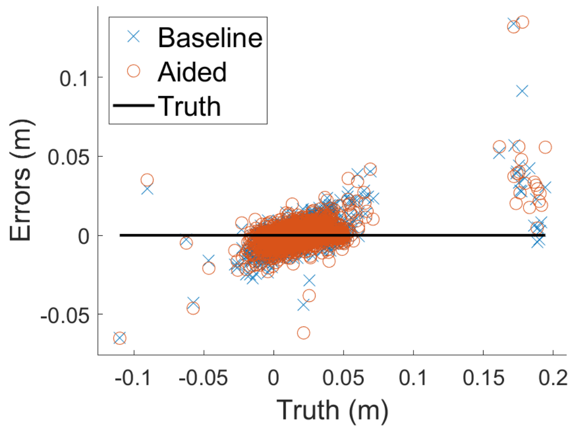
Figure 10: Test set rotation errors for both the baseline and INS-aided 6-DoF VO models. The rotation values presented are the individual axis components of a rotation vector.



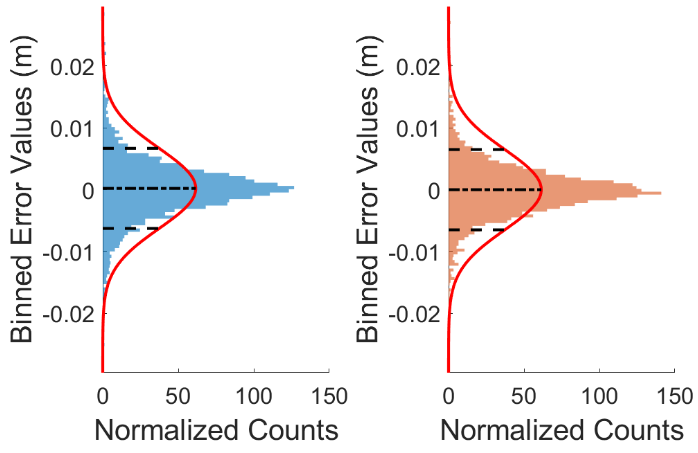
(a) 6-DoF Test Error - Translation X



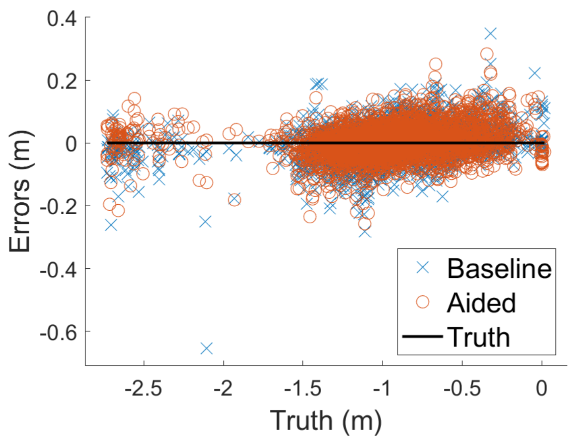
(b) Baseline (left) and Aided (right) Histograms Compared with Gaussian Fit - 6-DoF Translation X



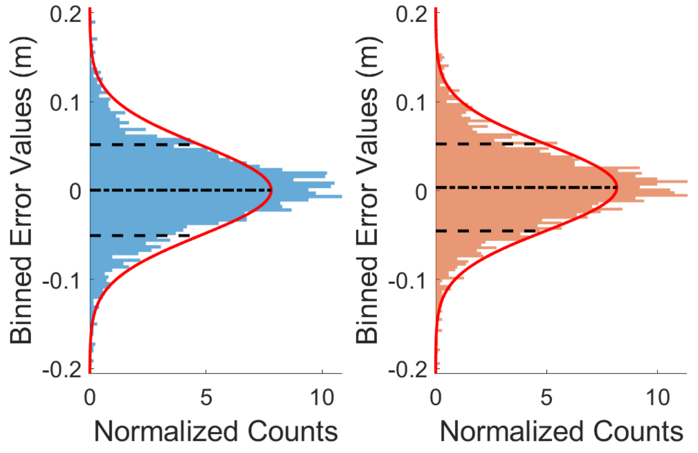
(c) 6-DoF Test Error - Translation Y



(d) Baseline (left) and Aided (right) Histograms Compared with Gaussian Fit - 6-DoF Translation Y

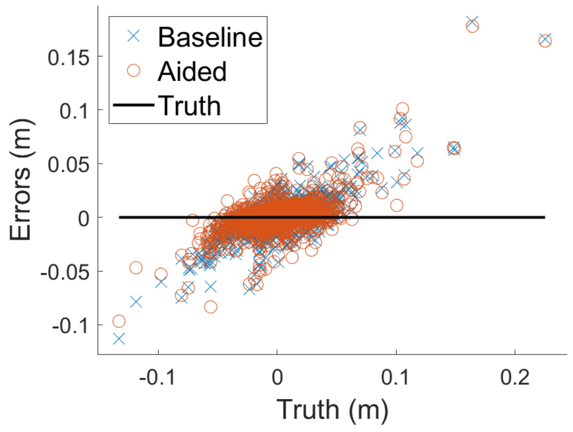


(e) 6-DoF Test Error - Translation Z

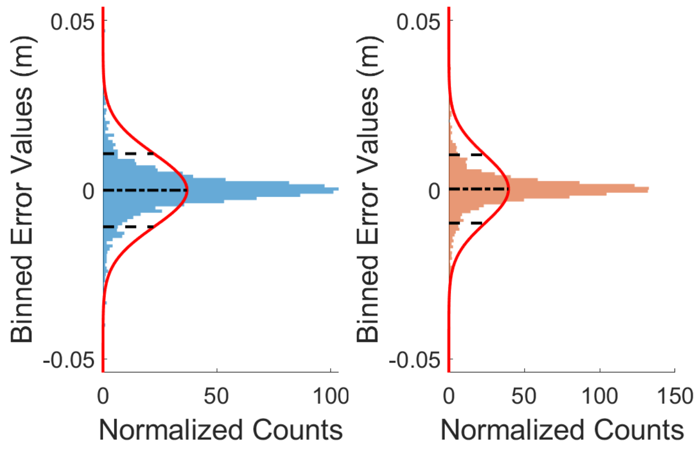


(f) Baseline (left) and Aided (right) Histograms Compared with Gaussian Fit - 6-DoF Translation Z

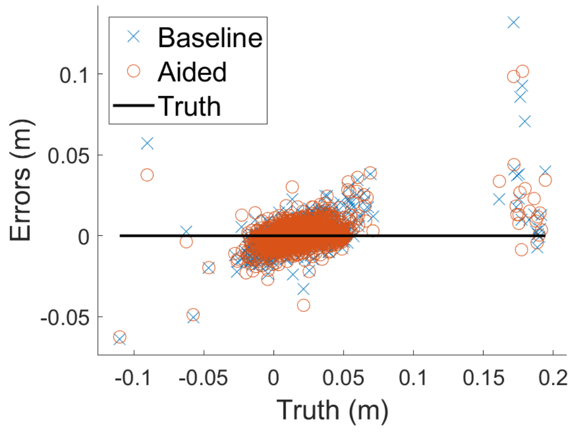
Figure 11: Test set translation errors for both the baseline and INS-aided 6-DoF VO models.



