A Deep Learning Method for GPS/GNSS Independent, Vision-Based Navigation and State Estimation of Class 8 Vehicles

by

Tahn Thawainin

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
August 8, 2026

Keywords: Deep Learning, Vision-based Navigation, GNSS Independent Navigation, State Estimation, Vehicle Dynamics

Approved by

David Bevly, Bill and Lana McNair Professor
Scott Martin, Professor of Mechanical Engineering
Brendon Allen, Professor of Mechanical Engineering

Abstract

This thesis develops a deep learning based method for vision-based navigation and state estimation of class 8 vehicles without relaying on GPS/GNSS. The novel deep learning system is compared to a vehicle-model-based system and a traditional inertial navigation system (INS). For the model-based method, a dead reckoning navigation filter is developed around a tractor-trailer dynamics model. To address the limitations of the model-based system, a deep learning neural network was designed as a data-driven dead reckoning and state estimation system.

First, a detailed derivation of a tractor-trailer bicycle model is given. Then, a navigation extended Kalman filter is designed around the vehicle states. Given that a unique challenge for autonomous trucking is high model variability from the trailer unit, accurate parameter knowledge of the vehicle model cannot be assumed. To address this issue, a deep learning neural network is designed to ingest camera and inertial sensor data to produce relative motion and trailer hitch angle predictions. The cameras are mounted on the rear-facing mirrors to monitor the trailer and the vehicle's motion over time. The proposed deep learning network is inspired from classical visual inertial odometry systems, but does not require information about the system or sensor models. The proposed network model is a transformer based model that uses a combination of convolutional neural networks, vision transformers, self-attention, and cross-attention. It is shown through various Monte Carlo simulations that the proposed deep learning system often outperforms the model-based system. Significant benefits are especially evident in cases where the vehicle model is not well known which causes the traditional model-based solutions to degrade. Experiments conducted further validate the benefits of the deep learning approach showing consistent improvements over the model-based method in positioning, orientation, and hitch angle accuracy, even when the model is known accurately. Furthermore, it is shown that the deep

learning network is able to maintain accurate positioning for significantly longer periods of time than the model-based system and the INS which is highly valuable for any dead reckoning system.

Artificial Intelligence (AI) Use Disclosure Statement

In the preparation of this thesis / dissertation, the following Artificial Intelligence (AI) tools were used: Claude. These tools were used primarily to generate test code and aid with debugging code. The author acknowledges full responsibility for the intellectual content of this work and has ensured that all AI-assisted sections have been reviewed and revised for accuracy and appropriate academic style. All AI-generated content was reviewed and validated for relevance, appropriateness, and accuracy before incorporation into the final document to maintain scholarly integrity of this research.

Digital Accessibility Disclosure Statement

In the preparation of this thesis / dissertation, the following digital accessibility tools were used to ensure this document complies with federal requirements: Adobe Acrobat Pro. The author acknowledges full responsibility for the intellectual content of this work and has made a good faith effort to comply with digital accessibility requirements in publishing, wherein the nature of the content does not significantly change in order to do so. Furthermore, all content has been reviewed and revised to meet these requirements prior to final publication

Acknowledgments

I would first like to thank my family. Especially my mother, for her incomprehensible sacrifices and unwavering support. I would also like to thank my brother, who has always been my greatest enemy and my closest friend. Without them, none of this would have been possible.

My experience with the GAVLab has been nothing short of enlightening and challenging. I owe a huge thanks to Dr. Dave Bevly and Dr. Scott Martin for their instruction and guidance during my time in the GAVLab. I would also like to thank Dr. Anh Nguyen whose engaging Deep Learning lectures inspired a large portion of this thesis. I would also like to acknowledge and thank my fellow lab members, whom several have become great colleagues and friends. Specifically, I owe a thanks to Tyler Flegel for being an incredibly helpful resource and for always knowing where every item in the lab is. I'd like to thank Evan Ellison and Kyle Thompson for their help with the experimental setup for this work, and for being much more capable with hardware than I am. I would also like to thank Sam Morgan for convincing me to move to Auburn to split rent with him and to pursue graduate school.

Finally, I owe a special thanks to my girlfriend, Jesse, who is the reason why I remained assiduous in completing this thesis, and the reason why it took longer than anticipated.

Table of Contents

Abstract 2

Artificial Intelligence (AI) Use Disclosure Statement 4

Digital Accessibility Disclosure Statement 5

Acknowledgments 6

List of Tables 10

List of Figures 11

1 Introduction 17

1.1 Background and Motivation 17

1.2 Prior Art 18

1.2.1 Inertial Navigation 18

1.2.2 Ground Vehicle Based Navigation 19

1.2.3 Perception Based Navigation 20

1.2.4 Trailer Hitch Angle Estimation 21

1.3 Contributions 22

1.4 Thesis Outline 22

2 GNSS-Independent Navigation Technical Background 24

2.1 Coordinate Frames 25

2.1.1 Earth-Centered Earth-Fixed and LLA Frames 25

2.1.2 Local Tangent Frame 25

2.1.3 Body Frame 26

2.1.4 Frame Transformations 27

2.2 Inertial Navigation System 29
2.2.1 IMU Mechanization 29
2.3 Ground Vehicle Odometry 33
2.4 Classical Visual Inertial Odometry 34

3 Tractor-Trailer Model-Based Navigation 38
3.1 Tractor-Trailer Lateral Bicycle Model 38
3.1.1 Lateral Tire Forces 41
3.1.2 Lagrangian Derivation 42
3.1.3 State-Space Representation 44
3.2 Navigation Extended Kalman Filter Design 48
3.2.1 State Propagation 49
3.2.2 Measurement Correction 52
3.2.3 Zero Velocity Updates 53
3.2.4 Limitations 54

4 Deep Learning Background 56
4.1 Deep Learning Background 56
4.1.1 Single Neuron Model 56
4.1.2 Backpropagation 58
4.1.3 Optimizers 61
4.1.4 Schedulers 62
4.1.5 Additional Design Techniques 63
4.2 Core Network Architectures 65
4.2.1 Multilayer Perceptrons 65
4.2.2 Convolutional Neural Networks 67
4.2.3 Transformers 69

5 Deep Learning, Vision-Based Dead Reckoning Network 75
5.1 Deep Learning VIO Network Design 76
5.2 Training Paradigm 82
5.3 Inference 83

6 Simulation and Experimental Results 85
6.1 Simulations 85
6.1.1 Simulation Data Generation 85
6.1.2 Simulation DL-VIO Parameters and Training Results 88
6.1.3 Monte Carlo Simulations 90
6.2 Experimental Results 102
6.2.1 Experimental Setup 102
6.2.2 Experimental DL-VIO Parameters and Training Results 106
6.2.3 Experimental Results on Testing Sequences 108

7 Conclusions and Future Works 118
7.1 Conclusions 118
7.2 Future Works 119

References 122

A Experimental Results on Training Sequences 133

B Swapped Training and Testing Results 140
B.1 Results on Testing Sequences 141
B.2 Results on Training Sequences 148

List of Tables

3.1 Tractor-Trailer Bicycle Model Parameters 47

5.1 Spatial and Temporal Attention Descriptions 80

6.1 Simulation DL-VIO Model and Training Parameters 88

6.2 Vehicle Parameter True Values and Ranges 91

6.3 Simulated IMU Parameters 92

6.4 Experimental DL-VIO Model and Training Parameters 107

List of Figures

2.1 Position Dead Reckoning 24
2.2 Earth-Centered Earth-Fixed (ECEF) and Local Tangent Coordinate Systems 26
2.3 Right-Handed Body Frame 27
2.4 IMU Mechanization Process 30
2.5 Classical Bicycle Free-Body Diagram 33
2.6 Visual-Inertial Odometry Problem Space 35
2.7 2D Pixel Point To 3D World Frame Pipeline 36
3.1 Tractor-Trailer Bicycle Model Free-Body Diagram 40
3.2 Kinematic Slip Angle Diagrams 41
3.3 Extended Kalman Filter Algorithm 49
4.1 Single Neuron Model 57
4.2 ReLU Activation and Its Variants 58
4.3 Gradients in a Single Neuron 74
4.4 Common Learning Rate Schedulers 63
4.5 General MLP Architecture 66
4.6 General CNN Architecture 67
4.7 Transformer Encoder Block 70
4.8 Cross-Attention Pipeline 73
5.1 Proposed Sensor Suite 76
5.2 The Tractor-Trailer VIO Network Architecture. 77
6.1 Simulation Application Setup 86

6.2 Simulated Camera Data Examples 87

6.3 Time and Setting Variations. **Left:** Training Set. **Right:** Testing Set. . . . 88

6.4 Simulation VIO Learning Rate 89

6.5 The training and validation RMSEs of the VIO network's outputs on simulation data. 90

6.6 Low dynamic sequence North and East trajectory. 93

6.7 Low dynamic sequence steering angle, longitudinal velocity, yaw Angle, and hitch Angle 93

6.8 Low dynamics sequence with high model uncertainty: North and East position errors 94

6.9 Low dynamics sequence with high model uncertainty: EKF and DL-VIO North and East Position Errors 95

6.10 Low dynamics sequence with high model uncertainty: Yaw and Hitch Angle Errors 95

6.11 Low dynamics sequence with low model uncertainty: North and East position errors 96

6.12 Low dynamics sequence with low model uncertainty: EKF and DL-VIO North and East Position Errors 96

6.13 Low dynamics sequence with low model uncertainty: Yaw and Hitch Angle Errors 97

6.14 High dynamic sequence North and East Trajectory 98

6.15 High dynamic sequence steering angle, longitudinal velocity, yaw angle, and hitch angle 98

6.16 High dynamics sequence with high model uncertainty: North and East Position Errors. 99

6.17 High dynamics sequence with high model uncertainty: EKF and DL-VIO North and East Position Errors. 100

6.18 High dynamics sequence with high model uncertainty: Yaw and Hitch Angle Errors. 100

6.19 High dynamics sequence with low model uncertainty: North and East Position Errors. 101

6.20 High dynamics sequence with low model uncertainty: EKF and DL-VIO North and East Position Errors. 101

6.21 High dynamics sequence with low model uncertainty: Yaw and Hitch Angle Errors. 102

6.22 GAVLab Class 8 Test Vehicle 103

6.23 Test Vehicle Equipment Setup 104

6.24 The six experimental sequences across Auburn and Opelika, AL. 105

6.25 Examples of the collected camera data across each sequence. 105

6.26 The experimental training and testing sequences for the DL-VIO network. 107

6.27 The training and validation RMSEs of the DL-VIO network's outputs on experimental data. 108

6.28 The position and position errors of the model-based EKF and DL-VIO network over Sequence 2. 110

6.29 The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 2. 110

6.30 The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 2. 111

6.31 Error Statistics for Sequence 2 111

6.32 Drift Times for Sequence 2 111

6.33 The position and position errors of the model-based EKF and DL-VIO network over Sequence 4. 113

6.34 The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 4. 113

6.35 The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 4. 114

6.36 Error Statistics for Sequence 4 114

6.37 Drift Times for Sequence 4 114

6.38 The position and position errors of the model-based EKF and DL-VIO network over Sequence 6. 116

6.39 The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 6. 116

6.40 The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 6. 117

6.41 Error Statistics forSequence 6 117

6.42 Drift Times forSequence 6 117

A.1 The position and position errors of the model-based EKF and DL-VIO network over Sequence 1. 133

A.2 The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 1. 134

A.3 The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 1. 134

A.4 Error Statistics for Sequence 1 135

A.5 Drift Times for Sequence 1 135

A.6 The position and position errors of the model-based EKF and DL-VIO network over Sequence 3. 136

A.7 The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 3. 136

A.8 The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 3. 137

A.9 Error Statistics for Sequence 3 137

A.10 Drift Times for Sequence 3 137

A.11 The position and position errors of the model-based EKF and DL-VIO network over Sequence 5 138

A.12 The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 5. 138

A.13 The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 5. 139

A.14 Error Statistics forSequence 5 139

A.15 Drift Times forSequence 5 139

B.1 The position and position errors of the model-based EKF and DL-VIO network over Sequence 1. 141

B.2 The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 1. 142

B.3 The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 1. 142

B.4 Error Statistics for Sequence 1 143

B.5 Drift Times for Sequence 1 143

B.6 The position and position errors of the model-based EKF and DL-VIO network over Sequence 3. 144

B.7 The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 3. 144

B.8 The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 3. 145

B.9 Error Statistics for Sequence 3 145

B.10 Drift Times for Sequence 3 145

B.11 The position and position errors of the model-based EKF and DL-VIO network over Sequence 5. 146

B.12 The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 5. 146

B.13 The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 5. 147

B.14 Error Statistics for Sequence 5 147

B.15 Drift Times for Sequence 5 147

B.16 The position and position errors of the model-based EKF and DL-VIO network over Sequence 1. 148

B.17 The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 2. 149

B.18 The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 2. 149

B.19 Error Statistics for Sequence 2 150

B.20 Drift Times for Sequence 2 150

B.21 The position and position errors of the model-based EKF and DL-VIO network over Sequence 4. 151

B.22 The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 4. 151

B.23 The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 4. 152

B.24 Error Statistics forSequence 4 152

B.25 Drift Times forSequence 4 152

B.26 The position and position errors of the model-based EKF and DL-VIO network over Sequence 6. 153

B.27 The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 6. 153

B.28 The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 6. 154

B.29 Error Statistics forSequence 6 154

B.30 Drift Times forSequence 6 154

Chapter 1

Introduction

1.1 Background and Motivation

Heavy semi-trucks account for more miles on US roads than any other vehicle type. A survey conducted in 2024 by the Department of Energy (DOE) indicates that class 8 trucks account for nearly 30% of the average annual miles traveled among major vehicle classes [1]. Furthermore, an analysis on 2023 traffic accidents by the National Highway Traffic Safety Administration (NHTSA) reports that while large trucks are less involved in fatal traffic accidents (9% of all fatal crashes), over 80% of these fatalities were occupants in other vehicles or pedestrians [2]. To this end, autonomous technologies that aim to improve safety systems for class 8 vehicles are highly valuable.

Autonomous trucking presents challenges that are unique from other autonomous ground vehicle applications. In particular, the trailer unit introduces additional dynamics and high model variability. Trailers often change parameters such as payload mass, payload distribution, hitch point location, tire stiffness, and axle locations. Trailers may be interchanged between operations making it impractical to equip them with sensors. As a result, navigation and control systems designed for tractor-trailers must be robust to unknown model configurations. Furthermore, class 8 vehicles must operate in a variety of urban and rural environments. This is problematic since over reliance on Global Navigation Satellite Systems (GNSS) such as the Global Positioning System (GPS) can cause critical positioning and navigation failures in contested environments. Robust autonomous trucking systems must have reliable navigation methods in the absence of these signals.

The motivation for this thesis is to address the above challenges from a navigation and state estimation context. Specifically, this research aims are to address GNSS independent navigation, and tractor-trailer state estimation. Although navigation and state estimation are closely related, the addition of a trailer unit requires further addressing. This thesis seeks to develop and compare two systems. The first is a classical filtering system that is designed around a tractor-trailer dynamics model. The second is a data-driven deep learning network designed to ingest inertial and image data for GNSS independent navigation and trailer hitch estimation.

1.2 Prior Art

GNSS-alternative navigation is a very broad area of research, as most modern position, navigation and timing (PNT) systems rely on satellite navigation. As such, several different methods that leverage a variety of sensor alternatives have been explored. Additionally, most navigation systems designed for a tractor unit will not account for the trailer unit without additional sensors. To recover trailer information, this thesis focuses on estimating the trailer hitch angle, which is the relative angle between the tractor and trailer headings. This section will discuss prior research for GNSS-independent navigation in the areas of inertial, ground vehicle, and perception navigation, and independently, trailer hitch angle estimation.

1.2.1 Inertial Navigation

The traditional “Gold Standard” for navigation systems applies fusion of GNSS and inertial navigation systems (INS) referred to as GNSS/INS fusion. The benefits and drawbacks of each system are complementary. GNSS provide long-term position solutions but have relatively low bandwidth and are prone to outages, whereas INS have high bandwidth with short-term accuracy but the solution degrades over time [3]. Detailed coverage of GNSS/INS systems are omitted here, but several fusion methods are practiced. Most of which generally involve recovering position, velocity, and attitude (PVA) from an error state Kalman filter (KF) or extended Kalman filter

(EKF) where the INS propagates states through time and the GNSS provides aiding measurements when available [3–7]. Naturally, in GNSS denied scenarios, the INS will provide a stand-alone navigation solution. However, the limitation with stand-alone INS is that it is a purely integrated or dead-reckoned solution that degrades over time due to sensor errors. Moreover, INS that use low-cost inertial measurement units (IMUs) typically have higher solution errors. For ground vehicles, lower quality, micro-electro-mechanical systems (MEMS) IMUs are often deployed to conserve costs, and driving induced vibrations, dynamics, or disturbances could further compromise an INS solution [8, 9].

1.2.2 Ground Vehicle Based Navigation

For ground vehicle applications, vehicle dynamics can be exploited to improve navigation systems. This process typically involves computing dynamic states or constraints from a vehicle model to replace or aid an INS [10–12]. The main challenges with vehicle model based systems are model parameter uncertainty and unmodeled disturbances which can lead to large errors if unaddressed. Nevertheless, a well configured vehicle-based system can be much more accurate than stand-alone INS. Articulating tractor-trailer models are much more complicated than typical ground vehicle models. However, lateral dynamic models for tractor-trailers have been present for decades. Modern lateral models are heavily akin to classical bicycle models used for single unit vehicles. Jindra developed a linear state-space model for a tractor and double trailer combination in 1968 [13]. This was adopted as the standard for trailer stability analysis and control, and extended by other researchers for various applications [14]. Similar principles and assumptions from Jindra's model were adopted in more recent works by Luijten in 2010 [15] and Wolfe in 2014 [16] which aim to model more typical systems with a single articulated trailer. Wolfe also fused the dynamics model with IMU measurements which greatly outperforms the stand-alone model for navigation without GNSS.