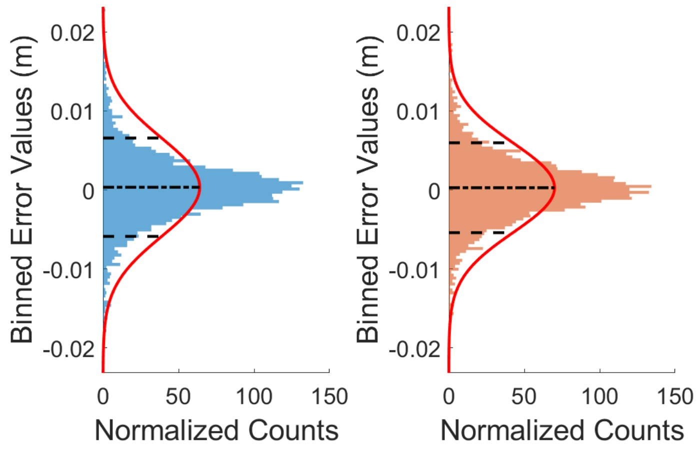
(a) Trans Test Error - Translation X



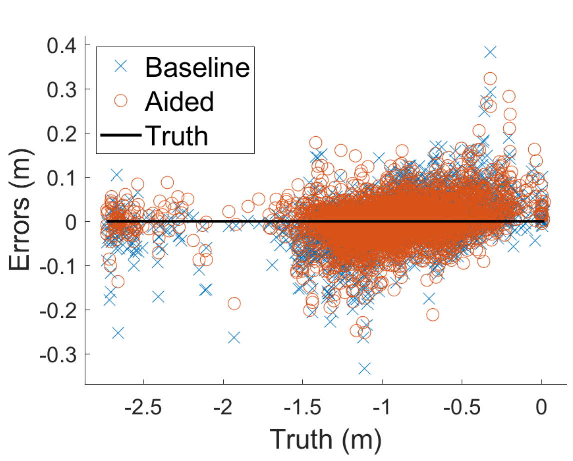
(b) Baseline (left) and Aided (right) Histograms Compared with Gaussian Fit - Trans model Translation X



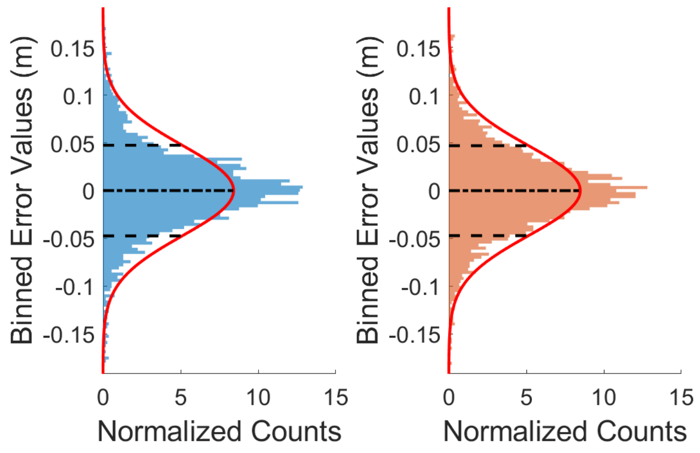
(c) Trans Test Error - Translation Y



(d) Baseline (left) and Aided (right) Histograms Compared with Gaussian Fit - Trans model Translation Y

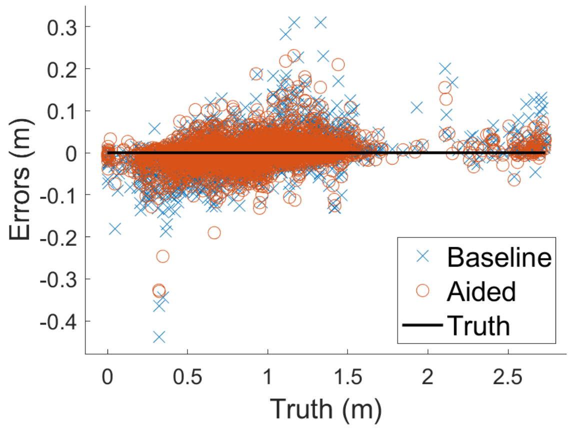


(e) Trans Test Error - Translation Z

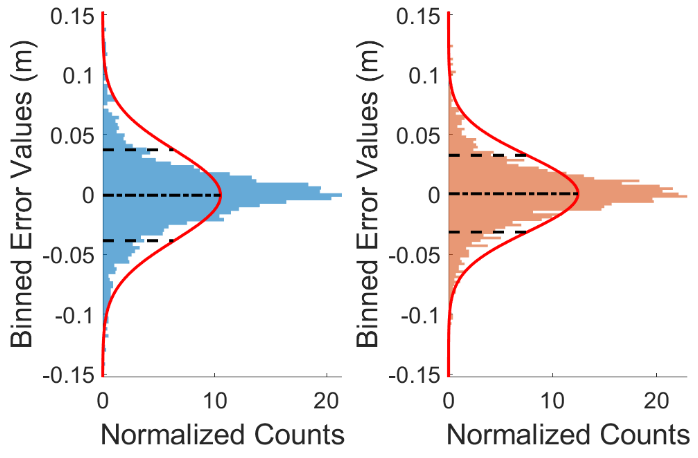


(f) Baseline (left) and Aided (right) Histograms Compared with Gaussian Fit - Trans model Translation Z

Figure 12: Test set errors for both the baseline and INS-aided Cartesian translation VO models.



(a) Scale Test Error - Scale



(b) Baseline (left) and Aided (right) Histograms Compared with Gaussian Fit - Scale

Figure 13: Test set errors for both the baseline and INS-aided Scale VO models.

The error distributions show visual results consistent with the error statistics already presented. The rotation errors in Figure 10 are drastically decreased when INS-aiding is used. The translation output errors all appear generally the same whether aiding is used or not. In addition, the baseline models show a correlation between the true movement value and the sign of the prediction errors. The predicted values have a smaller magnitude than the true values, especially as the true value magnitude increases. This is apparent for the baseline model errors in the x and z rotation (Figure 10) and the x and y translation plots (Figures 11 and 12). It appears that the z rotation baseline error correlation is removed or drastically reduced when aiding is used.

The error histograms in Figures 10-13a, show that the vast majority of the errors are centered around zero. Thus, the outliers do not really affect the overall distribution. In general, the rotation errors shown in Figure 10 get closer to a Gaussian distribution when INS-aiding is used. On the other hand, the translation errors in Figures 11-13a seem to only get slightly closer to a Gaussian distribution with the x and y translations in Figures 11b, 11d, 12b and 12d being especially non-Gaussian.

4.3.2.2 Turning vs. Non-Turning

In order to determine if the error effects were different when the vehicle was turning or not turning, the RMSE, error mean, and error standard deviation values were calculated for these cases as well. The test RMSE values of the turning and non-turning image pairs are shown in Table 13. In general, the RMSE values are larger for turning observations vs. non-turning observations. For both turning and non-turning image pairs, the rotation predictions are better when INS-aiding is used. For turning observations, the 6-DoF baseline network performed better than the aided network. For non-turning observations, the aided networks did better in nearly every case. These results are consistent with the overall RMSE results in Table 11. Overall, when aiding is used, the rotation RMSE decreases by an order of magnitude while the

Table 13: Test set RMSE results on turning and non-turning image pairs for the six VO models. The rotation values presented are the individual axis components of a rotation vector. Smaller values are better.