1.2.3 Perception Based Navigation

Perception sensors are also widely used for navigation. They allow for scene understanding and feature extraction capabilities that INS lack at the cost of additional computational complexity. GNSS fusion with LiDARs and cameras have been explored in [17] and [18]. Perception-based navigation algorithms that are completely independent of GNSS are also widely used. Several algorithms use a variety of perception sensors such as monocular, stereo, time-of-flight (ToF), event-based, and thermal cameras, LiDARs, or camera-LiDAR hybrids [19]. Most modern perception-based navigation systems can be categorized as Simultaneous Localization and Mapping (SLAM) or Visual Odometry/Visual-Inertial Odometry (VO/VIO). SLAM enables both navigation and scene mapping capabilities from detected and tracked features over time [20–23]. Upon revisiting points in the mapped environment, loop closure techniques can be used to correct accumulated drift. While SLAM is very powerful, it is often computationally expensive to maintain a 3D reconstruction of a scene, though modern SLAM algorithms have shown viable real-time performance [22]. However, for extended, outdoor ground vehicle applications, loop closures may be extremely sparse or nonexistent entirely, and scene reconstruction becomes added overhead. Alternatively, VO and VIO are the foundation of most modern SLAM, but focus on navigation without mapping [24–26]. The principal difference between VO and VIO is that VIO additionally fuse INS with the vision system and often outperforms VO alone. Research for SLAM and VO/VIO algorithms using neural networks have also been explored and show promising results [27–30]. Neural network approaches are able to combine feature extraction and sensor fusion all within a single model making it very powerful, although the literature for such approaches is sparse compared to classical SLAM and VO/VIO. It is important to mention that both SLAM and VO/VIO are still dead reckoning systems since they solve for relative motion. However, they are often much more capable than low-cost IMUs at the expense of computational complexity.

1.2.4 Trailer Hitch Angle Estimation

Traditional navigation systems designed for a tractor unit are unlikely able to recover any information about the trailer without additional sensors on the trailer unit. However, equipping sensors to a trailer is not practical for maintaining trailer interchangeability. As such, independent methods to estimate the trailer articulation angle are required, and prior research have explored several methods to do so. When model parameters are known well, model-based methods are largely successful for estimating hitch angle. Kinematic and dynamic tractor-trailer models coupled with state observers have been used to estimate trailer articulation [16,31–33]. However, model-based methods are heavily dependent on parameter knowledge, and ever-changing trailer parameters make using such methods difficult.

Alternatives to model-based methods are stand-alone, sensor-based algorithms, and multiple sensor suites have been investigated to estimate trailer hitch angle. Ultrasonic sensors were used in [34, 35], and RADARs in [36, 37]. Although both sensors are viable options, cameras can be much more cost-effective and apply to several other perception tasks [38, 39]. Several works have explored using cameras algorithms, including both classical and neural network approaches. Non-neural-network approaches such as template matching and trailer fiducial tags have been used in [40] and [41], respectively. However, template matching performance is limited by its database and trailer fiducial tags require markers on the trailer. Works that leverage neural networks are detailed in [42–45]. The methods used in [42] and [43] apply convolutional neural networks (CNNs) to detect a trailer face and other components of the vehicle but use classical image processing to compute articulation angle. The approach in [44] uses a unified CNN to estimate hitch angle directly from simulated RGB images, combining feature extraction and hitch angle computation entirely from a neural network. The research conducted in [45] extends this idea and investigates multiple pretrained CNN models on much more diverse simulated data, showing each

neural network's capability against real-world challenges such as lighting, shadows, occlusions, time diversity, and scene diversity.

1.3 Contributions

The present research aims to address several challenges of autonomous trucking through the design of a unified, navigation and state estimation algorithm for class 8 vehicles. This thesis seeks to design, evaluate, and compare a system designed around a tractor-trailer dynamic model, and a data-driven deep learning network without reliance on GNSS. The contributions in this thesis are:

- The use of a limited sensor suite consisting of the SAE J1939 CAN bus sensors, a tractor mounted IMU, and mirror-mounted, rear-facing monocular cameras for GNSS independent navigation and state estimation.
- The design and evaluation of a navigation Kalman filter built around a modern tractor-trailer dynamics model.
- The design and evaluation of a transformer based neural network trained for visual-inertial navigation and trailer hitch angle estimation.
- A comparison of the model-based and deep learning methods for dead reckoned positioning, orientation, and hitch angle estimation.
- A validation of the proposed deep learning model through various simulation studies and through experimental data.

1.4 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 discusses the technical background of relevant GNSS-independent navigation methods that the proposed algorithms are based upon. Chapter 3 introduces the tractor-trailer model derivation and the model-based navigation filter design. Chapter 4 provides a background of deep

learning and the core architectures used. Chapter 5 discusses the navigation neural network design and training procedure. Chapter 6 presents the simulation and experimental results which evaluates and compares the performance of the model-based system and the deep learning system. Finally, Chapter 7 closes with conclusions and future work.

Chapter 2

GNSS-Independent Navigation Technical Background

A common method for maintaining a GNSS-independent position solution is dead reckoning whereby a position solution is propagated over time using information about a unit's motion [3]. Although there are several techniques to dead reckon a position, the general process involves using motion information to estimate a change in position from a known or estimated prior position. A limitation held by all dead reckoning methods is that inherent integration of unknown errors and noise causes the solution errors to grow over time, which is often referred to as solution or position drift. A simple schematic of position dead reckoning is illustrated in Figure 2.1. This chapter discusses the basic foundations of three dead reckoning techniques: an inertial navigation system, vehicle model based odometry, and a classical visual-inertial odometry.

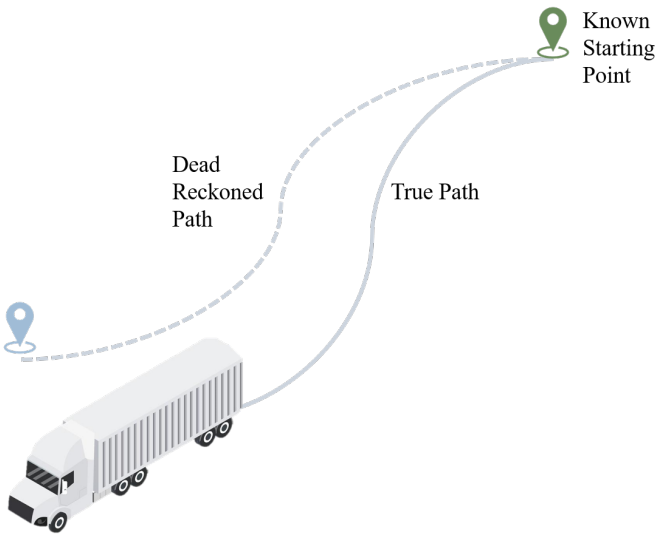


Figure 2.1: Position Dead Reckoning

2.1 Coordinate Frames

Before discussing a navigation approach, a fundamental understanding of coordinate frames is required. For navigation applications, multiple viable coordinate frames are available and often determined by the specific application or a user's preference. A brief discussion of these frames and their relationships to each other and the Earth are provided in this section. Note that for this work, the most relevant coordinate frames are a local tangent or navigation frame, and the body frame.

2.1.1 Earth-Centered Earth-Fixed and LLA Frames

The Earth-Centered Earth-Fixed (ECEF) frame is a common frame to define a navigation solution relative to the Earth [3]. An illustration of the ECEF frame is shown in Figure 2.2 denoted by axes (x, y, z). The ECEF frame is a Cartesian coordinate system with its origin fixed to the Earth's center. The axes of the ECEF frame are fixed to the Earth where the z-axis lies along the Earth's axis of rotation (i.e., from the center to the true North Pole), the x-axis points from the center along the prime meridian (i.e., zero-degree longitude), and the y-axis completes the orthogonal set (i.e., points from the center to the 90-degree east meridian) [3]. In Figure 2.2, geodetic latitude is denoted by Φ and longitude by λ. This is also a commonly used Earth coordinate frame but is instead a curvilinear coordinate system described by latitude, longitude, and altitude (LLA).

2.1.2 Local Tangent Frame

The local tangent frame is a tangential frame fixed to Earth's surface and is used in this work as the primary reference frame for navigation. The point of origin is arbitrary and may represent the coordinate where the system loses GPS/GNSS positioning and begins to dead reckon. An illustration of a local tangent frame is shown in Figure 2.2 denoted by East, North, and Up (ENU) axes. An alternative axis orientation is North,

East, Down (NED), and this selection is typically made based upon application, sensor orientation, or user preference.

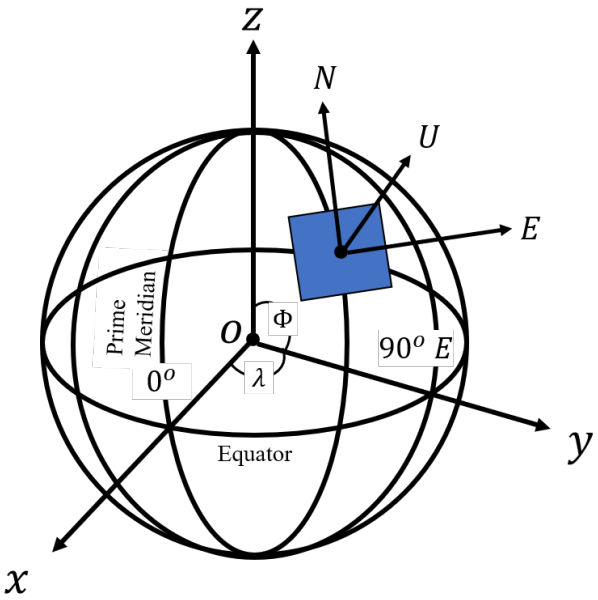


Figure 2.2: Earth-Centered Earth-Fixed (ECEF) and Local Tangent Coordinate Systems

2.1.3 Body Frame

The body frame represents a Cartesian coordinate system that is fixed to a body of interest such as a vehicle or sensor body. For ground vehicle applications, the origin is typically set at the center of gravity (CG) of the vehicle, and axes orientation can vary. This work uses a right-handed system with x as positive forward, y as positive left, and z as positive up. For angular motion, x is the rolling axis, y is the pitching axis, z is the yawing axis with positive counterclockwise rotation. A representation is shown in Figure 2.3.

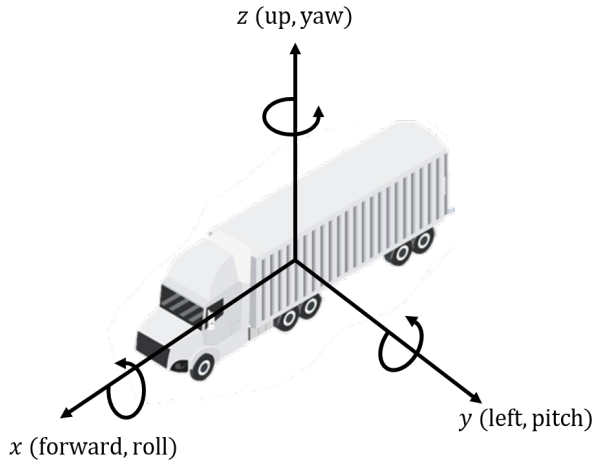


Figure 2.3: Right-Handed Body Frame

2.1.4 Frame Transformations

Information resolving in one coordinate frame can be expressed in another using rotation matrices. A rotation matrix is a 3×3 matrix which relates Cartesian position, velocity, acceleration, and angular rate from one frame to another as shown in Equation (2.1),

$$\mathbf{x}^a = C_b^a \mathbf{x}^b, \quad \mathbf{x} \in r, v, a, \omega \in \mathbb{R}^3 \tag{2.1}$$

where a and b represent arbitrary coordinate frames. Reconversion (i.e., x^a to x^b) can be done by a similar operation using the transpose of the rotation matrix such that $C_a^b = C_b^{aT}$.

This research is concerned with three frame transformations: Body frame (b) to local tangent frame (t), local tangent frame to ECEF frame (e), and ECEF frame to LLA. Again, transformations between the body and the local tangent frames are most important for this work. Irrespectively, the other two conversions are shown as a method to retrieve a position relative to an Earth reference system.

The body frame to tangent frame rotation matrix can be formulated using three successive elementary rotations as a function of the body's roll (ϕ), pitch (θ), and yaw

(ψ). The resulting rotation matrix relating the body frame to a local NED frame is given in Equation (2.2).

$$C_b^t = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{2.2}$$

Next, the rotation matrix relating the tangent frame to the ECEF frame is given in Equation (2.3).

$$C_t^e = \begin{bmatrix} -\sin\Phi_o\cos\lambda_o & -\sin\lambda_o & -\cos\Phi_o\cos\lambda_o \\ -\sin\Phi_o\sin\lambda_o & \cos\lambda_o & -\cos\Phi_o\sin\lambda_o \\ \cos\Phi_o & 0 & -\sin\Phi_o \end{bmatrix} \tag{2.3}$$

The matrix is a function of a reference or origin latitude, Φ_o, and longitude, λ_o of the body which can represent the last available coordinate received from GNSS. This conversion can be used to project tangent frame states to and from an Earth referenced system, which is useful for geolocating positions resolved in an arbitrary tangent plane. Finally, conversions to and from the ECEF frame and LLA are briefly discussed. Unlike the previous conversions, LLA descsribes a curvilinear position instead of a typical Cartesian coordinate system. A rotation matrix therefore cannot describe the relationship between ECEF and LLA. However, conversions can be solved for trigonometrically. ECEF position can be calculated from LLA using Equation (2.4),

$$\begin{aligned} x &= (R_E(\Phi) + h)\cos(\Phi)\cos(\lambda) \\ y &= (R_E(\Phi) + h)\cos(\Phi)\sin(\lambda) \\ z &= ((1-e^2)R_E + h)\sin\Phi \end{aligned} \tag{2.4}$$

where h is equivalent to the body's altitude, e is the eccentricity of the Earth, and $R_E(\Phi)$ is the transverse radius of curvature as a function of latitude, Φ. Their formal equations are not included here but can be accessed in [3] and [46], along with other

World Geodetic System 1984 (WGS 84) standards. Converting from ECEF to LLA is more difficult and care must be taken to avoid divergence. An iterative approach presented in [3] is shown in Equation (2.5).

$$\begin{aligned} \Phi &= \sin^{-1}\left(\frac{z}{(1-e^2)R_E(\Phi)+h}\right) \\ \lambda &= \tan^{-1}\left(\frac{y}{x}\right) \\ h &= \frac{\sqrt{x^2+y^2}}{\cos(\Phi)} - R_E(\Phi) \end{aligned} \tag{2.5}$$

Note that for longitude, $\tan^{-1}$ refers to a four-quadrant arctangent [3]. Additionally, since R_E is a function of latitude, latitude and height must be solved iteratively with $R_E = R_0$ (WGS84 defined equatorial radius) on the first iteration. Alternate closed-form solutions are presented in [47] and [48].

2.2 Inertial Navigation System

An inertial navigation system (INS) involves using the motion of a body described by measurements from an inertial measurement unit (IMU) to calculate the body's orientation, velocity, and position. An IMU is typically composed of a three-axis accelerometer which measures 3-DOF specific force, and a three-axis gyroscope which measures 3-DOF angular velocities in the sensor's body frame. These measurements are used in a process called IMU mechanization which is discussed in detail in this section.

2.2.1 IMU Mechanization

IMU mechanization refers to processing measurements received from an inertial measurement unit to calculate the attitude, velocity, and position of the sensor body relative to a tangential or global frame. The process to mechanize an IMU is distinct depending on the resolving frame. Again, the main reference frame targeted for this thesis is the local tangent frame, thus the appropriate equations are defined here.

Additional equations for IMU mechanization resolving in different coordinate frames may be found in [3]. A diagram of the mechanization process is shown in Figure 2.4.

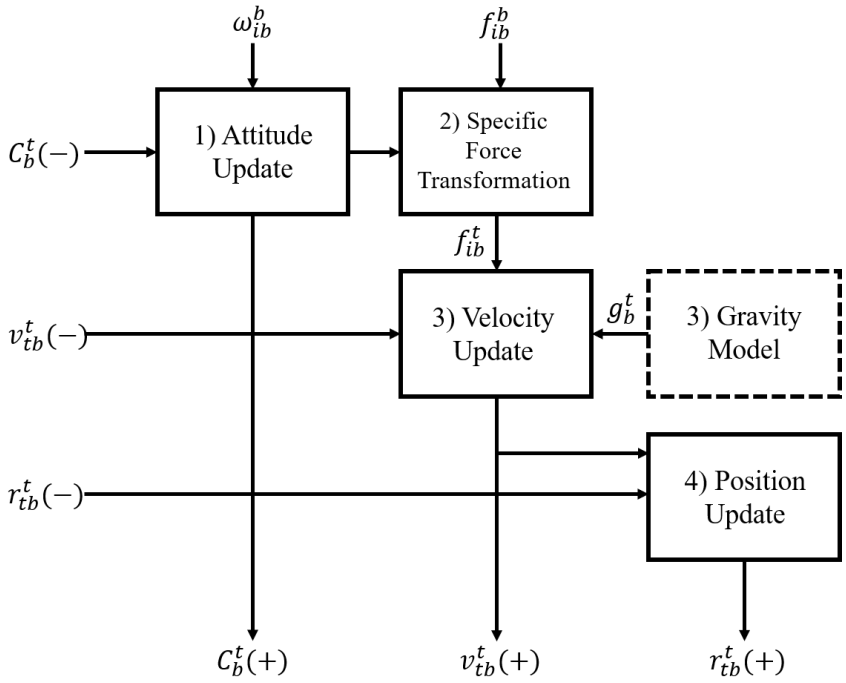


Figure 2.4: IMU Mechanization Process

In the figure, ω_{ib}^{b} and f_{ib}^{b} represent a vector of angular rate measurements and a vector of specific force measurements in the IMU's body frame. In this thesis, the IMU is considered rigidly mounted to the tractor's body frame with their appropriate axes aligned. As such, ω_{ib}^{b} and f_{ib}^{b} measurements are considered valid for describing the tractor's body frame motion. Furthermore, the previous position, velocity, and attitude (PVA) (i.e., $C_b^t(-)$, $v_{tb}^{t}(-)$, and $r_{tb}^{t}(-)$) are required to estimate the current PVA. Regarding the initialization process for this thesis, the initial PVA is considered known from the last available GNSS or INS/GNSS solution.

Attitude Update

The first step to mechanization is the attitude update where the angular rate measurements, ω_{ib}^{b}, are used to calculate the current attitude rotation matrix, $C_b^t(+)$. The discretized approximation of this step is given in Equation (2.6),

$$C_b^t(+) \approx C_b^t(-)(\mathbf{I}_3 + \mathbf{\Omega}_{ib}^b \Delta t) - \mathbf{\Omega}_{ie}^t C_b^t(-)\Delta t \tag{2.6}$$

where $\mathbf{\Omega}_{ib}^b$ is the skew-symmetric matrix of the angular rates given in Equation (2.7).

$$\mathbf{\Omega}_{ib}^b = \mathsf{Skew}(\omega_{ib}^b) = \begin{bmatrix} 0 & -\omega_{ib,z}^b & \omega_{ib,y}^b \\ \omega_{ib,z}^b & 0 & -\omega_{ib,x}^b \\ -\omega_{ib,y}^b & \omega_{ib,x}^b & 0 \end{bmatrix} \tag{2.7}$$

$\mathbf{\Omega}_{ie}^t$ is the skew-symmetric matrix of the of the Earth-rotation rate resolved in the tangent frame given in Equation (2.8).

$$\mathbf{\Omega}_{ie}^t = \mathsf{Skew}(\omega_{ie} \begin{bmatrix} \cos(\phi) \\ 0 \\ -\sin(\phi) \end{bmatrix}) = \omega_{ie} * \begin{bmatrix} 0 & \sin(\phi) & 0 \\ -\sin(\phi) & 0 & -\cos(\phi) \\ 0 & \cos(\phi) & 0 \end{bmatrix} \tag{2.8}$$

The above equation is a function of the Earth's rotation rate, $\omega_{ie} = 7.2921 \times 10^{-5} \frac{rad}{s}$. Additionally, it is a function of latitude, ϕ, which can be converted from the calculated tangent frame position. For simple applications, especially within local tangent frame (e.g., indoor navigation, simple simulations), the contributions from the Earth's rotation rate are negligible. Thus, an alternative, simplified attitude update follows Equation (2.9).

$$C_b^t(+) \approx C_b^t(-)(\mathbf{I}_3 + \mathbf{\Omega}_{ib}^b \Delta t) \tag{2.9}$$

Specific Force Transformation

The IMU's accelerometer measures the specific force, f_{ib}^b, acting along the sensor's body axes. Before it is processed in the velocity update step, a transformation is required to resolve this measurement in the tangent frame. The discretized approximation using the average of the prior and current attitude rotation matrices is given in Equation (2.10).

$$f_{ib}^t \approx \frac{1}{2}(C_b^t(-) + C_b^t(+))f_{ib}^b \tag{2.10}$$

Velocity Update

Next, the current velocity vector can be updated using the prior velocity and the current acceleration vector. The acceleration vector is calculated from the newly transformed specific force, f_{ib}^t, by accounting for gravity components measured by the accelerometer. This compensation is shown in Equation (2.11).

$$a_{tb}^t = f_{ib}^t + g_b^t \tag{2.11}$$

where g_b^t is the gravity model. For this application, where large errors are accumulated from other error sources, the simplified gravity model in Equation (2.12) is sufficient.

$$g_b^t = \begin{bmatrix} 0 & 0 & -9.81 \end{bmatrix}^T \tag{2.12}$$

A more accurate but complex gravity model as a function of current LLA position and standard definitions from WGS84 is detailed in [3]. The velocity vector can then be updated using Equation (2.13).

$$v_{tb}^t(+) = v_{tb}^t(-) + (a_{tb}^t + 2\mathbf{\Omega}_{ie}^t v_{tb}^t(-))\Delta t \tag{2.13}$$

Again, for simple applications, the contributions from the Earth's rotation rate can be ignored giving Equation (2.14) which is simply an Euler integration.

$$v_{tb}^t(+) \approx v_{tb}^t(-) + a_{tb}^t \Delta t \tag{2.14}$$

Position Update

Finally, the position can be updated using the current velocity vector. Equation (2.15) shows the discrete position update.

$$r_{tb}^{t}(+) = r_{tb}^{t}(-) + (v_{tb}^{t}(-) + v_{tb}^{t}(+))\frac{\Delta t}{2} \tag{2.15}$$

Note that alternative updates for position can be accessed in [3], and represent variations of numerical integration methods. For simple applications, an Euler integration may be sufficient.

2.3 Ground Vehicle Odometry

For ground vehicle navigation, a complimentary or alternative option to INS is to leverage vehicle dynamics and constraints. A representative example is a 2-DOF bicycle dynamics model with lateral velocity and yaw rate states. A free-body diagram of the system is shown in Figure 2.5.

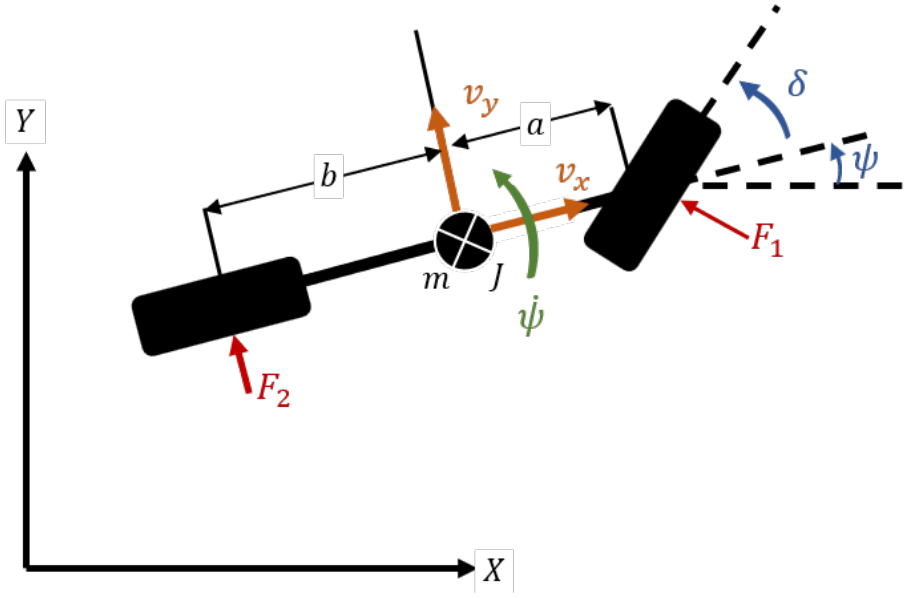


Figure 2.5: Classical Bicycle Free-Body Diagram

This work assumes that readers are familiar with the classical bicycle model, so a detailed derivation is omitted but readers are referred to [11] for detailed derivations. However, the state-space representation of the system is described in Equation (2.16).

$$\begin{bmatrix} \dot{v}_y \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} -\frac{C_1+C_2}{mv_x} & -\frac{aC_1-bC_2}{mv_x} - v_x \\ -\frac{aC_1-bC_2}{Jv_x} & -\frac{a^2C_1+b^2C_2}{Jv_x} \end{bmatrix} \begin{bmatrix} v_y \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} \frac{C_1}{m} \\ \frac{aC_1}{J} \end{bmatrix} \delta \tag{2.16}$$

In the above equation, v_y is the lateral velocity, $\dot{\psi}$ is the yaw rate, m is the mass of the vehicle, J is the yaw moment of inertia, a and b are distances from the front and rear wheel centers to the CG, respectively, and C_1 and C_2 are the tire cornering stiffnesses of the front and rear tires, respectively. Note that it is a common assumption that the longitudinal velocity v_x is available through onboard sensors and the tire steer angle at the road δ can be computed using a hand wheel steer angle reported by onboard sensors and a known steering ratio.

Using this dynamics model, the yaw angle of the vehicle can be integrated from yaw rate and lateral velocity can be directly used to solve for a tangent frame position as shown in Equation (2.17),

$$\begin{aligned} X_{k+1} &= X_k + (v_{x,k}\cos(\hat{\psi}_k) - \hat{v}_{y,k}\sin(\hat{\psi}_k))\Delta t \\ Y_{k+1} &= Y_k + (v_{x,k}\sin(\hat{\psi}_k) + \hat{v}_{y,k}\cos(\hat{\psi}_k))\Delta t \end{aligned} \tag{2.17}$$

where X and Y are arbitrary tangent frame positions of the vehicles's CG, Δt is the integration interval, and the accent $\hat{}$ denotes states that are computed from the vehicle model. Using the model to determine navigation states is contingent on having reasonable knowledge of the model's parameters. Poor model knowledge can lead to large errors that grow quickly since position is computed from an integrated yaw angle. In practice, it is often insufficient to use a dynamics model alone for navigation without external measurements. Furthermore, the 2-DOF system makes a planar assumption and is unable to recover vertical position, roll angle, and pitch angle. Nevertheless, in the context of ground vehicles, planar information can be used for dead reckoning, and a well configured vehicle model can outperform INS that use poor IMUs.

2.4 Classical Visual Inertial Odometry

This section introduces the problem space that most classical visual inertial odometry (VIO) systems seek to solve. As there are multiple distinct VIO algorithms, this section does not detail any one algorithm in particular but aims to describe the general

approach for most VIO. Figure 2.6 illustrates a camera-IMU system moving through time and observing a single feature with an unknown global position, p_f^G.

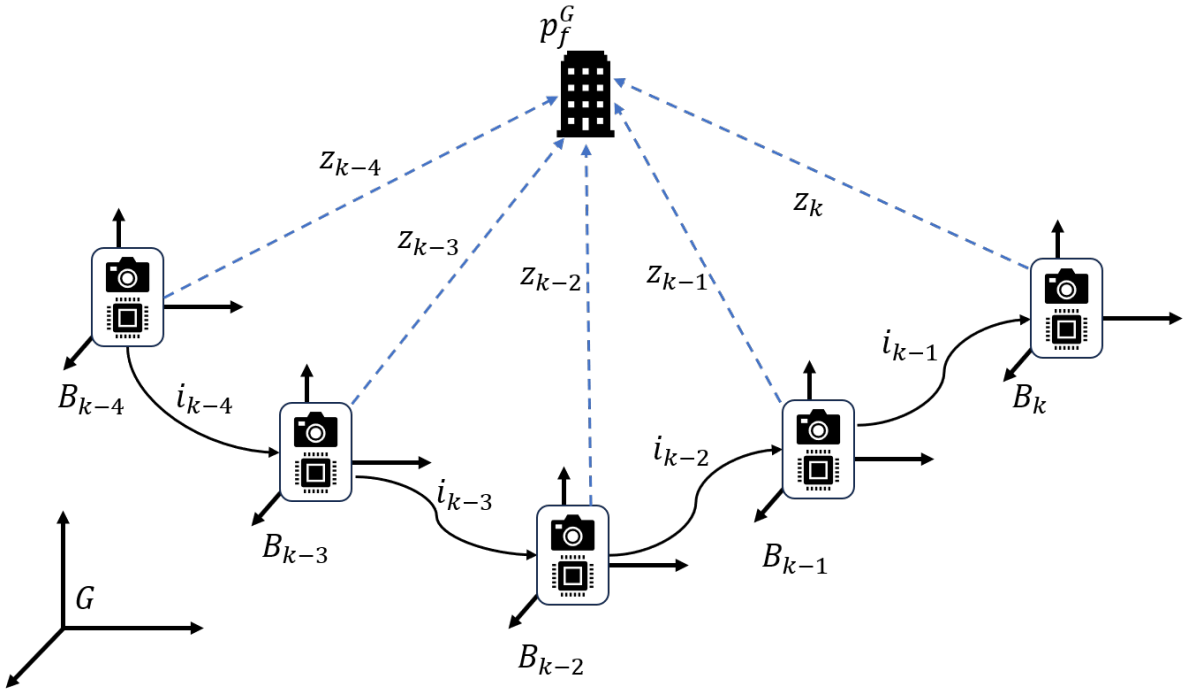


Figure 2.6: Visual-Inertial Odometry Problem Space

In Figure 2.6, the system pose is defined by coordinate frame B and the fixed global frame is defined by coordinate frame G. As the system translates and rotates through the global frame, bearings z to a feature are captured by the camera at each step k and inertial measurements i are recorded continuously in between. The general pipeline for VIO algorithms applies an INS using the inertial measurements and a camera system to detect and track features to reduce drift. The two systems are typically fused using a Kalman filter (KF) where the INS propagates states forward in time and the camera system updates the states using feature information.

Generating a measurement model from features is a complex process, but the core approach for many VIO is to use the camera's intrinsic and extrinsic parameters [26]. Doing so enables the creation a measurement model that relates the pixel point of a feature in the image frame to the state vector which is typically represented in the global frame. Specifically, intrinsic parameters include distortion coefficients,

focal lengths, and principal points which are parameters that define an individual camera system. Extrinsic parameters define the position and orientation of the physical camera to the global frame or another reference frame and is often expressed with a rotation matrix and a lever arm. Intrinsic parameters map raw pixel points from a distorted image onto an undistorted image and represent the newly rectified points in the camera's frame. Extrinsic parameters can then project those points onto the desired resolving frame. A simplified diagram of this pipeline is shown in Figure 2.7 which follows a typical VIO setup where features are first represented in the IMU sensor frame before projecting to the global frame. This is what allows measurements to directly relate to evolving states from the INS, and also accounts for various mounting configurations. For more details about intrinsic and extrinsic parameters or the projection of pixel coordinates to a 3D world frame, readers are referred to [38] and [49].

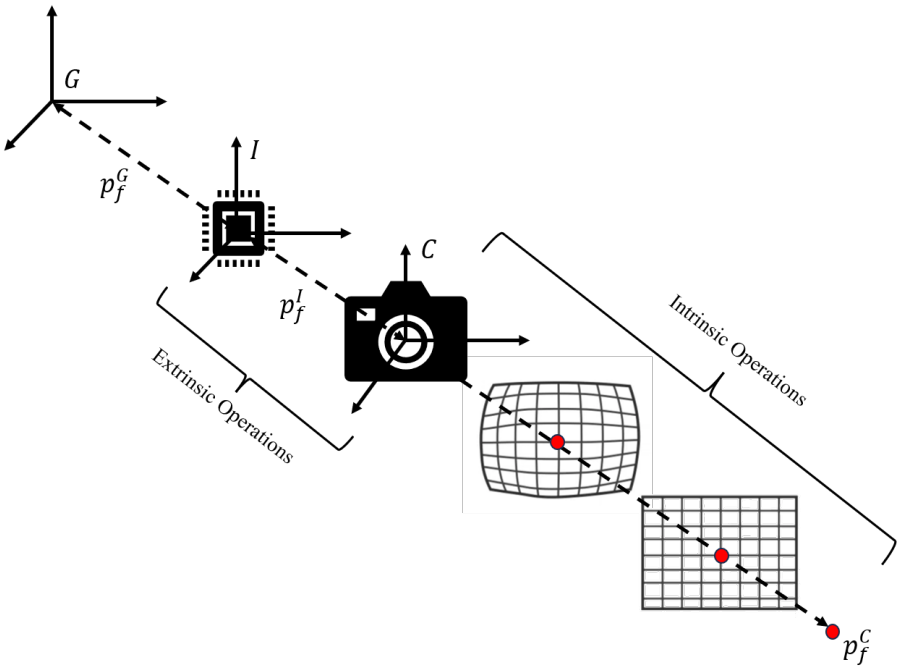


Figure 2.7: 2D Pixel Point To 3D World Frame Pipeline

Then, a single nonlinear measurement model that combines the above operations can be derived to relate feature pixel points to the selected state vector [26]. Although various VIO algorithms handle this process in different ways, most methods fundamentally require a known camera model (i.e., intrinsic and extrinsic parameters).

These parameters can be calibrated and measured prior to operation. However, the requirement to have knowledge of the system and sensor models is a fundamental limitation of modern VIO.

Chapter 3

Tractor-Trailer Model-Based Navigation

This chapter introduces the tractor-trailer dynamic model and a complementary navigation filter that is designed around the model. First, a detailed derivation of the dynamic model is presented, highlighting all assumptions. Second, the navigation filter design is presented. Third, and finally, the limitations of the system are discussed.

3.1 Tractor-Trailer Lateral Bicycle Model

As previously discussed, prior research has documented derivations for a lateral tractor-trailer system [13, 15, 16]. Although the methodology among these works slightly differ, their models make the same core assumptions which this chapter will discuss. Additionally, the previously documented models share similar state vectors despite having slight variations, but the principal states that are used in this work that represent the tractor-trailer system are defined by Equation (3.1),

$$\mathbf{x} = \begin{bmatrix} v_y & \dot{\psi} & \psi & \dot{\gamma} & \gamma \end{bmatrix}^T \tag{3.1}$$

where

v_y: Tractor lateral velocity

$\dot{\psi}$: Tractor yaw rate

ψ: Tractor yaw angle

$\dot{\gamma}$: Trailer hitch rate

γ: Trailer hitch angle

The model derived in this thesis closely follows the Wolfe model [16]. Wolfe makes multiple revisions testing different assumptions and deriving multiple models. The model derived here follows Wolfe's Revision 4 model which assumes $\cos(\gamma) = 1$ and $\sin(\gamma) = 0$, where γ is the hitch angle. This assumption is close to the small angle approximation (SAA) with $\sin(\gamma) = 0$ instead of $\sin(\gamma) = \gamma$. In [16], various approximations are made to study which model estimates the hitch angle well. Although this rendition is the simplest of Wolfe's revisions, it was demonstrated that this model performed equivalently or better than more complex versions [16]. There is no comparison of model performance in this thesis, however, the interested reader is referred to [16] to access the other revisions and their assumptions. Additionally, the analysis done in [14] compares model performance of different tractor-trailer models under various maneuvers.

The first core assumption is in regards to the selection of degree of freedom (DOF) states. The tractor-trailer mechanical system is complex and in reality has multiple degrees of freedom. In the works discussed earlier, and the derivation performed in this thesis, a simplified single-track bicycle model is adopted and illustrated in Figure 3.1. For this model, the system tracks yawing motion of the tractor and trailer bodies and neglects pitching and rolling dynamics. This is a common assumption of the bicycle model and allows for a derivation which is decoupled from those neglected states. As such, the degrees of freedom for the system are tractor lateral motion, y, tractor yaw motion, ψ, and trailer hitch motion, γ. Additionally, while the longitudinal motion of the system is not restrained, it is not part of the state vector. This is another classical bicycle model assumption where the longitudinal velocity is assumed to be non-zero and constant between integration periods.