Type	**Test**	**Rotation (deg/1000)**			
		X	**Y**	**Z**	**Mag**
Turn	6DoF-Baseline	53.27	166.58	104.54	203.75
	6DoF-INS-Aided	**3.97**	**19.73**	**4.24**	**20.56**
Non-Turn	6DoF-Baseline	49.50	61.77	75.12	109.13
	6DoF-INS-Aided	**3.75**	**13.64**	**3.68**	**14.62**

Type	**Test**	**Translation (mm)**			
		X	**Y**	**Z**	**Mag**
Turn	6DoF-Baseline	**19.96**	**6.80**	**54.59**	**58.52**
	6DoF-INS-Aided	20.09	6.80	55.75	59.65
Non-Turn	6DoF-Baseline	6.49	**6.41**	50.27	51.09
	6DoF-INS-Aided	**5.60**	6.42	**47.36**	**48.11**
Turn	Trans-Baseline	21.64	7.55	**55.14**	**59.72**
	Trans-INS-Aided	**20.51**	**5.74**	57.51	61.32
Non-Turn	Trans-Baseline	6.12	5.90	45.48	46.26
	Trans-INS-Aided	**5.44**	**5.69**	**44.34**	**45.03**
Turn	Scale-Baseline				47.15
	Scale-INS-Aided				**36.69**
Non-Turn	Scale-Baseline				35.51
	Scale-INS-Aided				**30.92**

Table 14: Test set error mean and standard deviation results on turning and non-turning image pairs for the six VO models. The rotation values presented are related to the individual axis components of a rotation vector. Mean values closer to 0 and smaller standard deviation values are better.

Type	**Test**	**Rotation (deg/1000)**					
		X		**Y**		**Z**	
		Mean	**Std Dev**	**Mean**	**Std Dev**	**Mean**	**Std Dev**
Turn	6DoF-Baseline	4.075	53.16	**-2.748**	166.69	5.421	104.48
	6DoF-INS-Aided	**-0.366**	**3.96**	-4.572	**19.21**	**-1.244**	**4.06**
Non-Turn	6DoF-Baseline	5.031	49.25	6.625	61.43	3.408	75.06
	6DoF-INS-Aided	**-0.400**	**3.73**	**-3.586**	**13.16**	**-1.212**	**3.47**

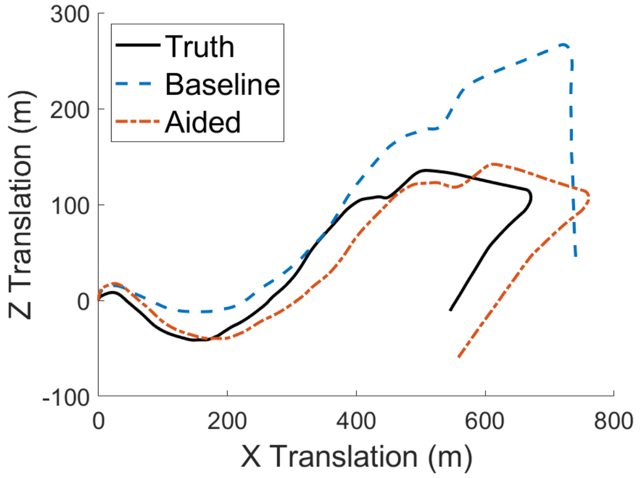
Type	**Test**	**Translation (mm)**					
		X		**Y**		**Z**	
		Mean	**Std Dev**	**Mean**	**Std Dev**	**Mean**	**Std Dev**
Turn	6DoF-Baseline	0.054	**19.97**	0.261	**6.80**	**3.963**	**54.50**
	6DoF-INS-Aided	**-0.030**	20.11	**0.004**	6.81	4.843	55.59
Non-Turn	6DoF-Baseline	-0.165	6.49	0.188	**6.41**	**-0.290**	50.28
	6DoF-INS-Aided	**-0.061**	**5.60**	**-0.008**	6.42	3.214	**47.26**
Turn	Trans-Baseline	**-0.088**	21.66	0.422	7.55	8.731	**54.49**
	Trans-INS-Aided	0.980	**20.50**	**0.279**	**5.74**	**5.635**	57.28
Non-Turn	Trans-Baseline	-0.182	6.12	0.317	5.90	-1.924	45.44
	Trans-INS-Aided	**0.029**	**5.44**	**0.273**	**5.68**	**-1.349**	**44.33**

Type	**Test**	**Scale (mm)**	
		Mean	**Std Dev**
Turn	Scale-Baseline	-10.303	46.05
	Scale-INS-Aided	**-2.599**	**36.63**
Non-Turn	Scale-Baseline	1.485	35.48
	Scale-INS-Aided	**1.366**	**30.89**

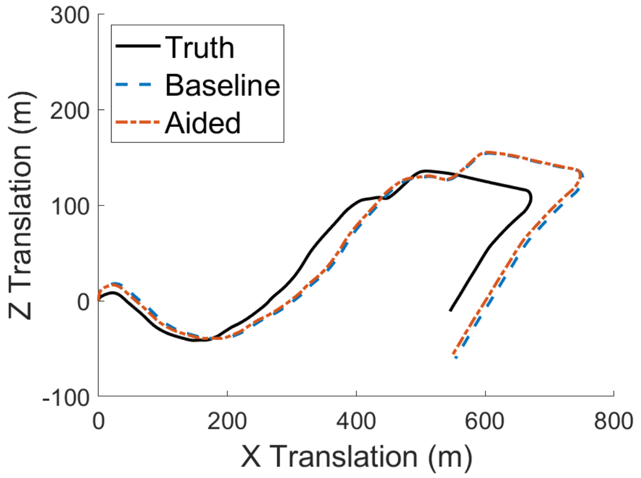
translation RMSE only decreases slightly. The error mean and standard deviation results shown in Table 14 show similar results to Table 13 and to Table 12.

4.4 Full Trajectory Evaluation

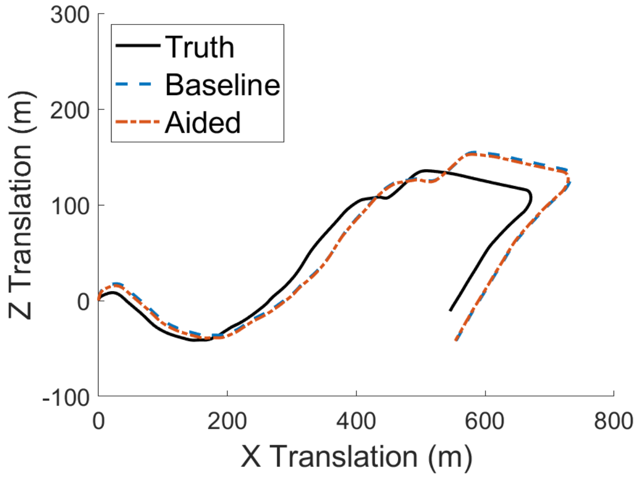
In addition to the test set, the models were also evaluated using all of sequence 10 from the KITTI Odometry dataset [2]. Each of the 1200 observations in the sequence were given as input to the models. This allowed the individual predictions to be integrated so that the each model's performance over a full trajectory could be



(a) 6-DoF Models



(b) Translation Models



(c) Scale Models

Figure 14: Bird's-eye views of the predicted vehicle trajectories for sequence 10 of the KITTI Odometry dataset [2].

visualized. For each of the three output types, the full trajectory predictions were compared with each other and with the true trajectory.

The full 6-DoF model results are shown in Figure 14a. It is immediately apparent that the rotation predictions are much better with INS aiding, but translation errors are not reduced by much. The translation model results are shown in Figure 14b and the scale model results are shown in Figure 14c. Because there were no rotation predictions for these model types, the true rotation is used for these plots. In addition, for the scale models, the true translation direction angles are used to convert the scale back into Cartesian coordinates.

The RMSE analysis methods that were run on the test set in Section 4.3 were also run on the data from sequence 10. The mean prediction RMSE analysis table for sequence 10 is shown in Table 15.

The RMSE results for the six neural network models are shown in Table 16 for all of the observations and Table 17 for the turning and non-turning observations. The sequence 10 rotation results are similar to the test set results (Tables 11 and 13). However, the baseline model rotation errors are much higher for sequence 10. The translation errors are also much higher for sequence 10, especially in the z axis. In addition, the 6-DoF aided model did worse than the 6-DoF baseline model when making translation predictions.

These results do not support the conclusion that INS aiding helps the translation predictions in these neural network models. This could suggest that because the test

Table 15: Sequence 10 RMSE results when the sequence 10 mean values are used as the VO prediction for each output. The rotation values presented are the individual axis components of a rotation vector. Smaller values are better.

Mean Prediction	**X**	**Y**	**Z**	**Mag**
Rot RMSE (deg/1000)	212.24	848.39	166.84	890.31
Trans RMSE (mm)	15.67	11.37	334.65	335.21

Table 16: Sequence 10 RMSE results for the six VO models. The rotation values presented are the individual axis components of a rotation vector. Smaller values are better.

Test	**Rotation (deg/1000)**			
	X	**Y**	**Z**	**Mag**
6DoF-Baseline	83.22	210.38	116.69	254.56
6DoF-INS-Aided	**3.75**	**18.44**	**3.61**	**19.16**

Test	**Translation (mm)**			
	X	**Y**	**Z**	**Mag**
6DoF-Baseline	13.47	**9.46**	**170.96**	**171.75**
6DoF-INS-Aided	**12.96**	10.20	260.63	261.16
Trans-Baseline	14.13	**10.38**	**276.30**	**276.86**
Trans-INS-Aided	**12.94**	10.80	277.51	278.02
Scale-Baseline				231.05
Scale-INS-Aided				**206.68**

Table 17: Sequence 10 RMSE results on turning and non-turning image pairs for the six VO models. The rotation values presented are the individual axis components of a rotation vector. Smaller values are better.

Type	**Test**	**Rotation (deg/1000)**			
		X	**Y**	**Z**	**Mag**
Turn	6DoF-Baseline	108.35	483.63	146.21	516.73
	6DoF-INS-Aided	**4.64**	**31.98**	**4.59**	**32.64**
Non-Turn	6DoF-Baseline	78.47	118.44	111.27	180.46
	6DoF-INS-Aided	**3.59**	**15.21**	**3.43**	**16.00**

Type	**Test**	**Translation (mm)**			
		X	**Y**	**Z**	**Mag**
Turn	6DoF-Baseline	29.41	**6.73**	**163.33**	**166.09**
	6DoF-INS-Aided	**28.54**	8.50	255.73	257.46
Non-Turn	6DoF-Baseline	8.51	**9.82**	**172.15**	**172.64**
	6DoF-INS-Aided	**8.06**	10.44	261.41	261.74
Turn	Trans-Baseline	30.11	**8.35**	**235.23**	**237.30**
	Trans-INS-Aided	**28.51**	9.18	235.85	237.74
Non-Turn	Trans-Baseline	9.33	**10.67**	**282.29**	**282.65**
	Trans-INS-Aided	**8.04**	11.03	283.58	283.91
Turn	Scale-Baseline				225.35
	Scale-INS-Aided				**176.78**
Non-Turn	Scale-Baseline				231.94
	Scale-INS-Aided				**211.06**

set contained samples taken from in between the training set samples, the neural networks could be using the learned distributions of the training set to interpolate the test set predictions. Because no sequence 10 observations are used in the training set, this interpolation is not possible.

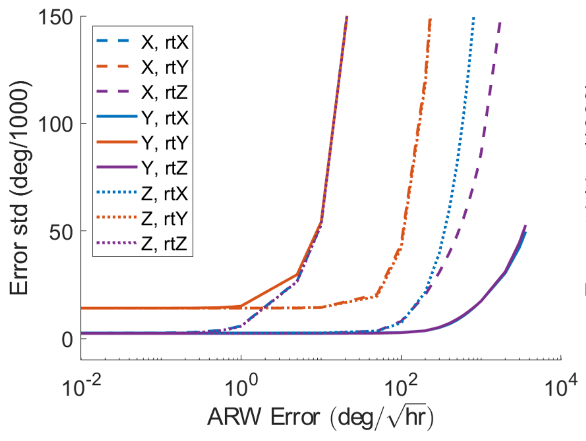
For the turning and non-turning errors in Table 17, the rotation errors are lower on the non-turning image pairs. This is to be expected as the amount of rotation is also lower. With the exception of the x axis, the translation errors are higher when the vehicle is not turning.

4.5 INS-Aided Model Sensitivity Tests

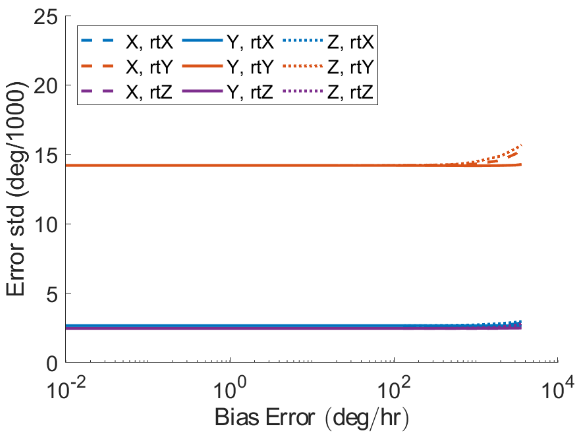
Each INS-aided network was tested for sensitivity to errors in the input rotation values. Two different types of errors were evaluated: bias error and angular random walk (ARW) or sensor error. Each type of error was applied to each component of the true rotation separately. For instance, a bias error of a given strength was added to the x component of the true rotation and then given as input to an INS-aided model to produce the a set of VO predictions. Likewise, this was done for the y and z components. These predictions were then subtracted from the true motion values to obtain the amount of estimation error for each observation. The mean and standard deviation of the output estimation errors were then calculated for each set of VO predictions. By varying the strength of the bias or ARW error, the effect of the errors on the VO estimation can be observed. This analysis helps to characterize the Inertial Measurement Unit (IMU) quality needed to produce similar VO estimation results.

4.5.1 Bias Error Sensitivity

A bias error changes the INS measurements by a fixed offset. These can produce significant drift in VO estimation because the bias error integrates over the number of samples. For the bias error sensitivity analysis, a fixed bias amount was added to the true rotations in the test set. These noisy rotations were then given to the neural network along with the corresponding image pairs to produce VO estimates. This was done for all three of the INS-aided models. The means and standard deviations of estimation errors at different bias strengths are shown in the right half of Figures 15, 16 and 17. Figure 15 shows the error standard deviation responses of the 6-DoF rotation outputs. The translation error plots for each model are given with the mean responses in Figure 16 and the standard deviation responses in Figure 17. According to [55], the NovAtel HG1700-AG62 has a gyro rate bias drift of 5.0 deg/hr. It can be clearly seen in the plots that this amount of bias error present in the rotation inputs would not change the results of network training, even though the rotation estimates used for training did not have any bias error.