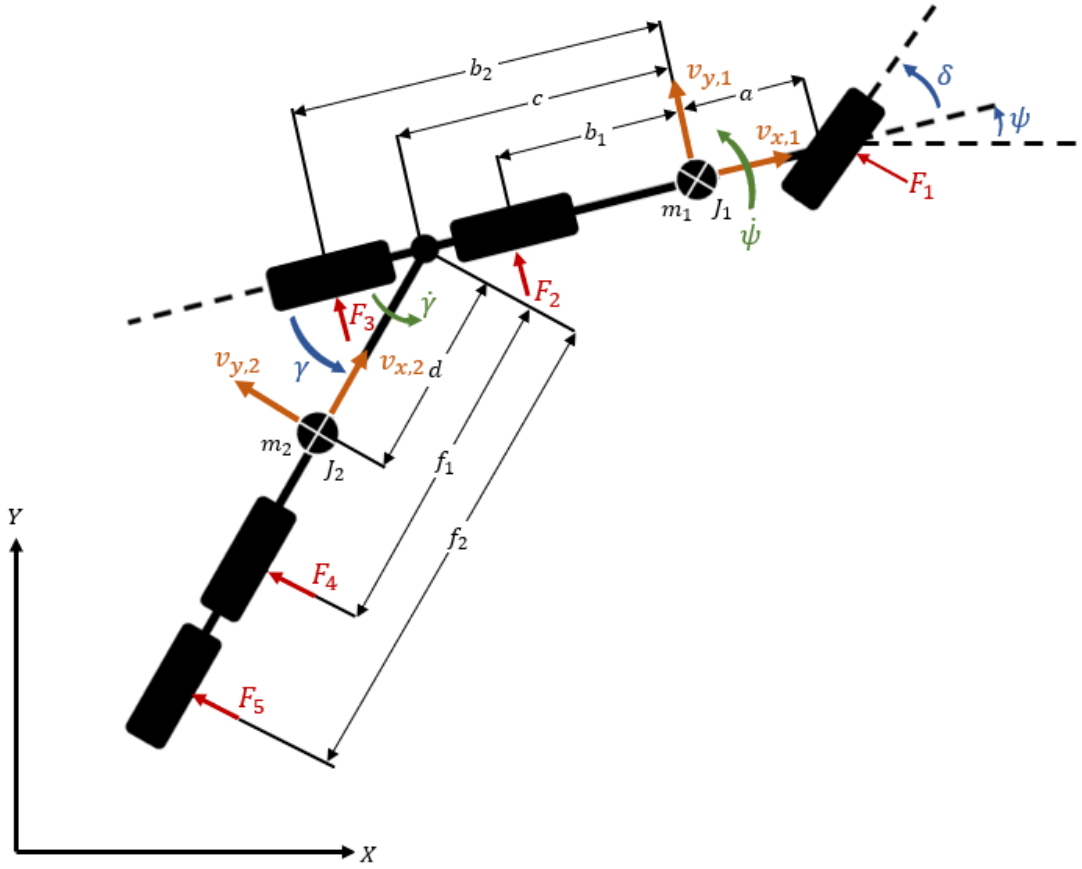


Figure 3.1: Tractor-Trailer Bicycle Model Free-Body Diagram

The tractor and trailer bodies have independent coordinate frames (denoted by subscripts 1 and 2, respectively) fixed to their CGs. Each body frame is a right-handed coordinate system with positive forward x, positive left y, and positive counterclockwise rotation. To derive the equations of motion (EOMs) of the system, all dynamics must resolve in a common frame. As such, the trailer's velocity vector, $\vec{V}_2$, is resolved in the tractor's frame using Equation (3.2),

$$\vec{V}_2 = \vec{V}_1 + \dot{\vec{r}}_{2/1} + (\vec{\omega} \times \vec{r}_{2/1}) \tag{3.2}$$

where $\vec{V}_1$ is the tractor's velocity vector, $\vec{\omega}$ is the angular velocity vector acting on the reference body (i.e., tractor), $\vec{r}_{2/1}$ is the lever arm from the tractor's coordinate frame to the trailer's coordinate frame, $\dot{\vec{r}}_{2/1}$ is the rate of change of the lever arm, and $(\times)$

denotes the cross product. The trailer's longitudinal and lateral velocity components resolved in the tractor's frame are given in Equations (3.3) and (3.4), respectively.

$$v_{x,2} = v_{x,1} + (\dot{\psi} + \dot{\gamma}) d \sin(\gamma) \approx v_{x,1} \tag{3.3}$$

$$v_{y,2} = v_{y,1} - (c + d\cos(\gamma))\dot{\psi} - d\cos(\gamma)\dot{\gamma} \tag{3.4}$$

Equation (3.3) indicates the trailer's longitudinal velocity, $v_{x,2}$, is approximately the tractor's longitudinal velocity, $v_{x,1}$, since in most cases $v_{x,1}$ dominates the remaining terms ($v_{x,1} >> (\dot{\psi}+\dot{\gamma}) d \sin(\gamma)$). The same assumption cannot be applied to the trailer's lateral velocity, $v_{y,2}$. In the remainder of this thesis, the trailer's velocity components, $v_{x,2}$ and $v_{y,2}$, will be considered resolved in the tractor's coordinate frame.

3.1.1 Lateral Tire Forces

The second core assumption is another bicycle model assumption regarding tire forces. All lateral forces acting through the tires are assumed to stay within the linear tire region as described in Equation (3.5),

$$F_j = C_{\alpha,j}\alpha_j, \quad \forall j \tag{3.5}$$

where C_α is the tire cornering stiffness in $\frac{N}{rad}$, and α is the tire slip angle in rad of the j^{th} tire. The tire slip angles are defined as the angle between the instantaneous velocity vector at the tire and the tire's longitudinal axis which is shown in the kinematic diagrams in Figure 3.2. The tire slip angle expressions for each axle are given in Equations (3.6 - 3.8).

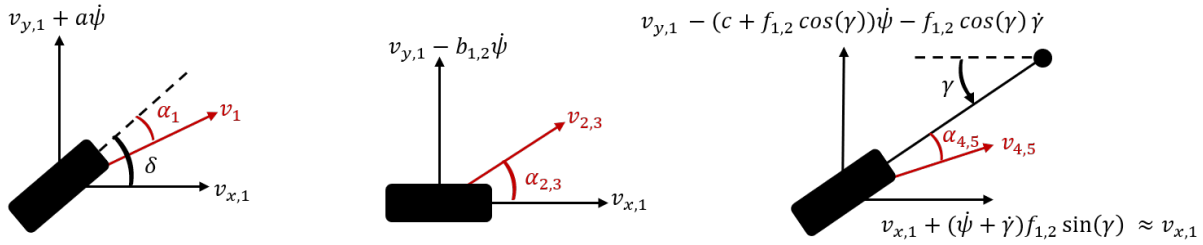


Figure 3.2: Kinematic Slip Angle Diagrams

$$\alpha_1 = \delta - \arctan(\frac{v_{y,1} + a\dot{\psi}}{v_{x,1}}) \approx \delta - (\frac{v_{y,1} + a\dot{\psi}}{v_{x,1}}) \tag{3.6}$$

$$\alpha_{2,3} = -\arctan(\frac{v_{y,1} - b_{1,2}\dot{\psi}}{v_{x,1}}) \approx -(\frac{v_{y,1} - b_{1,2}\dot{\psi}}{v_{x,1}}) \tag{3.7}$$

$$\begin{aligned} \alpha_{4,5} &= \gamma - \arctan(\frac{v_{y,1} - (c + f_{1,2}\cos(\gamma))\dot{\psi} - f_{1,2}\cos(\gamma)\dot{\gamma}}{v_{x,1}}) \\ &\approx \gamma - (\frac{v_{y,1} - (c + f_{1,2}\cos(\gamma))\dot{\psi} - f_{1,2}\cos(\gamma)\dot{\gamma}}{v_{x,1}}) \end{aligned} \tag{3.8}$$

Again, in most scenarios, $v_{x,1}$ dominates the other terms so a SAA can be applied.

3.1.2 Lagrangian Derivation

Once dynamics are resolved in a common frame, and lateral tire forces are defined, the system's EOMs can be derived using Langrange's equation given by Equation (3.9).

$$\frac{d}{dt}(\frac{\delta L}{\delta \dot{q}_i}) - \frac{\delta L}{\delta q_i} = Q_{q_i} \tag{3.9}$$

The Langrangian, L, given in Equation (3.10).

$$L = T - V \tag{3.10}$$

The above equation is the difference between a system's kinetic and potential energies and the expression for this system is given in Equation (3.11).

$$L = T = \frac{1}{2}m_1(v_{x,1}^2 + v_{y,1}^2) + \frac{1}{2}J_1\dot{\psi}^2 + \frac{1}{2}m_2(v_{x,2}^2 + v_{y,2}^2) + \frac{1}{2}J_2(\dot{\psi} + \dot{\gamma})^2 \tag{3.11}$$

Note that for this system, the potential energy, V, is zero, thus the Lagrange, L, is composed of the kinetic energy term, T, only.

The generalized coordinates, q_i, correspond to the degrees of freedom such that $q_1 = y$, $q_2 = \psi$, and $q_3 = \gamma$. To complete Langrange's equation, the corresponding generalized forces, Q_{q_i}, must be calculated. The generalized forces of a system are forces or moments corresponding to the generalized coordinates and represent the work done on the system by external forces. The general expression is given in Equation (3.12),

$$Q_{q_i} = \frac{W_{q_i}}{\delta q_i} \tag{3.12}$$

where W_{q_i}, shown in Equation (3.13), is the virtual work of the forces, F_j, acting on particles, $p_j,\ j = 1, \ldots, n$, induced by a virtual displacement, δq_i.

$$W_{q_i} = \sum_{j=1}^{n} F_j \cdot \delta q_i \tag{3.13}$$

To derive the generalized forces, Q_{q_i}, each corresponding generalized coordinate, q_i, is displaced while holding all other DOFs in place. For each configuration, the sum of forces or moments that contribute to the displacement defines the virtual work and generalized force expressions. Equations (3.14 - 3.16) give the generalized forces for this system.

$$Q_y = F_1 \cos(\delta) + F_2 + F_3 + F_4 \cos(\gamma) + F_5 \cos(\gamma) \tag{3.14}$$

$$Q_\psi = F_1 a \cos(\delta) - F_2 b_1 - F_3 b_2 - F_4(c \cos(\gamma) + f_1) - F_5(c \cos(\gamma) + f_2) \tag{3.15}$$

$$Q_\gamma = -F_4 f_1 - F_5 f_2 \tag{3.16}$$

Note that these expressions simply state how the forces or moments induced by the lateral tire forces acting on the system work to displace individual degrees of freedom.

With all the elements in Equation (3.9) defined, the EOMs of the system can be derived. The derivations are performed using Python's symbolic library, SymPy, and three EOMs corresponding to each generalized coordinate are derived. It is important

to note that the resulting lateral acceleration terms of the tractor, $\ddot{y}$, do not account for the contribution from the forward velocity and yaw rate. Thus, the lateral acceleration terms are augmented accordingly as given by Equation (3.17).

$$\ddot{y} = \dot{v}_{y,1} + v_{x,1}\dot{\psi} \tag{3.17}$$

As described previously, an approximation on hitch angle terms is applied such that $\cos(\gamma) = 1$ and $\sin(\gamma) = 0$. This assumption creates quasi-linear EOMs as described in [16]. Nonlinearities as a function of the steer angle, δ, are still present, but can be easily accounted for as δ is assumed to be known. Due to the complexity of the EOMs, they are not listed in this section and a state-space representation is presented instead.

3.1.3 State-Space Representation

Due to the complexity of the EOMs, they are rearranged into a state-space representation with respect to the states, $\mathbf{x}$, as described in Equation (3.18).

$$\mathbf{M}\dot{\mathbf{x}} = \mathbf{K}\mathbf{x} + \mathbf{F}\mathbf{u} \tag{3.18}$$

The continuous-time dynamics of the system are given by Equation (3.19) and can be rewritten in a more common form as shown in Equation (3.20),

$$\dot{\mathbf{x}} = \mathbf{M}^{-1}\mathbf{K}\mathbf{x} + \mathbf{M}^{-1}\mathbf{F}\mathbf{u} \tag{3.19}$$

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{3.20}$$

where $\mathbf{M}$ is the mass/inertial matrix, $\mathbf{K}$ is the stiffness matrix, and $\mathbf{F}$ is the force matrix, $\mathbf{A}$ is the system's state transition matrix, and $\mathbf{B}$ is the system's input matrix. The input, $\mathbf{u}$, to the system is the steer angle of the front tires with respect to the longitudinal axis given in Equation (3.21).

$$\mathbf{u} = \delta \tag{3.21}$$

The compounding matrices, $\mathbf{M}$, $\mathbf{K}$, and $\mathbf{F}$, which make up the state transition and input matrices are formulated by grouping common terms across each EOM. $\mathbf{M}$ organizes terms on the left-hand side of each EOM with respect to the time derivative of the state vector, $\dot{\mathbf{x}}$, $\mathbf{K}$ organizes terms on the right-hand side with respect to the state vector, $\mathbf{x}$, and $\mathbf{F}$ organizes terms on the right-hand side with respect to the input, $\mathbf{u}$. Furthermore, to track the yaw angle, ψ, and the hitch angle, γ, the matrices must be augmented with integrators. For the continuous-time matrices, the augmentations are carried out by appending rows such that $\dot{\psi} = \dot{\psi}$ and $\dot{\gamma} = \dot{\gamma}$. When discretized, the augmentations give Equations (3.22) and (3.23), which are Euler integrations.

$$\psi_{k+1} = \psi_k + \dot{\psi}_k \Delta t \tag{3.22}$$

$$\gamma_{k+1} = \gamma_k + \dot{\gamma}_k \Delta t \tag{3.23}$$

Each matrix is defined in Equations (3.24 - 3.26).

$$\mathbf{M} = \begin{bmatrix} m_1 + m_2 & m_2\left(-c-d\right) & \left(m_1+m_2\right)v_{x,1} & -dm_2 & 0 \\ m_2\left(-c-d\right) & J_1 + J_2 + m_2(c^2+2cd+d^2) & m_2\left(-c-d\right)v_{x,1} & J_2 + m_2(cd+d^2) & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -dm_2 & J_2 + m_2(cd+d^2) & -dm_2 v_{x,1} & J_2 + d^2 m_2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.24}$$

$$\mathbf{K} = \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} & K_{15} \\ K_{21} & K_{22} & K_{23} & K_{24} & K_{25} \\ 0 & 1 & 0 & 0 & 0 \\ K_{41} & K_{42} & K_{43} & K_{44} & K_{45} \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.25}$$

Where:

$$K_{11} = -\frac{1}{v_{x,1}}(C_1 \cos(\delta) + C_2 + C_3 + C_4 + C_5)$$

$$K_{12} = \frac{1}{v_{x,1}}(-C_1 a \cos(\delta) + C_2 b_1 + C_3 b_2 + C_4(c + f_1) + C_5(c + f_2))$$

$$K_{13} = 0$$

$$K_{14} = \frac{1}{v_{x,1}}(C_4 f_1 + C_5 f_2)$$

$$K_{15} = C_4 + C_5$$

$$K_{21} = \frac{1}{v_{x,1}}(-C_1 a \cos(\delta) + C_2 b_1 + C_3 b_2 + C_4(c + f_1) + C_5(c + f_2))$$

$$K_{22} = -\frac{1}{v_{x,1}}(C_1 a^2 \cos(\delta) + C_2 b_1^2 + C_3 b_2^2 + C_4(c^2 + 2cf_1 + f_1^2) + C_5(c^2 + 2cf_2 + f_2^2))$$

$$K_{23} = 0$$

$$K_{24} = -\frac{1}{v_{x,1}}(C_4(cf_1 + f_1^2) + C_5(cf_2 + f_2^2))$$

$$K_{25} = -(C_4(c + f_1) + C_5(c + f_2))$$

$$K_{41} = \frac{1}{v_{x,1}}(C_4 f_1 + C_5 f_2)$$

$$K_{42} = -\frac{1}{v_{x,1}}(C_4(cf_1 + f_1^2) + C_5(cf_2 + f_2^2))$$

$$K_{43} = 0$$

$$K_{44} = -\frac{1}{v_{x,1}}(C_4 f_1^2 + C_5 f_2^2)$$

$$K_{45} = -C_4 f_1 - C_5 f_2$$

$$\mathbf{F} = \begin{bmatrix} C_1 \cos(\delta) \\ C_1 a \cos(\delta) \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.26}$$

Note that different revisions of the matrices can be accessed in [16] or manually derived from the full nonlinear EOMs.

The required model parameters for this system and their descriptions are given in Table 3.1.

Table 3.1: Tractor-Trailer Bicycle Model Parameters

Parameters	Description	Units
C_1	First axle tire cornering stiffness	N/rad
$C_{2,3}$	Second and third axle tire cornering stiffness	N/rad
$C_{4,5}$	Fourth and fifth axle tired cornering stiffness	N/rad
m_{s1}	Tractor sprung mass	kg
$m_{us,A1}$	First axle unsprung mass	kg
$m_{us,A23}$	Second and third axle unsprung mass	kg
a	Distance from the first axle to tractor CG	m
b_1	Distance from the second axle to the tractor CG	m
b_2	Distance from the third axle to the tractor CG	m
c	Distance from the tractor CG to the hitch point	m
R_{z1}	Tractor Yaw radius of gyration	m
m_{s2}	Trailer sprung mass	kg
$m_{us,A45}$	Fourth and fifth axle unsprung mass	kg
d	Distance from the hitch point to the trailer CG	m
f_1	Distance from the hitch point to the fourth axle	m
f_2	Distance from the hitch point to the fifth axle	m
R_{z2}	Trailer Yaw radius of gyration	m

The sprung and unsprung masses are summed to compute the total masses of the tractor and trailer as shown in Equations (3.27) and (3.28).

$$m_1 = m_{s1} + m_{us,A23} \tag{3.27}$$

$$m_2 = m_{s2} + m_{us,A45} \tag{3.28}$$

Furthermore, the yaw radius of gyration is used to compute the yaw moment of inertia as shown in Equations (3.29) and (3.30).

$$J_1 = m_1 R_{z1}^2 \tag{3.29}$$

$$J_2 = m_2 R_{z2}^2 \tag{3.30}$$

3.2 Navigation Extended Kalman Filter Design

To navigate using the dynamic model, errors from model uncertainty and unknown disturbances must be addressed. Therefore, the tractor-trailer model is fused with yaw rate measurements from an onboard IMU in a navigation filter architecture. The navigation filter is an extended Kalman filter (EKF) which is similar to the Kalman filter, but does not require the state transition and observation models to be linear. Instead, nonlinear functions that represent the states and observations may be used to propagate states forward in time and compute the predicted measurements. Then, their Jacobians are evaluated around the current state estimate, and used for the Kalman filter equations. Furthermore, the EKF does not require that both the state dynamics and all measurement observation models be nonlinear, and the algorithm may handle an arbitrary combination of linear and nonlinear models. A diagram of the EKF algorithm is shown in Figure 3.3. This thesis assumes that readers are familiar with the KF and EKF algorithm, and omits detailed derivations and discussions of the standard KF equations, but more information may be found in [50] and [51]. The next subsections detail the design, implementation, and limitations of the navigation filter.