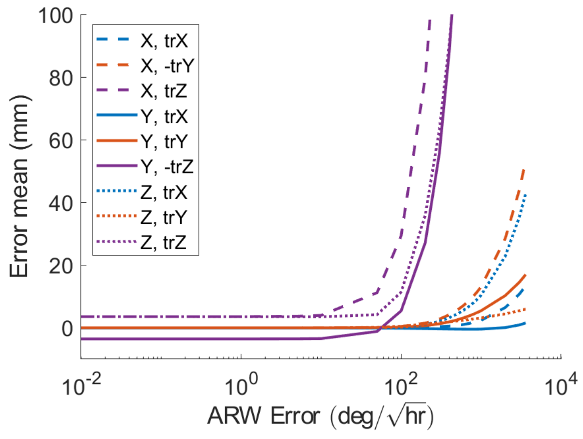


(a) 6-DoF ARW - Rotation Std Deviation

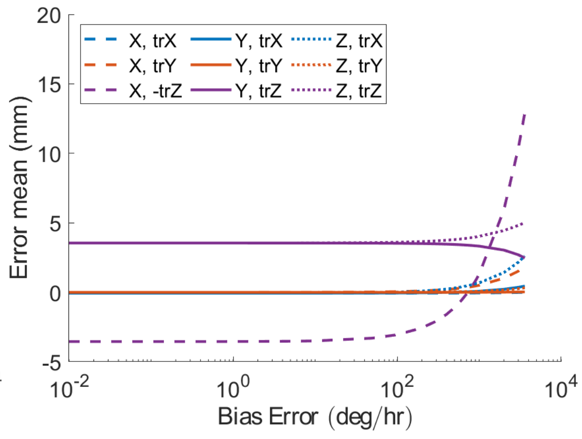


(b) 6-DoF Bias - Rotation Std Deviation

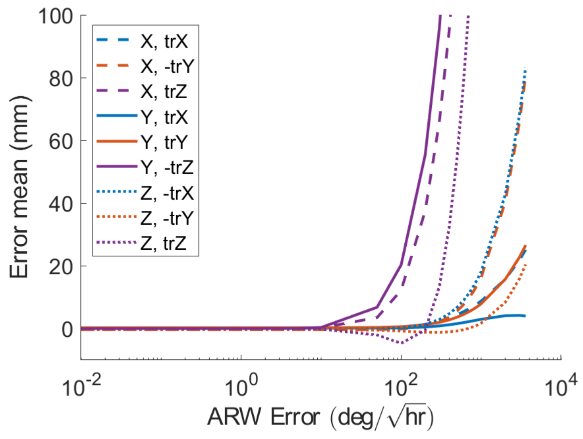
Figure 15: Rotation error sensitivity to ARW and bias errors for the INS-Aided 6-DoF VO models. The rotation values presented are related to the individual axis components of a rotation vector. The legend elements identify the INS axis to which the error was applied and the output response shown, respectively.



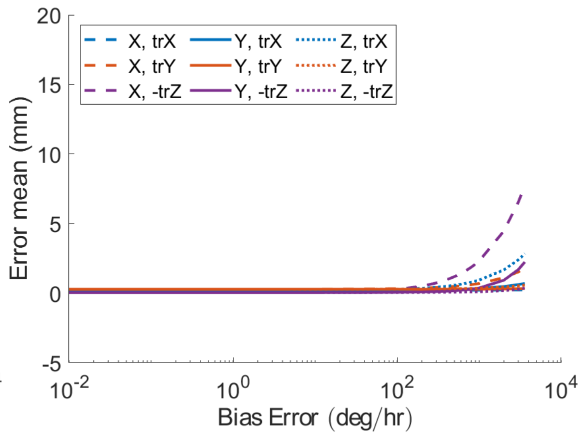
(a) 6-DoF ARW - Translation Mean



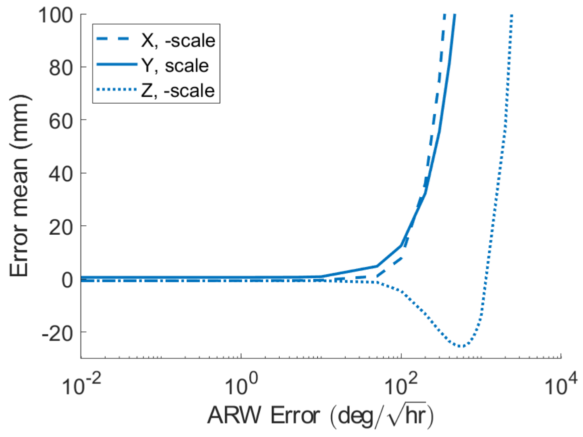
(b) 6-DoF Bias - Translation Mean



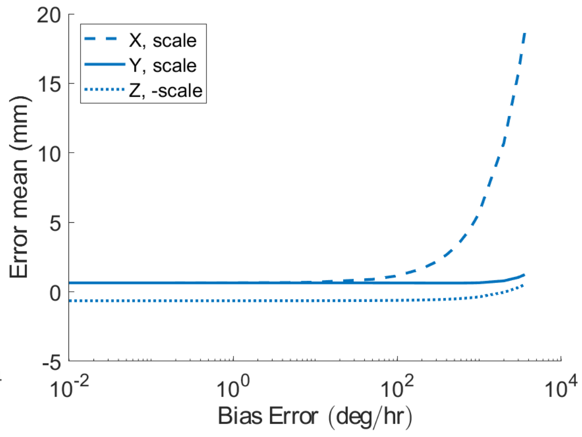
(c) Translation ARW - Mean



(d) Translation Bias - Mean

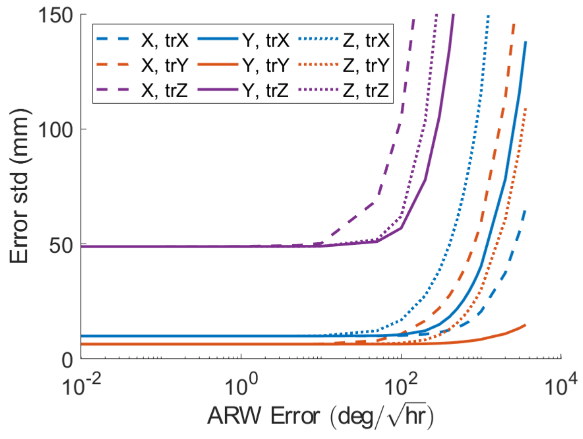


(e) Scale ARW - Mean

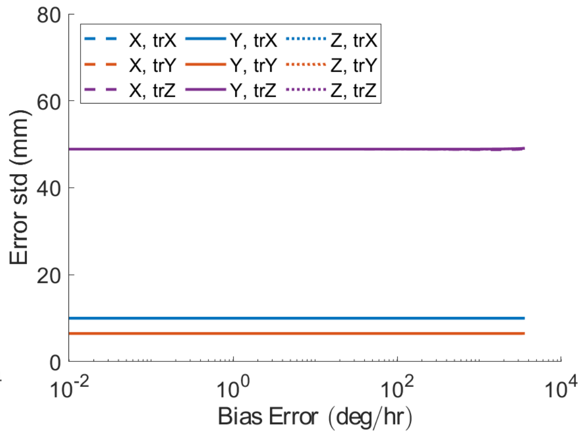


(f) Scale Bias - Mean

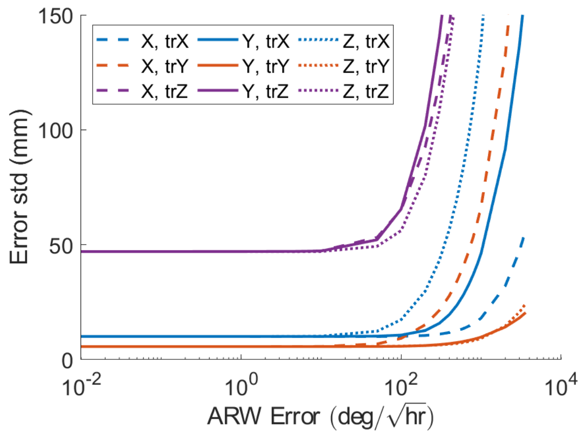
Figure 16: Translation error mean sensitivity to ARW and bias errors for the INS-Aided VO models. The legend elements identify the INS axis to which the error was applied and the output response shown, respectively. In the mean plots, when a curve ended negative, it was flipped across the x axis for ease of comparison.