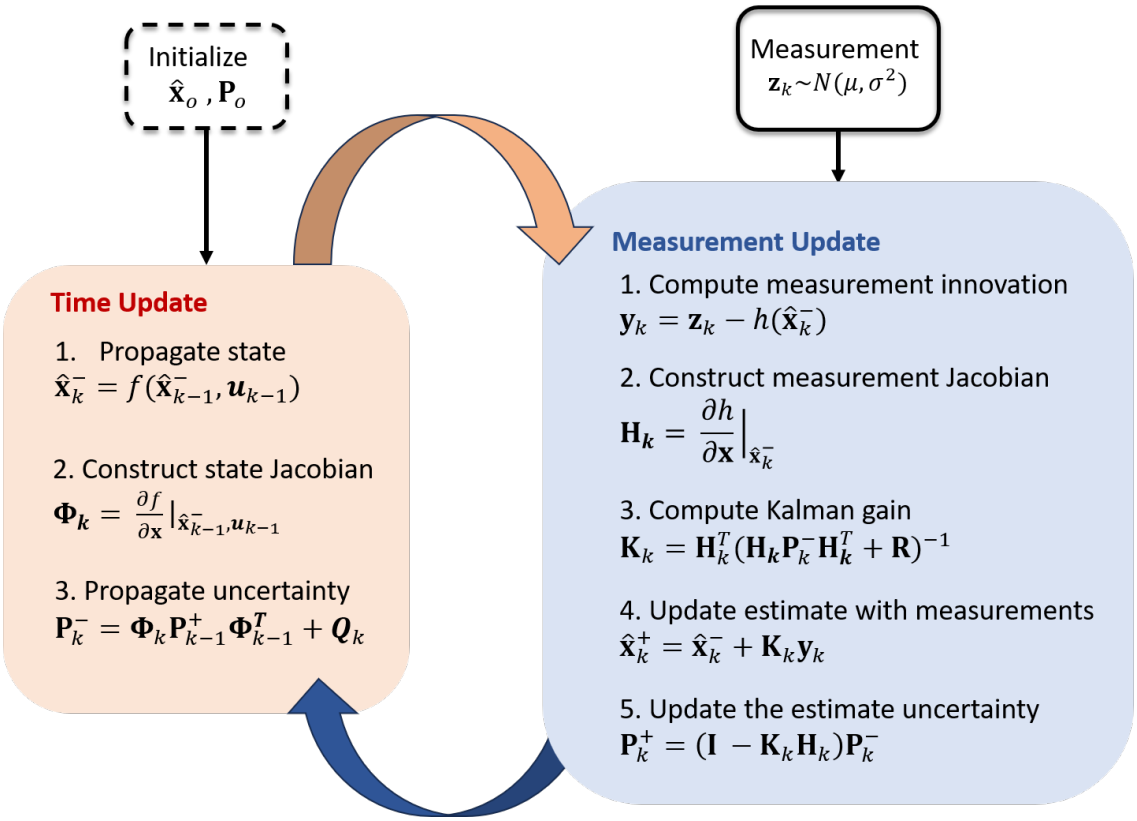


Figure 3.3: Extended Kalman Filter Algorithm

3.2.1 State Propagation

First, the filter states are defined by the state vector in Equation (3.31).

$$\mathbf{x} = \begin{bmatrix} N & E & v_x & v_y & \dot{\psi} & \psi & \dot{\gamma} & \gamma & b_{\dot{\psi}} \end{bmatrix}^T \tag{3.31}$$

Note that this state vector is similar to the tractor-trailer model state vector with the addition of tangential positions (i.e., North N and East E positions), the tractor's longitudinal velocity, v_x, and a yaw rate bias, $b_{\dot{\psi}}$, from the gyroscope.

The North and East positions are estimated using the most recent vehicle states as described in Equations (3.32) and (3.33).

$$\hat{N}_{k+1} = \hat{N}_k + \left[\hat{v}_{x,k} \sin(\hat{\psi}_k) + \hat{v}_{y,k} \cos(\hat{\psi}_k) \right] \Delta t \tag{3.32}$$

$$\hat{E}_{k+1} = \hat{E}_k + \left[\hat{v}_{x,k} \cos(\hat{\psi}_k) - \hat{v}_{y,k} \sin(\hat{\psi}_k) \right] \Delta t \tag{3.33}$$

The calculation is essentially an Euler integration of body velocities rotated into the tangential frame. Since the states contain nonlinear sines and cosines of the yaw state, they are propagated using the nonlinear functions and their Jacobians are used to propagate their covariances. The longitudinal velocity, v_x is modeled as constant within integration periods as described in Equation (3.34), and is provided as a measurement in the measurement correction stage.

$$\hat{v}_{x,k+1} = \hat{v}_{x,k} \tag{3.34}$$

Unlike the tractor-trailer dynamics model, the longitudinal velocity is incorporated as a state and a measurement which allows for advantages such as covariance tracking, measurement error modeling, and measurement rejection capabilities.

The lateral velocity, yaw rate, yaw, hitch rate, and hitch states are propagated using the discretized tractor-trailer state-transition matrix, $\mathbf{A_d}$, and input matrix $\mathbf{B_d}$ as described by Equation (3.35),

$$\mathbf{x_{veh},}_{k+1} = \mathbf{A_{d},}_{k}\mathbf{x_{veh},}_{k} + \mathbf{B_{d},}_{k}\mathbf{u}_k \tag{3.35}$$

where $\mathbf{x_{veh}}$ is equivalent to Equation (3.1), and $\mathbf{u}$ is the steer angle of the tires. The discretized system is computed from the continuous-time matrices that are detailed in the previous section using Zero-Order Hold (ZOH) discretization. The discrete state transition matrix is given in Equation (3.36).

$$\mathbf{A_d} = e^{\mathbf{A}\Delta t} \tag{3.36}$$

The discrete input matrix is given in Equation (3.37).

$$\mathbf{B_d} = \mathbf{A}^{-1}(e^{\mathbf{A}\Delta t} - \mathbf{I})\mathbf{B} \tag{3.37}$$

Finally, the yaw rate bias, is modeled as a constant bias as in Equation (3.38). Additionally, since yaw is directly integrated from yaw rate, the bias term affects the yaw estimate as described by Equation (3.39).

$$\hat{b}_{\dot{\psi},k+1} = \hat{b}_{\dot{\psi},k} \tag{3.38}$$

$$\hat{\psi}_{k+1} = \hat{\psi}_k + \hat{\dot{\psi}}_k \Delta t + \hat{b}_{\dot{\psi},k} \Delta t \tag{3.39}$$

Each state is propagated using their respective equations and the complete Jacobian of the state dynamics is constructed as shown in Equation (3.40),

$$\mathbf{\Phi}_k = \left[\begin{array}{cccccc:c} 1 & 0 & \mathbf{\Phi_{13}} & \mathbf{\Phi_{14}} & 0 & \mathbf{\Phi_{16}} & \\ 0 & 1 & \mathbf{\Phi_{23}} & \mathbf{\Phi_{24}} & 0 & \mathbf{\Phi_{26}} & \mathbf{0}_{3\times 3} \\ 0 & 0 & 1 & 0 & 0 & 0 & \\ \hdashline & & & & & & 0 \\ & & & & & & 0 \\ & \mathbf{0}_{5\times 3} & & \mathbf{A}_{\mathbf{d},k} & & & \mathbf{\Phi_{69}} \\ & & & & & & 0 \\ & & & & & & 0 \\ \hdashline & & & \mathbf{0}_{1\times 8} & & & 1 \end{array}\right] \tag{3.40}$$

where

- $\mathbf{\Phi_{13}} = \sin(\hat{\psi}_k)\Delta t$
- $\mathbf{\Phi_{14}} = \cos(\hat{\psi}_k)\Delta t$
- $\mathbf{\Phi_{16}} = (\hat{v}_{x,k}\cos(\hat{\psi}_k) - \hat{v}_{y,k}\sin(\hat{\psi}_k))\Delta t$
- $\mathbf{\Phi_{23}} = \cos(\hat{\psi}_k)\Delta t$
- $\mathbf{\Phi_{24}} = -\sin(\hat{\psi}_k)\Delta t$
- $\mathbf{\Phi_{26}} = (-\hat{v}_{x,k}\sin(\hat{\psi}_k) - \hat{v}_{y,k}\cos(\hat{\psi}_k))\Delta t$

- $\mathbf{\Phi_{69}} = \Delta t$

Since $\mathbf{A_d}$ is already a linearized representation of the vehicle state dynamics, the Jacobian is constructed around it, and only the North and East positions are linearized.

Again, this navigation filter is generally intended to take over during GNSS outages. Therefore, it is assumed that this filter can be initialized with accurate state estimates from high-grade sensors or a preceding navigation filter where positioning, velocity, and attitude measurements are available. It is imperative that the filter be initialized this way since it provides a dead-reckoned position solution, and poor initialization, especially for the heading state, will immediately cause large and unrecoverable position errors. The states can be propagated using their respective equations and their covariances can be extrapolated as shown in as shown in Equations (3.41 - 3.43), respectively.

$$\hat{\mathbf{x}}_k^- = f(\hat{\mathbf{x}}_{k-1}^-, \mathbf{u}_{k-1}) \tag{3.41}$$

$$\mathbf{P}_k^- = \mathbf{\Phi}_{k-1}\mathbf{P}_{k-1}^+\mathbf{\Phi}_{k-1}^T + \mathbf{Q} \tag{3.42}$$

3.2.2 Measurement Correction

Because small errors in yaw estimates lead to large position errors, the yaw rate measurements from a tractor-mounted IMU serves to further reduce yaw drift induced by model uncertainty and unknown disturbances. The yaw rate measurement is modeled as shown in Equation (3.43),

$$\tilde{\dot{\psi}} = \dot{\psi} + b_{\dot{\psi}} + \nu \tag{3.43}$$

where $\dot{\psi}$ is the true yaw rate signal, and $b_{\dot{\psi}}$ is the yaw rate bias, and $\nu \sim \mathcal{N}(0, \sigma_{\dot{\psi}}^2)$ is additive, zero-mean, Gaussian white noise (GWN). Additionally, as discussed previously, the tractor's longitudinal velocity, v_x is also provided as a measurement. For most ground vehicle platforms, including the tractor unit tested in this work, this signal is readily available from reasonably accurate onboard measurements such as wheel

speed sensors used for the Anti-lock Break System (ABS). As such, this measurement is modeled with white noise and no bias as shown in Equation (3.44),

$$\tilde{v}_x = v_x + \nu \tag{3.44}$$

where v_x is the true longitudinal velocity signal and $\nu \sim \mathcal{N}(0, \sigma_{v_x}^2)$ is additive, zero-mean GWN. The measurement vector can then be expressed by Equation (3.45).

$$\mathbf{z}_k = \begin{bmatrix} \tilde{v}_{x,k} \\ \tilde{\dot{\psi}}_k \end{bmatrix} \tag{3.45}$$

Thus, the corresponding measurement observation matrix can be written by Equation (3.46).

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.46}$$

Finally, the measurement correction equations can be applied. This includes computing the Kalman gain, updating state estimates and updating state covariances which are represented by Equations (3.47 - 3.49) respectively.

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R})^{-1} \tag{3.47}$$

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k (\mathbf{z}_k + \mathbf{H}_k \hat{\mathbf{x}}_k^-) \tag{3.48}$$

$$\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^- \tag{3.49}$$

3.2.3 Zero Velocity Updates

Because the tractor-trailer bicycle model requires that v_x be non-zero, a zero velocity update (ZUPT) can be used. For this implementation, a simple approach is used which exploits basic ground vehicle constraints. First, a velocity threshold is set and a ZUPT is implemented if the measured velocity falls below the threshold. For the time

update, all rate states are set to zero and position states are held over from the previous time step while the yaw rate bias state is still modeled as a constant bias. This is achievable by augmenting the state transition matrix as shown in Equation (3.50),

$$\mathbf{\Phi}_{\text{zupt},k} = \text{diag}\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{3.50}$$

where the diagonal elements shown represent the ZUPT dynamics and the upper and lower triangular elements are zeros. For the measurement update, the same observation matrix H is applied, however the measurements themselves are augmented. The longitudinal velocity measurement is set to zero, while the yaw rate measurements are still use as a measurement as shown in Equation (3.51).

$$\mathbf{z}_{\text{zupt},k} = \begin{bmatrix} 0 \\ \hat{\dot{\psi}}_k \end{bmatrix} \tag{3.51}$$

This selection allows for the observability of the yaw rate bias during ZUPTs.

3.2.4 Limitations

The limiting factor of the model-based navigation approach lies in having to know the model's parameters. When parameters are well known, this approach can outperform INS, especially when using a low-cost IMU. However, model parameters are often not known perfectly and for tractor-trailer systems, the model uncertainty problem is very common. Referring again to Table 3.1, several of the required parameters may not be known accurately as they may vary between operations. For example, several of the trailer distances require knowledge of the trailer's CG location which is often not known and will likely change with payload configurations. Similarly, the trailer mass may not be known accurately and is likely to change depending on payloads. To reliably use the model-based method, the system parameters must be measured prior to operation or estimated online. Otherwise, the navigation and state estimation performance may severely degrade. To address this limitation, this thesis proposes an

alternative, data-driven method that only uses sensor inputs. This is achieved through modern deep learning network design, and is discussed next in Chapters 4 and 5.

Chapter 4

Deep Learning Background

To address the limitations of classical dead reckoning systems for tractor-trailers, a deep learning approach is proposed. The goal of using deep learning methods is to recover navigation and state estimation solutions purely from sensor data which eliminates the reliance on a vehicle model. In this chapter, a brief technical background of modern deep learning methods and architectures are presented. First, the general foundations of deep learning are discussed. Lastly, relevant network architectures used for the final proposed model are detailed.

4.1 Deep Learning Background

The term "deep learning" simply refers to a machine learning method that leverages a neural network or combinations of networks which are composed of multiple layers. Deep learning has become an incredibly popular tool used for many distinct applications in data analysis, language processing, image processing, and multimodal fusion [52]. Deep learning models are parametric models which may include multilayer perceptrons (MLPs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), generative adversarial networks (GANs), and transformers. This section aims to discuss the basic foundations for deep learning.

4.1.1 Single Neuron Model

First, the model of a single neuron is introduced which will act as a platform to discuss the entire learning process. Consider an arbitrary vector of inputs, x, from

the $(L-1)^{th}$ layer that are entering a single neuron. These inputs may represent the input data to the network or outputs from previous neurons. The input vector has a corresponding vector of learnable weights, w^L, unique to that neuron. Within the neuron, a weighted sum of the inputs and weights is taken and a learnable bias, b^L, is optionally added to the result. Finally, the output is passed to an optional activation function, f, where the final activation value, a^L, of the neuron is computed. The generic model of a single neuron is represented in Equation (4.1), and a visual diagram is illustrated in Figure 4.1.

$$a^L = f(b^L + \sum_i x_i w_i^L) \tag{4.1}$$

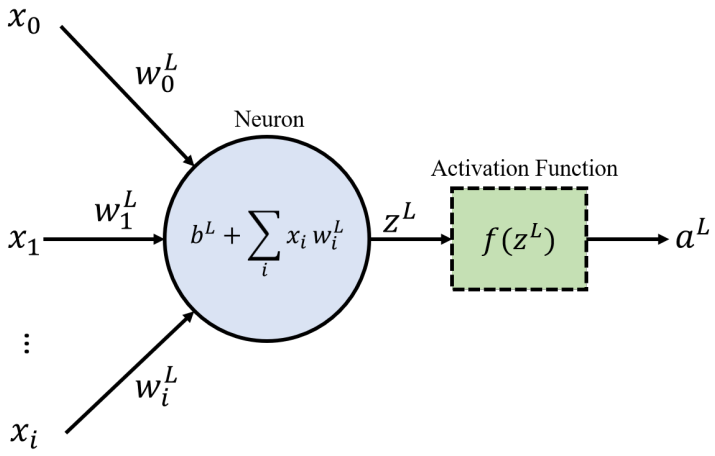


Figure 4.1: Single Neuron Model

Activation functions are nonlinear functions that enable a network to learn complex patterns. They determine whether the result of the weighted sum inside the neuron should cause that neuron to activate and by what scale. Although, activation functions are considered optional, they are often included between network layers. Cases where an activation function is omitted may be any layer where unrestricted output values are desired. There are several activation functions, and it is generally a design choice when constructing the network architecture [53]. However, modern deep learning models most often use the rectified linear unit (ReLU), which is simply $\max(0, x)$,

or its variants as they often promote better gradient flow than other activation functions [54]. ReLU variants are not defined formally in this section, however several common variants are shown in Figure 4.2. Note that the main distinction among the variants is how they handle negative input differently from each other, promoting different types of gradient flow for those inputs.

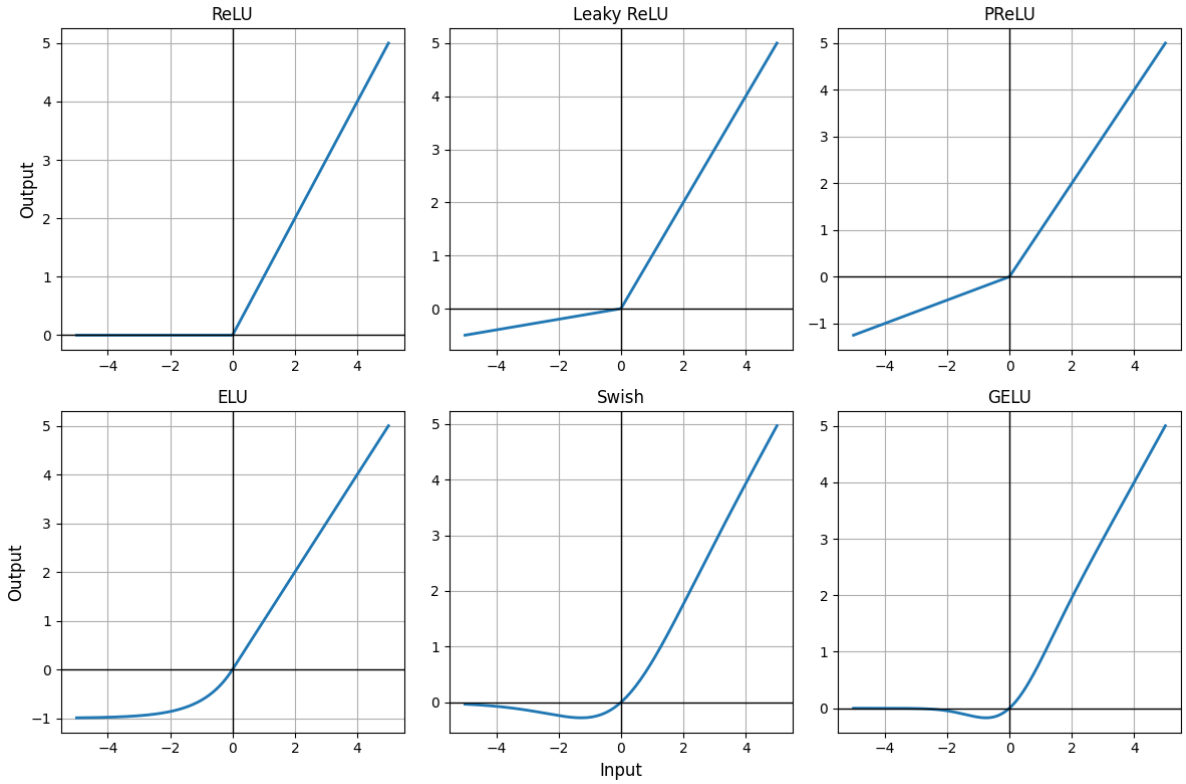


Figure 4.2: ReLU Activation and Its Variants

4.1.2 Backpropagation

Backpropagation is the fundamental process behind network learning. At its core, backpropagation is used to determine gradients from a cost function to update the learnable parameters such as the weights and biases of a network. Before discussing backpropagation in detail, it is important to understand the role of cost functions for neural networks. There are several cost functions, each appropriate for the network's main task whether it be regression, classification, or generative modeling [55]. As such, Equation (4.2) shows the generic representation of a cost function, C, which takes the output from the neural network, x, and a target truth, y, as inputs.

$$\text{cost} = C(x, y) \tag{4.2}$$

A cost value is calculated measuring the network's ability to predict the target truth. To better understand how this value is used to update the network parameters, it is more common to consider the cost as a multidimensional function where the network's weights and biases are its inputs. The learning process involves minimizing this cost function across all its dimensions by updating the network's parameters appropriately.