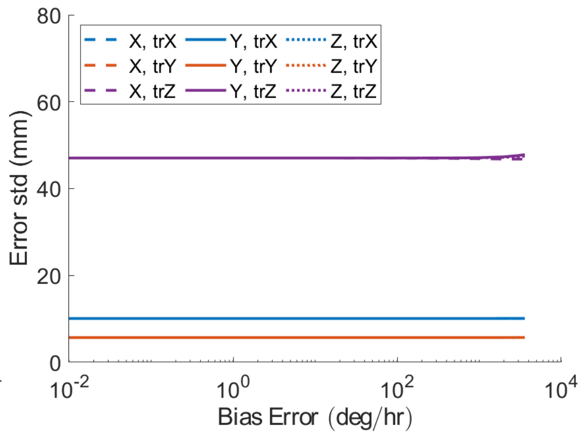
(a) 6-DoF ARW - Translation Std Dev



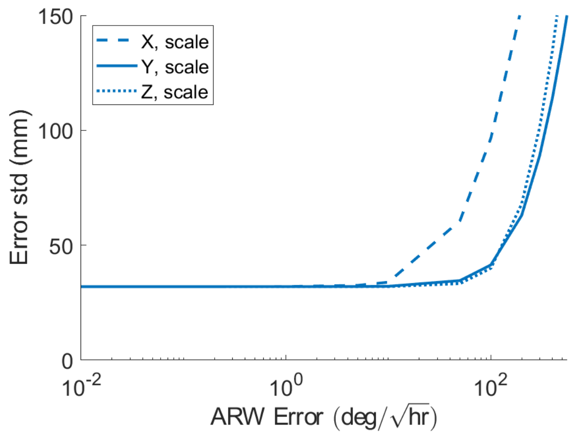
(b) 6-DoF Bias - Translation Std Dev



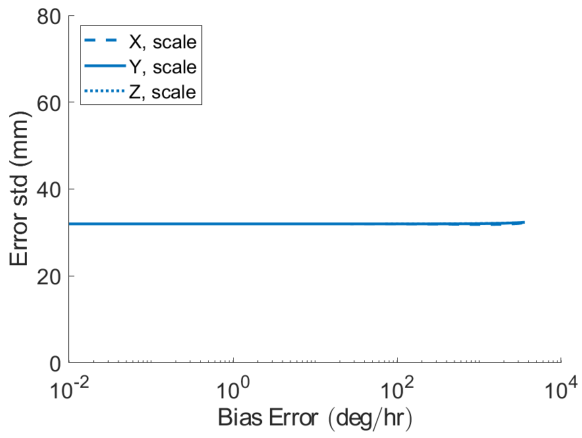
(c) Translation ARW - Std Dev



(d) Translation Bias - Std Dev



(e) Scale ARW - Std Dev



(f) Scale Bias - Std Dev

Figure 17: Translation error standard deviation sensitivity to ARW and bias errors for the INS-Aided VO models. The legend elements identify the INS axis to which the error was applied and the output response shown, respectively.

4.5.2 ARW Error Sensitivity

The ARW error is related to sensor noise. It is present even when the IMU is stationary. This ARW error can be modeled using white Gaussian noise of a particular strength. A sample of random Gaussian numbers was obtained using python's `numpy` library. The same seed value for generating the Gaussian noise was used for all of the ARW tests so that they could be compared with one another. These noise values were then multiplied by the a particular strength value and added to the true rotations. Like with the bias sensitivity tests, the ARW error was only added to one rotation component for each test so that the error effects in each axis could be observed. The results are shown in the left half of Figures 15, 16 and 17. Like with the bias error, the rotation error responses are given in Figure 15, the translation error mean responses are given in Figure 16, and the translation error standard deviation responses are given in Figure 17.

The NovAtel HG1700-AG62 [55] has an ARW error of 0.5 deg/$\sqrt{\text{hr}}$. This is the strength of the error that was used during training for the INS-aided networks. It can be seen that for most of the outputs, this amount of ARW error does not affect the mean or standard deviation response. However, the rotation error standard deviations shown in Figure 15a start changing significantly at approximately 1 deg/$\sqrt{\text{hr}}$ error.

4.6 Summary

The results show that the validation loss noise during training made it difficult to know which epoch was the best for further evaluation. This may have been due in part to the normalization of the rotation and translation values. That said, the aiding networks did better in general than the baseline networks on the test set, especially when predicting rotation. However, the baseline networks did noticeably better than the aided networks when predicting the sequence 10 translations. The test set translation results were not as stark and seem to show that there was only marginal improvement when using aiding.

V. Conclusions

5.1 Significance of Results

The Convolutional Neural Network (CNN) selection results show that larger input images did not necessarily help with the Visual Odometry (VO) calculation. Because the validation loss was noisy, this made it difficult to know which epoch was actually the best. However, the noise in the validation loss did not seem to affect the actual error which decreased steadily during training. In addition, the results indicate that the learning rate reduction factor was too drastic because of the considerable lack of further improvement once the learning rate was reduced. Unfortunately, this was not noticed until after all models training and time was short.

The results of the Inertial Navigation System (INS)-aiding tests indicate that aiding reduces errors in rotation estimation by nearly an order of magnitude when compared with the baseline models. However, these errors are still larger than the amount of error present in the aiding inputs. A different neural network may be able to obtain rotation estimates that are better than the rotation input, but for these models, the overall VO result would be better if the INS inputs were used directly as the rotation estimate instead of the network's rotation prediction.

For translation, the results are more varied. Overall, the INS-aiding seems to improve the VO performance, but only slightly. However, The sequence 10 errors are not consistent with this, showing that the baseline networks did better. In general, when a network is estimating fewer outputs it performs better.

The results of plotting the error distributions show that when INS-aiding is used, the rotation errors become more Gaussian in distribution. This is especially apparent in the x and z rotation errors, but can also be seen in the y rotation errors. The translation error distributions do not appear more Gaussian, especially for x and y

translation. However, the z translation error distribution does appear slightly closer to Gaussian when aiding is used. The translational scale distribution does not appear Gaussian. These results seem to indicate that as the errors are reduced, they become more Gaussian in distribution. This could help with modelling these errors when fusing these neural network estimations with other navigation sensor measurements.