This process is achieved by an iterative method called gradient descent (GD) where the negative gradient of the cost function is used to take steps towards a minimum. Traditional GD is often too computationally complex since it requires the full cost to be computed over an entire training set before updating the network parameters. Alternatively, stochastic gradient descent (SGD) approximates the full cost as an average of costs across each training sample, k, in a batch of random training samples, n. The SGD approximated cost is represented by Equation (4.3).

$$C \approx \frac{1}{n} \sum_{k=0}^{n-1} C_k \tag{4.3}$$

Likewise, the gradient of the cost function with respect to the network parameters can be approximated similarly as shown in Equation (4.4).

$$\nabla C \approx \frac{1}{n} \sum_{k=0}^{n-1} \nabla C_k = \frac{1}{n} \sum_{k=0}^{n-1} \begin{bmatrix} \frac{\partial C_k}{\partial w^0} \\ \frac{\partial C_k}{\partial b^0} \\ \vdots \\ \frac{\partial C_k}{\partial w^L} \\ \frac{\partial C_k}{\partial b^L} \end{bmatrix} \tag{4.4}$$

The chain rule is used to calculate the gradient of the cost with respect to to the network's parameters and the inputs or activations from previous layers. A reillustration of gradients present in a single neuron model is shown in Figure 4.3. For this single

neuron example, the equations representing these gradients are shown in Equations (4.5 - 4.7).

$$\frac{\partial C_k}{\partial w_i^L} = \frac{\partial z^L}{\partial w_i^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C_k}{\partial a^L} \tag{4.5}$$

$$\frac{\partial C_k}{\partial b^L} = \frac{\partial z^L}{\partial b^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C_k}{\partial a^L} \tag{4.6}$$

$$\frac{\partial C_k}{\partial x_i} = \frac{\partial z^L}{\partial x_i}\frac{\partial a^L}{\partial z^L}\frac{\partial C_k}{\partial a^L} \tag{4.7}$$

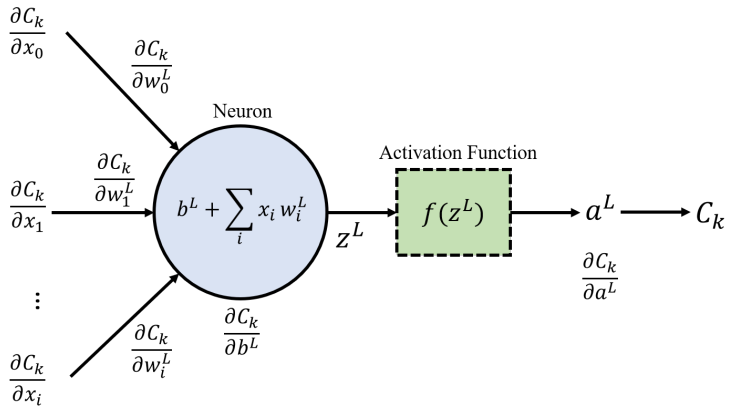


Figure 4.3: Gradients in a Single Neuron

The gradients with respect to weights and biases are used to update each weight and bias in the network. The standard SGD update is shown in Equations (4.8 - 4.9),

$$w^+ = w^- - \gamma\frac{1}{n}\sum_{k=0}^{n-1}\frac{\partial C_k}{\partial w} \tag{4.8}$$

$$b^+ = b^- - \gamma\frac{1}{n}\sum_{k=0}^{n-1}\frac{\partial C_k}{\partial b} \tag{4.9}$$

where γ represents the learning rate. Then, the gradients with respect to the inputs are used to compute the previous layer's gradients, and this process repeats for all layers. This idea of retrieving gradients along each layer to update learnable parameters is backpropagation.

Outside the trivial single neuron example, network architectures are often much more complex. Backpropagation computations are unique for each network and scales with the network's complexity. However, the core algorithm detailed using a single neuron model captures the core idea of deep learning. Additional variations to the backpropagation algorithms for distinct architectures, or a vectorized representation is out of scope for this background section. Modern deep learning libraries such as PyTorch and TensorFlow handle backpropagation computations, so backpropagation is rarely done manually. Nevertheless, intuitions on the learning process and network design can be built from understanding the fundamental backpropagation concepts.

4.1.3 Optimizers

The SGD update is considered the simplest optimizer. Modern optimization methods apply additional techniques that often improves convergence and overall performance for deep learning networks [56]. First is momentum, where parameters are updated from a pseudo velocity component. Inspired from a real physics perspective, this concept is similar to the concept of integrating a velocity to solve for position. In this case, the network's parameters can be considered the position and the gradients of these parameters only influence the velocity component. The most basic implementation of a momentum update is shown in Equations (4.10 - 4.11),

$$v^+ = \mu v^- - \gamma \partial x \tag{4.10}$$

$$x^+ = x^- + v^+ \tag{4.11}$$

where v is the pseudo velocity term, x generically represents the network parameters, ∂x is the parameter gradients, μ is a dampening constant that is typically set as a decimal value between 0 and 1, and γ is the learning rate. The key contribution for momentum is allowing x to still update even when ∂x is near zero, potentially escaping local minima. If gradients remain near zero over several iterations, the updates will gradually decay from the dampening term μ.

Another feature of modern optimizers is per-parameter adaptive learning rates. SGD uses a constant learning rate for all parameters, but optimizers that are able to use adaptive learning rates show significant gains over SGD [56]. The Adagrad optimizer is the first to propose per-parameter learning rates [57]. The Adagrad update is shown in Equations (4.12 - 4.26),

$$c^+ = c^- + \partial x^2 \tag{4.12}$$

$$x^+ = x^- - \gamma \frac{\partial x}{\sqrt{c} + \epsilon} \tag{4.13}$$

where c represents a cache of the squared gradients, and ϵ is a numerical stability term to avoid division by zero and is typically set to a low value. The cache term is used to normalize the incoming gradients element-wise which effectively reduces the learning rate for parameters that receive large gradients, and increases the learning rate for parameters that receive small gradients.

There are several different optimizers that apply momentum and adaptive learning rates either jointly or independently [56]. Distinct optimizers implement different variations of momentum and adaptive learning rates, but they are built on the basic implementations discussed in this section. This work uses the AdamW optimizer which is an adaptation of the Adam optimizer [58, 59]. AdamW applies both momentum and adaptive learning rates while decoupling weight decay regularization, showing improved performance over Adam [58].

4.1.4 Schedulers

Adaptive learning rates are powerful, but global learning rate schedulers have also been established to improve training stability and convergence [56]. A scheduler manages the global learning rate over training iterations or epochs and generally seeks to decrease the learning rate throughout training. This often stabilizes late-stage training and prevents large update steps while the network is in a stable condition. Some of the most commonly used schedulers include step decay, exponential decay, cosine

decay, and cosine decay with warm restarts [60]. Each of these schedulers are shown in Figure 4.4.

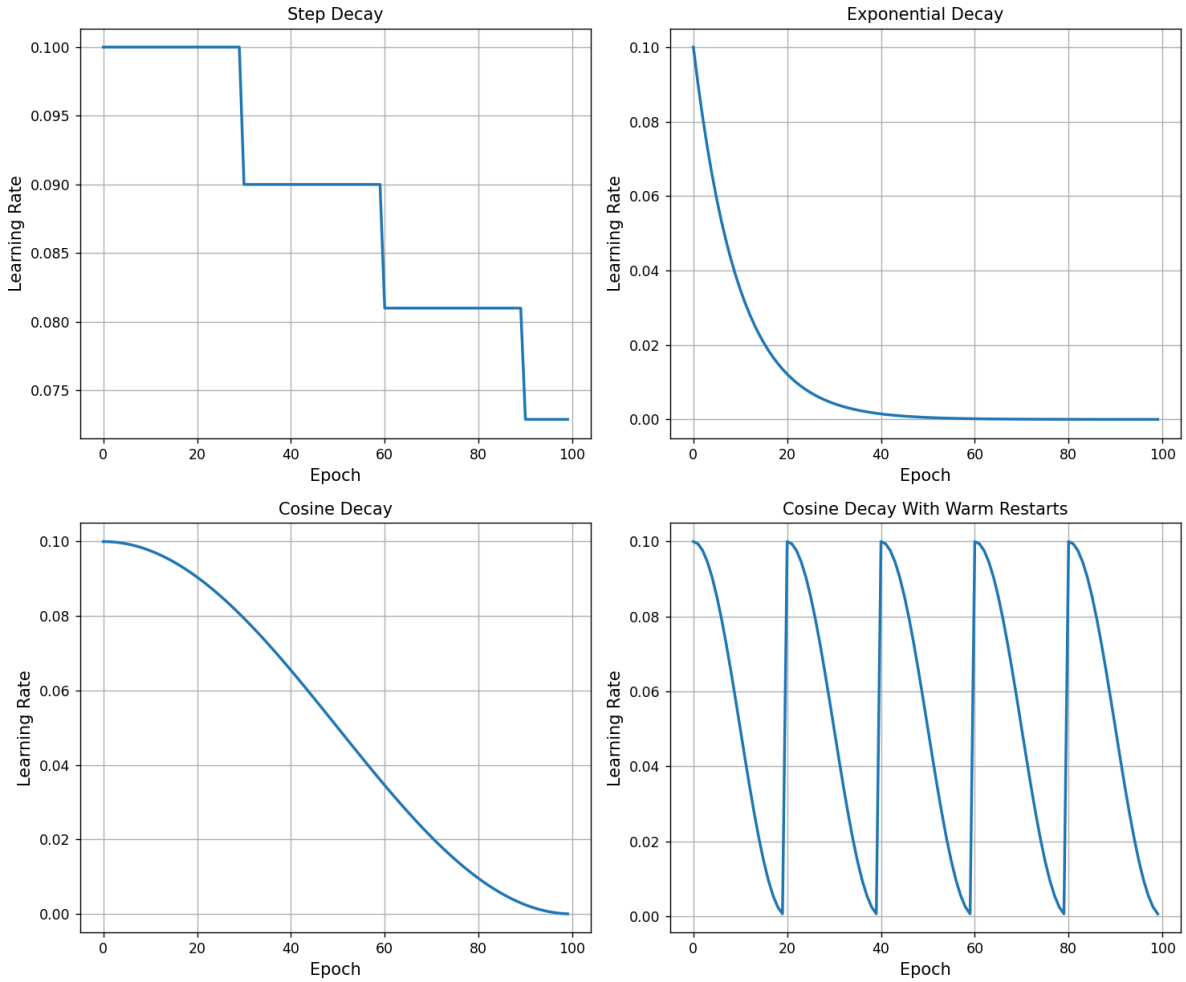


Figure 4.4: Common Learning Rate Schedulers

The schedulers shown are configured arbitrarily but have individual hyperparameters that define each of their dynamics. The desired behavior of each scheduler can be tuned using these hyperparameters, and the most optimal scheduler choice is generally dependent on the model, task, or data. The most appropriate selection is often unknown but can be made using hyperparameter optimization or manual trials.

4.1.5 Additional Design Techniques

Deep networks are a very active area of research and nuanced designs are constantly emerging. As such, this section briefly discusses additional techniques that are

relevant to the network design proposed in this thesis. These methods are considered standard practice and have been established to improve training stability, prevent overfitting, or improve network performance. For brevity, they are introduced in the following list with resources for additional reading:

- **Data preprocessing** are methods to manipulate the input data to account for biases and mismatched scales using the training data statisics. The two most common types for preprocessing are mean subtraction and z-score normalization [55].

- **L2/L1 regularization** is a technique to prevent overfitting by penalizing large weights through adding penalty terms to the loss value. L2 adds the sum of squared weights, $\lambda \sum w^2$, and L1 adds the sum of absolute weights, $\lambda \sum |w|$, where λ is the regularization factor [55].

- **Batch and layer normalizations** are methods to normalize activations between network layers and have been shown to produce more stable gradients and keep activation distributions more consistent across layers. Batch normalization applies normalization along the batch dimension, and layer normalization applies normalization along the feature dimension. Each of these two techniques have their own appropriate use case depending on goals and network architectures [61, 62].

- **Dropout** is another powerful and complementary regularization technique that sets random neuron activations to zero during training. Intuitively, this forces the network to learn with limited resources during training. It is defined by a probability hyperparameter that dictates the percentage of neurons that will return zero [63].

4.2 Core Network Architectures

This section introduces the core architectures used in the main proposed navigation network. The main network has a transformer backbone but also includes multilayer perceptrons (MLPs) as projection layers and predictions heads, and convolutional neural network (CNN) for image processing. Thus, these three core architectures are detailed in this section. Other network architectures such as recurrent neural networks (RNNs) or generative adversarial networks (GANs) are not discussed in this thesis since they are not used in the proposed navigation network. Readers are referred to [52] and [64] for additional resources on these types of networks.

4.2.1 Multilayer Perceptrons

Multilayer perceptrons (MLPs) are considered the simplest deep learning architectures. The architecture involves grouping neurons together in a layer and stacking multiple layers. Each layer in an MLP, otherwise known as a Fully-Connected layer (FC), linear layer, or Feed-Forward layer (FFW), can be classified as the input layer, a hidden layer, or the output layer. The input layer represents the input data in a vectorized form, hidden layers are any subsequent layers succeeding the input layer, and the output layer is the final layer of the MLP. Figure 4.5 illustrates an arbitrary MLP network.

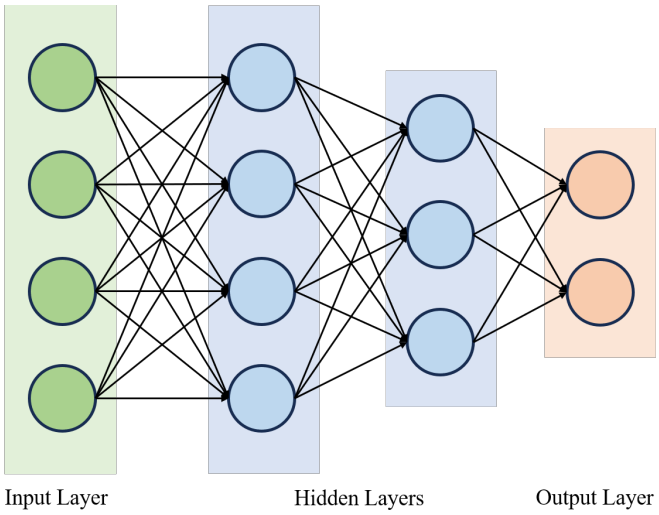


Figure 4.5: General MLP Architecture

Although not illustrated, activation functions, f, and dropout may be present between each layer. Similar to the single neuron example, activations, a^L, from the L^{th} layer can be computed by Equation (4.14),

$$a^L = f(xW^L + b^L) \tag{4.14}$$

where $x \in \mathbb{R}^{N^{L-1}}$ is the input from the $(L-1)^{th}$ layer, $W^L \in \mathbb{R}^{N^{L-1} \times N^L}$ and $b^L \in \mathbb{R}^{N^L}$ are the weights and biases in the current layer, and N represents the number of neurons.

The main MLP hyperparameters are the number of neurons for each layer, the activation functions between layers, and the total number of layers. Additional hyperparameters may include normalization layers and dropout layers. It is important to note that the main limitation of MLPs is that the input data is ingested in vectorized form. This can create major bottlenecks for multidimensional data structures such as images or point clouds. While it is possible to process multidimensional data with an MLP, it requires a flattening process, which converts the data into a single vector. This process can create extremely large input layers and corresponding initial layers, leading to computationally inefficient structures. A more suitable option for multidimensional data structures is a convolutional neural network which is discussed next.

4.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are very similar to MLPs. They include weights and biases as learnable parameters, compute a forward pass, and backpropagate from a computed loss as discussed in the previous section. The key difference is how CNNs process inputs and activations with multidimensional data structures. CNNs typically process data that can be described by two spatial dimensions and one depth dimension. The core CNN layers include convolutional, pooling, and fully-connected layers. A generic illustration of an arbitrary CNN architecture is shown in Figure 4.6.

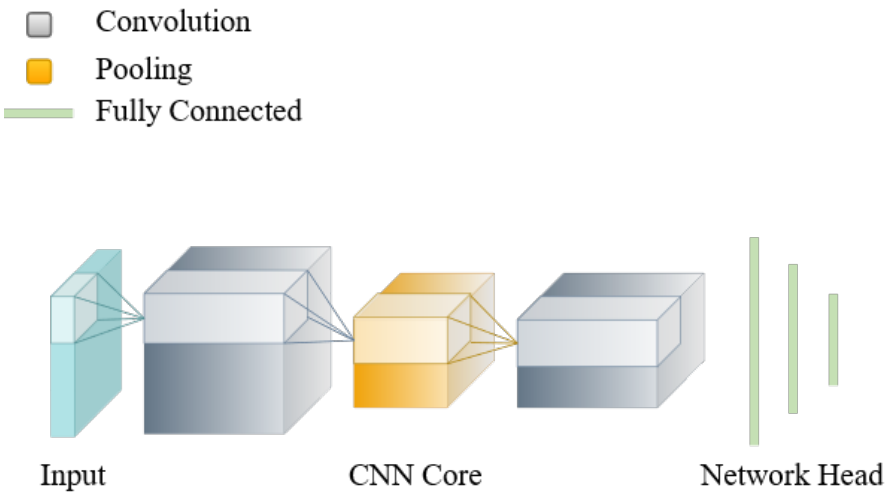


Figure 4.6: General CNN Architecture

Each layer has unique hyperparameters, operations, and purposes. The convolutional layer is the core building block of a CNN and is where the majority of the network's learnable parameters are computed. Given an arbitrary input, $x \in \mathbb{R}^{W^{L-1} \times H^{L-1} \times C^{L-1}}$ from the $(L-1)^{th}$ layer, convolutional operations are defined by K filters, $f \in \mathbb{R}^{F \times F \times C^{L-1}}$, a stride, $S \in \mathbb{R}_{\geq 0}$, and padding, $P \in \mathbb{R}_{\geq 0}$. Filters act as learnable parameters including weights and biases, the stride defines the increment which filters move or stride across the input, and padding, which is typically zero-padding, defines the number of borders to pad the input. These hyperparameters define the dimensions of the L^{th} layer which are given in Equations (4.15 - 4.17).

$$W^L = (W^{L-1} - F + 2P)/S + 1 \tag{4.15}$$

$$H^L = (H^{L-1} - F + 2P)/S + 1 \tag{4.16}$$

$$C^L = K \tag{4.17}$$

At a single stride instance, an output neuron from a convolution operation can be computed by Equation (4.18),

$$z^L_{i,j,k} = b^L_k + \sum_{c=0}^{C^{L-1}-1} \sum_{i'=0}^{F-1} \sum_{j'=0}^{F-1} (x^{'}_c \odot w^L_{c,k})_{i',j'}, \quad \forall k \in K \tag{4.18}$$

where:

- $x^{'}_c$ is a specific window of the padded input that is defined by the size of the filter.
- $w^L_{c,k}$ and b^L_k are the weights and biases of the k^{th} filter.
- i, j are the row and column indices of the output corresponding to the k^{th} filter.
- $i^{'}, j^{'}$ are the row and column indices of the resultant matrix from the Hadamard product, $\odot$, of the input and the weights.