The results show that aiding a CNN with geometric inputs can improve the VO estimation performance, but with some caveats. INS-aiding greatly improves rotation estimates, but the output is not better than simply using the rotation given by the Inertial Measurement Unit (IMU). The translation and scale estimation is not necessarily improved by rotational aiding as can be seen in the only slight improvement on the test set and detriment on sequence 10. Perhaps aiding with data related to the translation would be able to produce better overall results. For instance, the direction of translation calculated using Epipolar geometry could be used as input to the network. The approach explored in this thesis may help to reduce drift in VO solutions, but it needs to be used in conjunction with other sensors. This type of CNN aiding should be considered further to determine the types of input data and situations where it may be more useful.

5.2 Future Work

- Use pre-trained version of FlowNet [1], then fine-tune using transfer learning to see if results improve. This would give insight as to whether a model pre-trained for optical flow would be able to produce a better VO result.

- Use model architectures trained on the ImageNet database, then retrain to do VO and see how results change.

- Try other types of input aiding (e.g. rotation or normalized translation calculated with the essential matrix method, other sensor input, etc.).

- Use the CNN and image pair inputs to predict INS rotation errors instead of the camera's rotation. This may help to remove the INS errors improving the overall rotation predictions.
- Use a larger dataset to improve statistical variance.
- Try varying the size of the dense (fully-connected) layer for the aiding branch.
- Use a less drastic learning rate reduction factor (e.g. 0.8 instead of 0.1).
- Use multiple types of aiding concurrently.
- Try other normalization techniques (e.g. do not normalize rotation or translation features and scale rotation and translation Mean Squared Error (MSE) in the loss function instead).
- Remove rotation from images during pre-processing. This could be done by taking the true rotation between two images and dividing it in half to determine the amount that each image needs to be rotation to achieve the same orientation perspective. This half-rotation would then be multiplied by the normalized pixel coordinates to obtain the de-rotated images. Both images would then be converted back into pixel coordinates and the overlapping region of the two images could be determined. Because these perspective changes will reduce the field of view for each image, the frame rate would need to be high enough that there would still be adequate overlap between the two de-rotated images. Once these de-rotated images are obtained, the images could be cropped to a standard size such that the missing pixels produced by removing the rotation are not used.

- Remove the ground-based assumption that the camera is at a fixed height above the ground. In order to account for this change, a measurement of the camera's height above the ground could be given to the neural network as well.

Bibliography

1. A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox, "FlowNet: Learning Optical Flow with Convolutional Networks," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2758–2766, IEEE, 12 2015.

2. A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, IEEE, 2012.

3. "Camera Calibration and 3D Reconstruction." `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`, 2019. Accessed: 2020-02-12.

4. A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Sensor setup." `http://www.cvlibs.net/datasets/kitti/setup.php`. Accessed: 2020-03-03.

5. H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, pp. 133–135, 9 1981.

6. B. D. Scaramuzza and F. Fraundorfer, "Visual Odometry Part I: The First 30 Years and Fundamentals," *Robotics and Automation Magazine*, no. December, pp. 80–92, 2011.

7. F. Fraundorfer and D. Scaramuzza, "Visual odometry Part 2: Matching, Robustness, Optimization, and Applications," *Robotics and Automation Magazine*, no. June, pp. 78–90, 2012.

8. M. O. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, no. 1, 2016.

9. S. Wang, R. Clark, H. Wen, and N. Trigoni, "End-to-End, Sequence-to-Sequence Probabilistic Visual Odometry through Deep Neural Networks," *International Journal of Robotics Research*, 2016.

10. K. Konda and R. Memisevic, "Learning Visual Odometry with a Convolutional Network," *International Conference on Computer Vision Theory and Applications*, pp. 486–490, 2015.

11. P. Muller and A. Savakis, "Flowdometry: An Optical Flow and Deep Learning Based Approach to Visual Odometry," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 624–631, IEEE, 3 2017.

12. A. Valada, N. Radwan, and W. Burgard, "Deep Auxiliary Learning for Visual Localization and Odometry," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 6939–6946, 2018.

13. Y. Lin, Z. Liu, J. Huang, C. Wang, G. Du, J. Bai, S. Lian, and B. Huang, "Deep Global-Relative Networks for End-to-End 6-DoF Visual Localization and Odometry," *arXiv.org*, 2018.

14. R. Clark, S. Wang, H. Wen, A. Markham, and N. Trigoni, "VINet: Visual-Inertial Odometry as a Sequence-to-Sequence Learning Problem," in *AAAI Conference on Artificial Intelligence*, 2017.

15. L. Matthies and S. Shafer, "Error modeling in stereo navigation," *IEEE Journal on Robotics and Automation*, vol. 3, pp. 239–248, 6 1987.

16. D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," *Computer Vision and Pattern Recognition (CVPR), Proceedings of the 2004 IEEE Computer Society Conference on*, 2004.

17. D. Helmick, Yang Cheng, and S. Roumeliotis, "Path following using visual odometry for a Mars rover in high-slip environments," in *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720)*, vol. 2, pp. 772–789, IEEE, 2004.

18. A. Howard, "Real-time stereo visual odometry for autonomous ground vehicles," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3946–3952, IEEE, 9 2008.

19. L. Li, J. Lian, L. Guo, and R. Wang, "Visual odometry for planetary exploration rovers in sandy terrains," *International Journal of Advanced Robotic Systems*, vol. 10, pp. 1–7, 2013.

20. C. Chen, S. Rosa, Y. Miao, C. X. Lu, W. Wu, A. Markham, and N. Trigoni, "Selective Sensor Fusion for Neural Visual-Inertial Odometry," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10534–10543, 2019.

21. D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, 11 2004.

22. H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), vol. 3951, pp. 404–417, Berlin, Heidelberg: Springer Berlin Heidelberg, 6 2006.

23. S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: Binary Robust invariant scalable keypoints," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2548–2555, 2011.

24. K. Mikolajczyk and C. Schmid, "Scale & affine invariant interest point detectors," *International Journal of Computer Vision*, vol. 60, no. 1, pp. 63–86, 2004.

25. E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), pp. 430–443, Springer Berlin Heidelberg, 2006.

26. P. E. Forssén, "Maximally stable colour regions for recognition and matching," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 0–7, 2007.

27. D. Nistér and H. Stewénius, "Linear time maximally stable extremal regions," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5303 LNCS, no. PART 2, pp. 183–196, 2008.

28. M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary robust independent elementary features," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6314 LNCS, no. PART 4, pp. 778–792, 2010.

29. A. Alahi, R. Ortiz, and P. Vandergheynst, "FREAK: Fast retina keypoint," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 510–517, 2012.

30. E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2564–2571, 2011.

31. D. DeTone, T. Malisiewicz, and A. Rabinovich, "SuperPoint: Self-Supervised Interest Point Detection and Description," in *CVPR*, 2018.

32. P. Falez, P. Tirilly, I. M. Bilasco, P. Devienne, and P. Boulet, “Unsupervised Visual Feature Learning with Spike-timing-dependent Plasticity: How Far are we from Traditional Feature Learning Approaches?,” *arXiv.org*, 1 2019.

33. M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, pp. 381–395, 6 1981.

34. B. K. Horn and B. G. Schunck, “Determining optical flow,” *Artificial Intelligence*, vol. 17, pp. 185–203, 8 1981.

35. P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, “DeepFlow: Large displacement optical flow with deep matching,” *Proceedings of the IEEE International Conference on Computer Vision*, no. Section 2, pp. 1385–1392, 2013.

36. J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, “EpicFlow: Edge-preserving interpolation of correspondences for optical flow,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 1164–1172, 2015.