Although the above expression may seem complex, it is very similar to the forward pass for an MLP. The convolution still involves taking a weighted sum using the inputs and weights and adding biases. The key difference is that this operation happens over windows across the input along all of its channels [65]. With this architecture, multidimensional data is processed much more efficiently than the flattening method required by an MLP.

Another typical layer type in a CNN is a pooling layer. A pooling layer's main purpose is to reduce the spatial size of an input while preserving its channels. Pooling layers are defined by a spatial extent, F, a padding, $P \in \mathbb{R}_{\geq 0}$ and a stride, $S \in \mathbb{R}_{\geq 0}$. These $F \times F$ layers stride across an input similar to convolutional layers, but work to reduce spatial information. This is typically done by taking the max or average of the

numerical elements within the pooling window. These operations are known as max pooling and average pooling. Since pooling layers directly manipulate the input, they do not require learnable parameters. Given the same arbitrary input as before, the dimensions of the L^{th} output layer following a pooling operation are given in Equations (4.19 - 4.21).

$$W^L = (W^{L-1} - F)/S + 1 \tag{4.19}$$

$$H^L = (H^{L-1} - F)/S + 1 \tag{4.20}$$

$$C^L = C^{L-1} \tag{4.21}$$

Finally, fully-connected layers are often included to create an MLP at the top of the CNN called the network head as shown previously in Figure 4.6. The network head often acts as a projection that transfers features from the final CNN layer to process the downstream task or for continual processing in other network architectures.

There are several design choices and considerations when constructing a CNN. For example, a network could stack several sequential convolutional layers before pooling or it could alternate between the two more frequently. A general practice is to minimize pooling in the earlier layers to maximize feature extraction potential from convolutional layers. Pooling is a destructive operation so pooling too often or early can cause excessive loss of information. Many state-of-the-art CNNs design convolutional and pooling layers to reduce spatial dimensions while increasing channels as layers progress [66–68].

4.2.3 Transformers

The transformer is the core backbone of the network architecture proposed in this thesis. Transformers are capable of efficiently processing sequential and multimodal data through its attention mechanism which is essential for fusing camera and IMU data. Transformers were originally designed to challenge traditional CNN-RNN based

sequence processing models. The original transformer model was introduced in 2017 by Vaswani et al. to propose a model that only uses the attention mechanism on machine translation tasks [69]. The model showed improved performance and efficiency over its CNN-RNN counterparts. Today, transformers are the foundational architectures for many popular large language models (LLMs) and vision language models (VLMs). Examples include GPT [70], BERT [71], Gemini [72], Stable Diffusion [73], among several others. A diagram of a transformer encoder using pre-layer normalization (Pre-LN) [74] is shown in Figure 4.7.

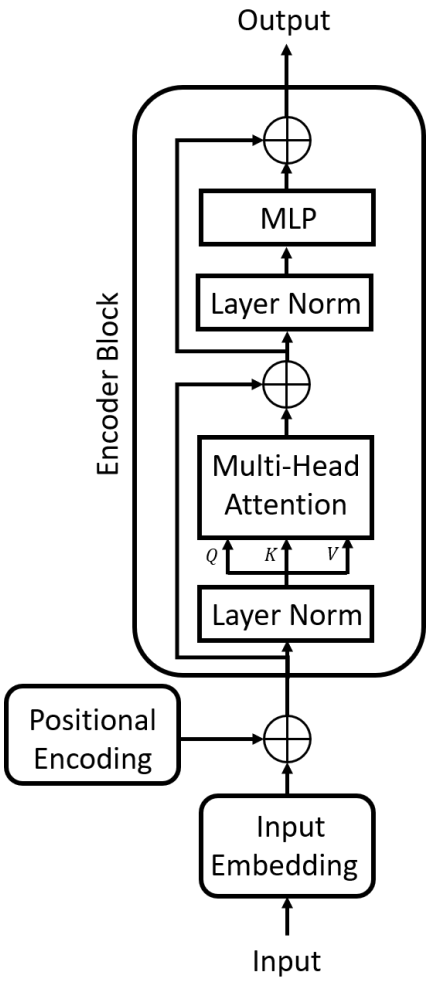


Figure 4.7: Transformer Encoder Block

Before discussing the transformer encoder, is important to understand how input data is processed. First, consider an input, $x \in \mathbb{R}^s$, where its elements may represent some arbitrary data, and s represents the sequence length of the data. A process

called input embedding is performed where each element is projected into vectors of a defined embedding dimension, d. For simple 1-dimensional, numerical inputs, a fully-connected layer can achieve this. For more complex inputs such as images, a CNN may be used to project them into the embedding space. Vision transformers (ViTs) propose splitting an image into spatial patches where each patch gets projected into the embedding space [75]. This thesis applies ViT patching to image inputs which is discussed in more detail in the network design section. The embedding processed is defined generically in Equation (4.22),

$$E^{raw} = \text{Embed}(x) \tag{4.22}$$

where $E^{raw} \in \mathbb{R}^{s \times d}$ are the embedding vectors that represent each sequence element in the original input. Learned positional encodings are then added to the raw embedding vectors which allows the model to determine the sequence location of each embedding. Equation (4.23) represents this process,

$$E = E^{raw} + E^{pos} \tag{4.23}$$

where $E^{pos} \in \mathbb{R}^{s \times d}$ are the positional encodings, and $E \in \mathbb{R}^{s \times d}$ are the final embedding vectors that are inputs into the transformer. These embeddings are then processed by an attention mechanism which is the core operation of a transformer block.

Transformer attention involves updating the values in each embedding vector using information from the other embedding vectors. For multi-head, self-attention, each embedding from the same input attend to themselves, capturing and contextualizing the meaning and importance of each element in the sequence [69]. The output of multi-head attention is described in Equation (4.24),

$$z = \mathop{\Big\|}_{h=1}^{H} (\text{softmax}(\frac{Q_h K_h^T}{\sqrt{d_k}})V_h) \tag{4.24}$$

where $Q_h \in \mathbb{R}^{s \times d_k}$, $K_h \in \mathbb{R}^{s \times d_k}$, $V_h \in \mathbb{R}^{s \times d_v}$ are the query, key, and value for each head, h, and $\|$ denotes the concatenation of the output from each head along the value dimension d_v. Note that although the query/key dimension and the value dimension may differ, they are often both set equal to the defined embedding dimension divided by the number of heads such that $d_k = d_v = d/H$ as in [69]. Thus, the final concatenated output will yield the same dimensions as the input embeddings such that $z \in \mathbb{R}^{s \times d}$. The query, key, and value are computed from learned weight matrices and the input embeddings as shown in Equations (4.25 - 4.27),

$$Q_h = EW_h^Q \tag{4.25}$$

$$K_h = EW_h^K \tag{4.26}$$

$$V_h = EW_h^V \tag{4.27}$$

where $W_h^Q \in \mathbb{R}^{d \times d_k}$, $W_h^K \in \mathbb{R}^{d \times d_k}$, and $W_h^V \in \mathbb{R}^{d \times d_v}$ are the query, key, and value weight matrices, respectively. Furthermore, the softmax operation in Equation (4.24) converts raw attention scores from the normalized query-key dot product into a probability distribution. This allows the model to determine relevant information from each embedding vector through probability scores. The softmax function is defined generically with an arbitrary vector, $x \in \mathbb{R}^n$, in Equation (4.28).

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{n} e^{x_j}} \tag{4.28}$$

Note that softmax is applied to the normalized query-key dot product along the key sequence dimension, thus operates on each individual raw attention score vector.

In addition, it is common to stack several transformer encoder blocks sequentially [69]. This is defined by a depth hyperparameter, D. In practice, the depth defines the number of sequential transformer blocks. The first transformer encoder receives the embeddings directly from the input and all subsequent encoders receive outputs from the previous transformer block.

For multimodal applications, such as the targeted IMU/camera fusion, a cross-attention transformer encoder can be used. The cross-attention encoder is very similar to the original encoder design except different modalities are used for the query, key and value. Specifically, one modality is used to compute the query input, while the other modality is used as the key and value inputs. An example diagram of the cross-attention process is shown in Figure 4.8.

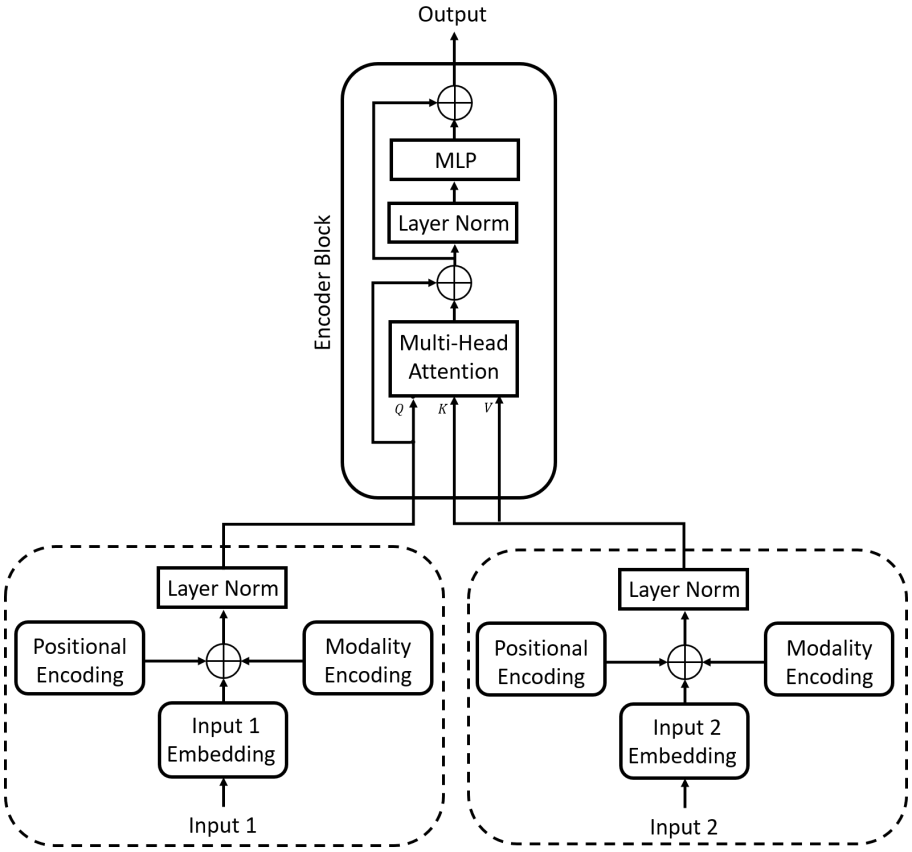


Figure 4.8: Cross-Attention Pipeline

Notice that the encoder block remains unchanged, and the only change is how the query, key, and value are determined from each modality. Furthermore, an additional learned modality encoding, which is similar to the positional encoding, is added to the embeddings to allow the model to learn which embeddings belong to which modality. The input embedding process for cross-attention can be described by Equation (4.29),

$$E = \text{Embed}(x) + E^{pos} + E^{mod} \tag{4.29}$$

where $E^{mod} \in \mathbb{R}^d$ is the learned modality encoding vector that is added to all the embeddings from an input source. Finally, it is important to note that each input will act as the query input while the others act as the key and value. As such, there is a cross-attention block dedicated to each modality where its outputs are embeddings that are contextualized from the other modalities.

Cross-attention is not the only method for fusing modalities, and multiple fusion methods have been explored [76]. Moreover, this section discusses a transformer encoder block, but omits the transformer decoder block. The decoder block is designed to autoregressively generate outputs such as text, audio, speech translations, or images, which is more commonly used for LLMs or VLMs [69]. Since the targeted downstream task for this work is to predict measurements, a decoder is unnecessary and is not discussed.

Chapter 5

Deep Learning, Vision-Based Dead Reckoning Network

This chapter presents a deep learning model that is designed to ingest sensor inputs and directly predict the navigation and state estimation outputs. This is a data-driven alternative that seeks to overcome the limitations of the previously discussed dead reckoning systems. First, an extension to the sensor suite is proposed. This work selects rear-viewing, side mirror cameras for several key reasons. First, this sensor selection does not violate the previous constraints to maintain a fully GNSS-independent navigation algorithm, and it avoids equipping sensors to the trailer. Second, cameras have a wide range of capabilities which can be leveraged to provide both navigation measurements and trailer hitch angle predictions. Finally, rear-viewing, side mirror cameras are increasingly being equipped on class 8 vehicles to enhance driver vision beyond traditional mirrors [77, 78]. The measurements from the full sensor suite thus includes the longitudinal velocity and steering angle available through the J1939 CAN bus, linear accelerations and angular rates a tractor-mounted IMU, and images from two tractor-mounted, rear-facing, side-mirror cameras. A simple schematic of the target sensor suite is illustrated in Figure 5.1.

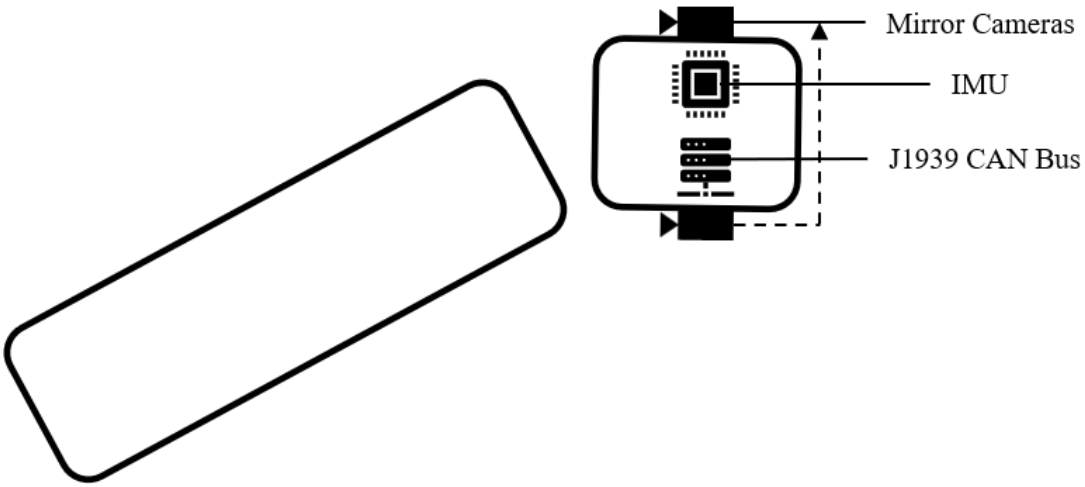


Figure 5.1: Proposed Sensor Suite

The deep learning network is heavily inspired from classical VIO systems, and is designed to simultaneously predict the relative motion of the tractor and the trailer hitch angle using camera and inertial data. Unlike the traditional methods discussed previously, the proposed network can handle the fusion and prediction pipeline without requiring knowledge of the system or sensor models. First, the proposed network design is presented. Second, the training paradigm for the network is discussed. Third, and finally, construction of navigation measurements during inference is covered.

5.1 Deep Learning VIO Network Design

This section discusses the full deep learning VIO network design. The model design is inspired from the classical VIO problem and previous deep learning VIO networks [29, 79]. As a result, the inputs to the network are consecutive images from the mirror cameras along with the IMU and CAN measurements taken in between the camera measurements. The outputs to the network are relative body-frame translations, Δx, Δy, the relative yaw rotation, $\Delta \psi$, and the trailer hitch angle γ. The network architecture is summarized in the diagram shown in Figure 5.2. The network is composed of individual phases that process or fuse the data for the VIO task. The hitch angle is predicted separately using a pretrained MobileNetV2 [68] that is finetuned during training. The individual phases of the network are detailed in the following subsections.

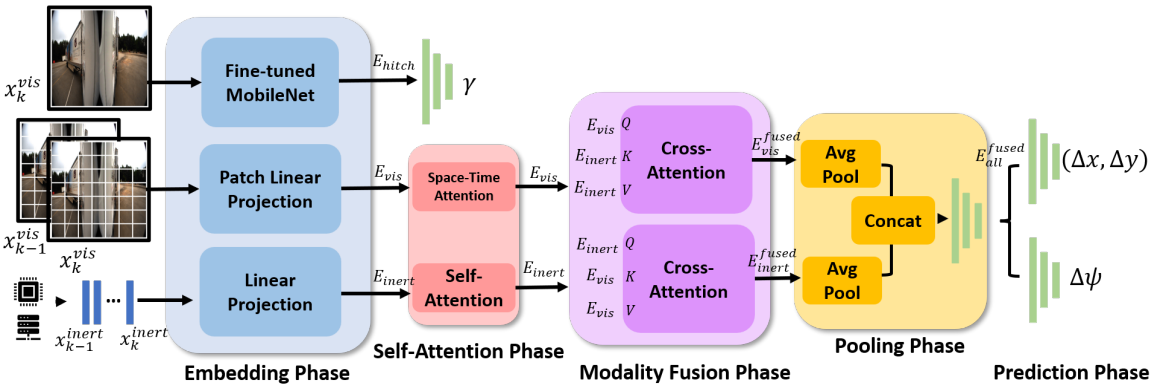


Figure 5.2: The Tractor-Trailer VIO Network Architecture.

Data Setup

Before the embedding phase, the raw data from the cameras, IMU, and J1939 CAN sensors are set as inputs into the network. First, the left and right mirror cameras are synchronized in time and the images from each camera are concatenated horizontally. This creates a single visual input, $x_{vis} \in \mathbb{R}^{2W_o \times H_o \times C_o}$, where W_o, H_o, and C_o are the width, height, and channels of the original images. The visual input is then resized to a defined spatial size, W, H. Similarly, the J1939 CAN measurements, which include the tractor's longitudinal velocity and the tire steer angle are synchronized with the IMU, and their measurements are concatenated to create a single inertial input, $x_{inert} \in \mathbb{R}^N$. Where $N = 8$ is the total number of individual sensors, and represents sensor readings of the 6-DOF IMU and the two J1939 CAN sensors at a single point in time.