37. J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, “DeepMatching: Hierarchical Deformable Dense Matching,” *International Journal of Computer Vision*, vol. 120, pp. 300–323, 12 2016.

38. E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, “FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1647–1655, 2017.

39. B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox, “DeMoN: Depth and motion network for learning monocular stereo,” *Proceedings*

- 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, vol. 2017-Janua, pp. 5622–5631, 2017.

40. N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 4040–4048, 2016.

41. P. Agrawal, J. Carreira, and J. Malik, "Learning to see by moving," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, pp. 37–45, 2015.

42. A. Nicolai, R. Skeele, C. Eriksen, and G. A. Hollinger, "Deep Learning for Laser Based Odometry Estimation," *Science and Systems Conf. Workshop on Limits and Potentials of Deep Learning in Robotics*, 2016.

43. G. Costante, M. Mancini, P. Valigi, and T. A. Ciarfuglia, "Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation," *IEEE Robotics and Automation Letters*, vol. 1, pp. 18–25, 1 2016.

44. T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised Learning of Depth and Ego-Motion from Video," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2017-Janua, pp. 6612–6619, IEEE, 7 2017.

45. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 770–778, 2016.

46. N. Radwan, A. Valada, and W. Burgard, "VLocNet++: Deep Multitask Learning for Semantic Visual Localization and Odometry," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4407–4414, 2018.

47. S. Pillai and J. J. Leonard, "Towards visual ego-motion learning in robots," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 5533–5540, 2017.

48. T. Feng and D. Gu, "SGANVO: Unsupervised Deep Visual Odometry and Depth Estimation With Stacked Generative Adversarial Networks," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4431–4437, 2019.

49. G. Korn and T. Korn, *Mathematical Handbook for Scientists and Engineers: Definitions, Theorems, and Formulas for Reference and Review.* Dover Civil and Mechanical Engineering Series, Dover Publications, 2000. p. 178.

50. O. J. Woodman, "An introduction to inertial navigation," Tech. Rep. UCAM-CL-TR-696, University of Cambridge, Computer Laboratory, 2007.

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

52. F. Chollet, *Deep Learning with Python.* Shelter Island, NY: Manning, 1st ed., 2017.

53. Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural Networks: tricks of the trade*, vol. 75, pp. 9–48, Springer, 2012.

54. J. Brownlee, "How to use Data Scaling Improve Deep Learning Model Stability and Performance." `https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/`, 2019. Accessed: 2020-02-12.

55. NovAtel, Inc., *SPAN UIMU-HG1700 Datasheet*, May 2016. vers. 12.

56. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

57. F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

58. A. Kendall and R. Cipolla, "Geometric loss functions for camera pose regression with deep learning," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6555–6564, 2017.

59. T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

60. X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research*, vol. 9, pp. 249–256, 2010.

61. D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

Acronyms

2D two-dimensional. 2, 6, 7, 11, 16

3D three-dimensional. 2, 3, 6, 7, 11, 12, 16, 19

6-DoF six degrees of freedom. iv, viii, 3, 5, 17, 33, 38, 40, 41, 42, 48, 53, 54, 55, 56, 59, 61, 62, 64, 65, 1

ARW angular random walk. iv, 23, 24, 36, 37, 43, 64, 65, 66, 65, 68, 1

BRIEF Binary Robust Independent Elementary Features. 10

BRISK Binary Robust Invariant Scalable Keypoints. 10

CNN Convolutional Neural Network. iv, vi, ix, 1, 3, 4, 14, 15, 16, 17, 18, 19, 25, 38, 40, 41, 44, 45, 46, 48, 70, 71, 72, 1

DCM Direction Cosine Matrix. 19, 20, 22, 32

DL Deep Learning. iv, 1, 2, 3, 4, 5, 14, 15, 18, 19

DNN Deep Neural Network. 25

F2F frame-to-frame. iv, 1, 15, 25, 1

FAST Features from Accelerated Segment Test. 10

FLANN Fast Library for Approximate Nearest Neighbors. 11

FREAK Fast Retina Keypoint. 10

GAN Generative Adversarial Network. 19

GNSS Global Navigation Satellite System. 1

GPS Global Positioning System. 1, 5, 8, 25, 30

IMU Inertial Measurement Unit. viii, 2, 8, 18, 23, 25, 30, 36, 64, 65, 71

INS Inertial Navigation System. iv, vii, viii, ix, 2, 3, 4, 5, 24, 25, 26, 35, 36, 37, 39, 41, 43, 44, 45, 48, 51, 52, 54, 55, 56, 57, 58, 59, 62, 64, 65, 66, 68, 70, 71, 72, 1

LIDAR light detection and ranging. 1, 5, 16, 25

LSTM Long Short-Term Memory. 17, 18

MSE Mean Squared Error. 42, 48, 50, 72

MSER Maximally Stable Extremal Region. 10

OpenCV Open-Source Computer Vision. 29, 32

ORB Oriented FAST and Rotated BRIEF. 10

PnP perspective from n points. 12

RADAR radio detection and ranging. 1

RANSAC random sample consensus. 11

RCNN Recurrent Convolutional Neural Network. 1, 17, 18, 19, 38

ReLU Rectified Linear Unit. 38, 41

RMSE Root Mean Squared Error. iv, viii, ix, x, 37, 44, 45, 46, 48, 50, 51, 52, 53, 54, 59, 62

RNN Recurrent Neural Network. 2, 17, 18

SE(3) special Euclidean group. 5, 20, 21, 22, 30, 31

SIFT Scale Invariant Feature Transform. 10

SLAM Simultaneous Localization and Mapping. 13

SONAR Sound Navigation and Ranging. 1

SURF Speeded Up Robust Features. 10

VAE Variational Auto-Encoder. 19

VO Visual Odometry. iv, vii, viii, ix, x, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 24, 25, 28, 38, 39, 40, 41, 44, 45, 48, 51, 52, 53, 55, 56, 57, 58, 59, 62, 64, 65, 66, 70, 71, 1

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 26–03–2020	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Sept 2018 — Mar 2020

4. TITLE AND SUBTITLE Improved Ground-Based Monocular Visual Odometry Estimation using Inertially-Aided Convolutional Neural Networks	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Watson, Josiah D, Mr.	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, Graduate School of Engineering and Management (AFIT/EN), 2950 Hobson Way, WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT-ENG-MS-20-M-072
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally Left Blank	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution Statement A:
Approved for Public Release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

While Convolutional Neural Networks (CNNs) can estimate frame-to-frame (F2F) motion even with monocular images, additional inputs can improve Visual Odometry (VO) predictions. In this thesis, a FlowNetS-based [1] CNN architecture estimates VO using sequential images from the KITTI Odometry dataset [2]. For each of three output types (full six degrees of freedom (6-DoF), Cartesian translation, and transitional scale), a baseline network with only image pair input is compared with a nearly identical architecture that is also given an additional rotation estimate such as from an Inertial Navigation System (INS). The inertially-aided networks show an order of magnitude improvement over the baseline when predicting rotation, but the aided rotation predictions are still worse than the input rotations. Translation predictions are not necessarily helped either. A full-trajectory analysis gives similar results. The INS-aided neural networks are also tested for sensitivity to angular random walk (ARW) and bias errors in the sensor measurements.

15. SUBJECT TERMS

Visual Odometry, Monocular, Convolutional Neural Networks, Deep Learning, Navigation

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Captain Joseph A. Curro, AFIT/ENG
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U	UU	96	(937) 255-3636, ext 4620; joseph.curro@afit.edu