To add temporal context, consecutive visual inputs and inertial inputs from times t_{k-n} and t_k are grouped and used as the inputs into the model, where n is a defined look back parameter. This creates a temporal dimension for each input, The final representation of visual and inertial inputs are described by Equations (5.1 - 5.2),

$$x_{vis} \in \mathbb{R}^{T_{vis} \times W \times H \times C} \tag{5.1}$$

$$x_{inert} \in \mathbb{R}^{T_{inert} \times N} \tag{5.2}$$

where T_{vis} and T_{inert} are the temporal dimensions of the visual and inertial inputs, respectively, and represent the sequence dimension for each input. It is important to note that the hitch angle is predicted using only the most current visual input, and a separate copy of the visual input at t_k is used for hitch angle predictions.

Embedding Phase

Each modality enters an embedding phase that projects the visual and inertial inputs into the embedding space, d. For the hitch angle task, this work follows [45] and uses a pretrained CNN with an augmented MLP head. The selected CNN is MobileNetV2 [68] pretrained on a popular classification dataset, ImageNet [80]. MobileNetV2 was selected as it was shown to be the best performing network despite being the smallest out of all other CNNs that were surveyed to predict hitch angle on a diverse simulated dataset [45]. The CNN weights are not frozen, and are finetuned during training. The entire pipeline for hitch angle predictions is described by Equation (5.3).

$$\gamma = \text{MLP}(\text{MobileNetV2}(x_k^{vis})) \tag{5.3}$$

Next, the visual and inertial inputs for the VIO pipeline must be embedded. The visual inputs for the VIO task undergo an embedding process that closely follows the ViT patch embedding process [75]. Each image is sliced into non-overlapping spatial patches of a defined size, P, where patches are represented by $x_p \in \mathbb{R}^{P \times P \times C}$, and $p = 1, ..., N_p$ is the spatial location of each patch. The resulting number of patches, N_p, is a function of the visual input's spatial dimensions and the patch size as shown in Equation (5.4).

$$N_p = \frac{WH}{P^2} \tag{5.4}$$

Each patch is then linearly projected into the embedding space, but the additional temporal dimension requires special handling of the embedding process. This work

follows the patch embedding process in [81] which handles temporally batched images for video understanding. For this process, each frame undergoes the original ViT patch embedding. The ViT patch embedding may be achieved by using a CNN with a filter size and stride that are equal to the patch size, and the number of filters equal to the embedding dimension, d. This process applies to all frames and each patch may be represented by $x_{p,t} =\in \mathbb{R}^{P\times P\times C}$ where $p = 1, \ldots, N_p$ and $t = 1, \ldots, N_t$ are the spatial and temporal locations of the patch, respectively. Lastly, learnable spatio-temporal positional encodings and modality encodings are added to the projected patches as shown in Equation (5.5),

$$E_{vis} = \mathsf{CNN}(x_{vis}) + E^{pos} + E^{mod}_{vis} \tag{5.5}$$

where $E_{vis} \in \mathbb{R}^{(T_{vis}\cdot N_p)\times d}$ is the final visual embedding, $E^{pos}_{p,t} \in \mathbb{R}^{(T_{vis}\cdot N_p)\times d}$ is the spatiotemporal positional encodings, and $E^{mod}_{vis} \in \mathbb{R}^d$ is the visual modality encodings. It is important to note that the sequence dimension of E_{vis} is the number of frames times the number of patches and careful selection of these parameters is necessary to avoid large sequence dimensions.

For inertial inputs, a simple fully connected layer is used to project all sensor readings into the embedding space, and learned positional and modality encodings are added to the embeddings as shown in Equation (5.6),

$$E_{inert} = \mathsf{FC}(x_{inert}) + E^{pos} + E^{mod}_{inert} \tag{5.6}$$

where $E_{inert} \in \mathbb{R}^{T_{inert}\times d}$ is the final inertial embedding, $E^{pos} \in \mathbb{R}^{T_{inert}\times d}$ is the spatiotemporal positional encodings, and $E^{mod}_{inert} \in \mathbb{R}^d$ is the inertial modality encodings. For the remaining phases, these embeddings are processed by various transformer encoders and retain their dimensionalities.

Self-Attention Phase

The next phase applies self-attention to the visual and inertial embeddings. The purpose of this phase is to let the network learn internal relationships within each modality. For visual embeddings, because each embedding represents a spatiotemporal patch within the entire visual input, this thesis follows [81] and applies divided space-time self-attention. The core idea of divided space-time attention is to decompose self-attention into two sequential steps which independently apply to the embedding's spatial and temporal dimensions. This approach reduces the time complexity from $O(T_{vis}^2 N_p^2)$ to $O(T_{vis} N_p^2 + T_{vis}^2 N_p)$, where the former would be the time complexity of regular self-attention. Divided space-time attention is achieved by simply rearranging the dimensions of the visual embeddings and sequentially applying independent self-attention encoders [81]. Table 5.1 details the required embedding shapes and time complexities for each attention type. It is important to note that after the visual embedding is processed in the space-time attention block, it must be reshaped to its original dimensions, $E_{vis} \in \mathbb{R}^{(T_{vis} \cdot N_p) \times d}$.

Table 5.1: Spatial and Temporal Attention Descriptions

Attention Type	**Embedding Reshape**	**Time Complexity**
Spatial	$E_{vis} \in \mathbb{R}^{T_{vis} \times N_p \times d}$	$O(T_{vis} N_p^2)$.
Temporal	$E_{vis} \in \mathbb{R}^{N_p \times T_{vis} \times d}$	$O(T_{vis}^2 N_p)$.

For the inertial embeddings, regular self-attention is applied since the sequence dimension is simply T_{inert}. The process of the Self-Attention Phase for each visual and inertial embeddings are generically described by Equations (5.7 - 5.8).

$$E_{vis} = \text{Space-Time-Attention}(Q(E_{vis}), K(E_{vis}), V(E_{vis})) \tag{5.7}$$

$$E_{inert} = \text{Self-Attention}(Q(E_{inert}), K(E_{inert}), V(E_{inert})) \tag{5.8}$$

Of note, the general Self-Attention expressions in these equations refer to the full transformer encoder previously described in Chapter 4.

Modality Fusion Phase

Next, each modality embedding cross-attends to the other in the modality fusion phase. The purpose of this phase is to fuse information between the visual and inertial embeddings. This is done using cross-attention encoders where each modality acts as the query while the other modality acts as the key and value. This process yields visual and inertial embeddings that are contextualized with respect to each other by having embeddings from each modality attend to each other. Equations (5.9) and (5.10) represent this process.

$$E_{vis}^{fused} = \text{Cross-Attention}(Q(E_{vis}), K(E_{inert}), V(E_{inert})) \tag{5.9}$$

$$E_{inert}^{fused} = \text{Cross-Attention}(Q(E_{inert}), K(E_{vis}), V(E_{vis})) \tag{5.10}$$

Again, the general Cross-Attention expressions in the following equations refer to the full cross-attention encoder previously described in Chapter 4.

Pooling Phase

Following modality cross-attention, the pooling phase aims to prepare the individual visual and inertial embeddings for target prediction. This phase applies average pooling to each modality embedding along the sequence dimension. This yields the pooled embeddings $E_{vis}^{pooled} \in \mathbb{R}^d$ and $E_{inert}^{pooled} \in \mathbb{R}^d$. The pooled embeddings are then concatenated to a single vector, $E_{all}^{pooled} \in \mathbb{R}^{2d}$. Finally, an MLP ingests this vector and maps it back into the embedding space, d. The final result is the complete, fused embedding, $E_{all,fused} \in \mathbb{R}^d$ that is used as an input in the prediction phase.

Prediction Phase

Finally, the body frame longitudinal and lateral translations, and yaw rotation, are predicted by individual MLP prediction heads. A single head is used for both translation

predictions since they are related tasks, and a separate head is used for the rotation prediction. This is represented by Equations (5.11 - 5.12)

$$(\Delta x, \Delta y) = \text{MLP}(E_{all}^{fused}) \tag{5.11}$$

$$\Delta\psi = \text{MLP}(E_{all}^{fused}) \tag{5.12}$$

Network Construction In PyTorch

The proposed network and all its components are constructed using the Python deep learning library, PyTorch [82]. PyTorch includes tools for building each of the phases discussed, where a user can define all components and their necessary hyperparameters. Moreover, the training process uses PyTorch tools to handle back-propagation computations and parameter updates, given user defined learning rates, optimizers, and schedulers. In this thesis, the specific hyperparameters vary depending on the simulation or experimental design. Therefore, the parameters that are used for each network rendition are detailed later in Chapter 6.

5.2 Training Paradigm

The training paradigm used for this network is augmented from the trivial training approach to efficiently improve VIO performance. Regarding the temporal look back parameter, n, it was found that larger look backs naturally improve VIO performance because it provides broader temporal context. Consequentially, applying a larger look back increases inference time which is not ideal for real-time performance.

As an alternative approach, a larger temporal look back is handled in the training loop while the model only uses a look back parameter of $n = 1$. This exposes the model to a larger temporal window through the training process while keeping the model architecture minimal. This augmentation requires special handling in the data loading process and an internal loop within the original training loop that iterates through a temporal window. Inside the internal loop, regular forward passes and

loss computations are performed, and the average of the losses computed are taken and used to backpropagate through the network. Furthermore, an additional loss is computed for the accumulated yaw angle from delta yaw rotations at the end of the temporal window to further enforce reliable yaw predictions. The full training loop is described in Algorithm 1.

Algorithm 1 Training Loop

for $epoch = 0$ in $epochs$ **do**
  $L_{\text{trans}} \leftarrow \emptyset$ {Initialize loss accumulators}
  $L_{\text{rot}} \leftarrow \emptyset$
  $\psi_{\text{accum}} \leftarrow 0.0$ {Initialize accumulated yaw}
  for $n = 0$ in N **do**
    $(\Delta x, \Delta y),\ \ \Delta\psi,\ \ \gamma \leftarrow \text{model}(\text{inputs}(n))$ {Forward pass and compute losses}
    $l_{\text{trans}} = \text{MSE}((\Delta x, \Delta y), ((\Delta x_{\text{target}}, \Delta y_{\text{target}})))$
    $l_{\text{rot}} = \text{MSE}(\Delta\psi, \Delta\psi_{\text{target}})$
    $L_{\text{trans}} \leftarrow L_{\text{trans}} \cup \{l_{\text{trans}}\}$ {Populate accumulators}
    $L_{\text{rot}} \leftarrow L_{\text{rot}} \cup \{l_{\text{rot}}\}$
    $\psi_{\text{accum}} = \psi_{\text{accum}} + \Delta\psi$
  end for
  $L_{\psi_{\text{accum}}} = \text{MSE}(\psi_{\text{accum}}, \psi_{\text{target}})$ {Compute additional loss for resultant yaw angle}
  $L_{\text{total}} = \frac{1}{N}\sum^{N} L_{\text{trans}} + \frac{1}{N}\sum^{N} L_{\text{rot}} + L_{\psi_{\text{accum}}}$
  backpropagate(L_{total})
end for

5.3 Inference

During inference, the network ingests two consecutive images and all IMU and CAN measurements in between the image measurements. In this work, images are received at 10 Hz while IMU and CAN measurements are received at 40 Hz. Given these sensor rates, two images and five IMU and CAN readings are processed at every 0.1 second interval. The outputs of the VIO network represent the change in translation and heading orientation over the 0.1 second interval. Given accurate initialization, the absolute North, East, and yaw measurements can be computed from the changes in motion. Equations (5.13 - 5.14) define the construction of the North position, East position, and yaw angle, respectively.

$$\tilde{N}_{k+1} = \tilde{N}_k + \sin(\tilde{\psi}_k)\Delta\tilde{x} + \cos(\tilde{\psi}_k)\Delta\tilde{y} \tag{5.13}$$

$$\tilde{E}_{k+1} = \tilde{E}_k + \cos(\tilde{\psi}_k)\Delta\tilde{x} - \sin(\tilde{\psi}_k)\Delta\tilde{y} \tag{5.14}$$

$$\tilde{\psi}_{k+1} = \tilde{\psi}_k + \Delta\tilde{\psi} \tag{5.15}$$

The inference time were also computed as an average of inference times over 1000 samples. This was computed on a PC with an AMD Ryzen 5 3600X 6-Core Processor and an NVIDIA GeForce RTX 2060. Note that for the sensor rates used in this work, the network must run at least 10 Hz for real-time performance. The average inference time when running the network entirely on the processor is approximately 0.03 second which supports a 29 FPS image stream. The average inference time when running the network on the GPU is approximately 0.028 second which supports a 35 FPS image stream. Note that the hardware used to compute these times are not considered low-cost hardware. Depending on the hardware components, these findings may vary. Ultimately, the network is able to run at real-time speeds on the CPU or the GPU given the tested hardware.

Chapter 6

Simulation and Experimental Results

In this chapter, the performance of an INS, model-based EKF, and deep learning VIO (DL-VIO) network are evaluated and compared in simulations and experiments. The INS is used only as a baseline to better compare the two alternative dead reckoning systems. Simulation studies cover the performance of each system using a controlled and customized simulation dataset designed to capture realistic scenarios. Then, experiments are performed to evaluate and compare the model-based EKF and the DL-VIO on a collected dataset which features both rural and urban environments around Auburn and Opelika, Alabama. The proceeding sections discuss the simulation and experimental setup and results in detail.

6.1 Simulations

The simulation studies aim to evaluate and compare an INS, model-based EKF, and DL-VIO network without GNSS through a series of Monte Carlo simulations. The Monte Carlo simulation studies were chosen to fully evaluate the performance of each dead reckoning system given realistic error sources. This section discusses the simulated data generation, the DL-VIO training and validation results for simulated data, and the Monte Carlo simulation setup and results

6.1.1 Simulation Data Generation

The simulation environment is built upon TruckSim® [83], RoadRunner [84], and Simulink® [85]. TruckSim is a professional simulation environment designed to configure and simulate high-fidelity vehicle dynamics for heavy truck models, and it is used

to generate ground truth signals and IMU and CAN measurements. RoadRunner is an MathWorks® application to generate traffic environments, and is used to generate custom scenes for diverse image data. Finally, Simulink connects these applications and produces the simulated data. A diagram of the simulation setup is shown in Figure 6.1.

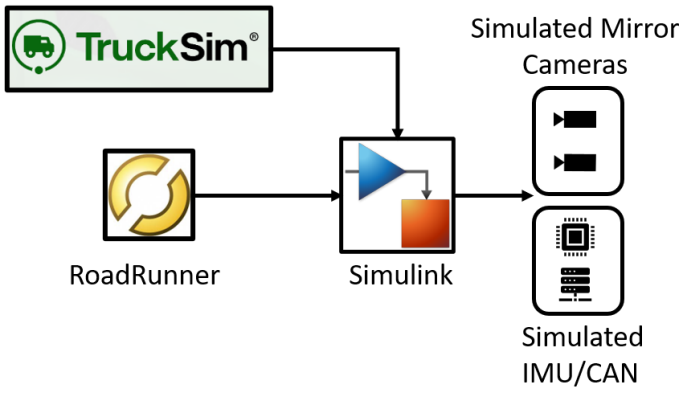


Figure 6.1: Simulation Application Setup

The simulated camera data are rendered in Simulink using scenes from RoadRunner and Simulink's 3D Animated Toolbox. Each mirror camera is simulated independently at a resolution of 1080×1280. Examples of the simulated camera data is shown in Figure 6.2.

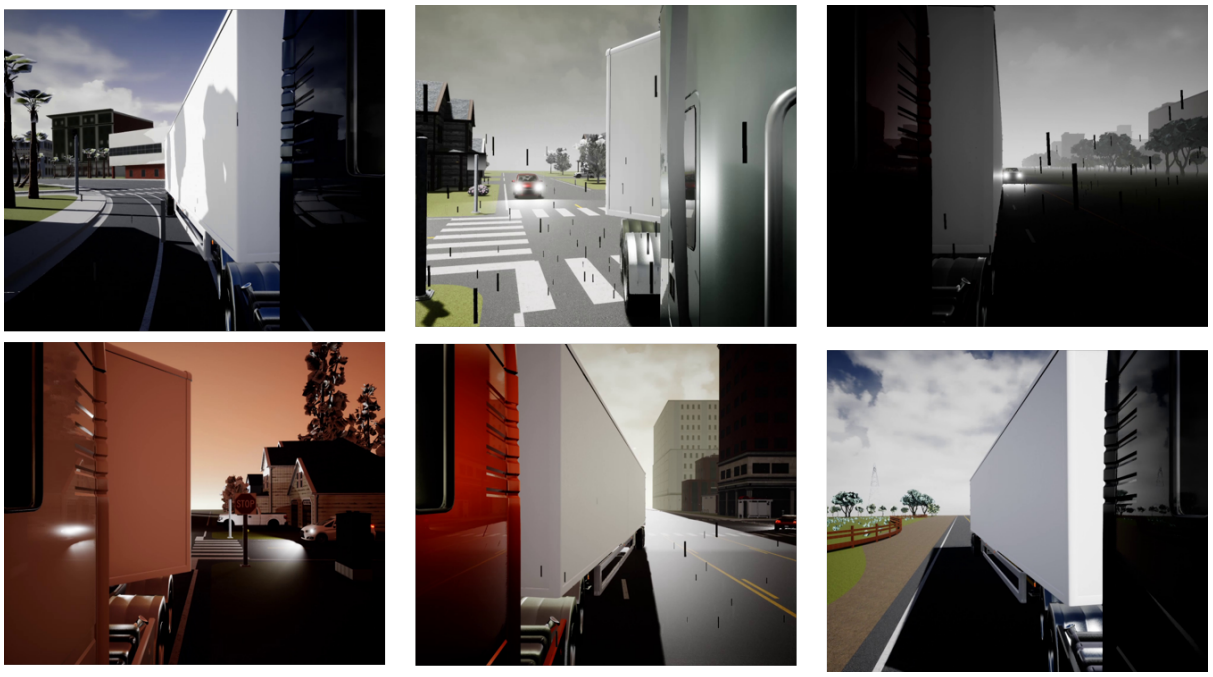


Figure 6.2: Simulated Camera Data Examples

For the purpose of training the DL-VIO network, the dataset is composed of individual sequences, each with unique trajectories and scene conditions. The sequences are split among a training set, a validation set, and a testing set. The training set includes approximately 17K data samples, the testing set includes approximately 13K data samples, and the validation set is a random 1.5K subset of the testing set. Furthermore, each sequence is designed to incorporate both time variations (e.g., day, night, and mixed) and environment variations (e.g., rural, urban, and mixed). Mixed tags represent ambiguous time and environment classifications (e.g., sunsets, overcast, rain, and mixed rural and urban scenes.). The variations for each category are represented in Figure. 6.3.

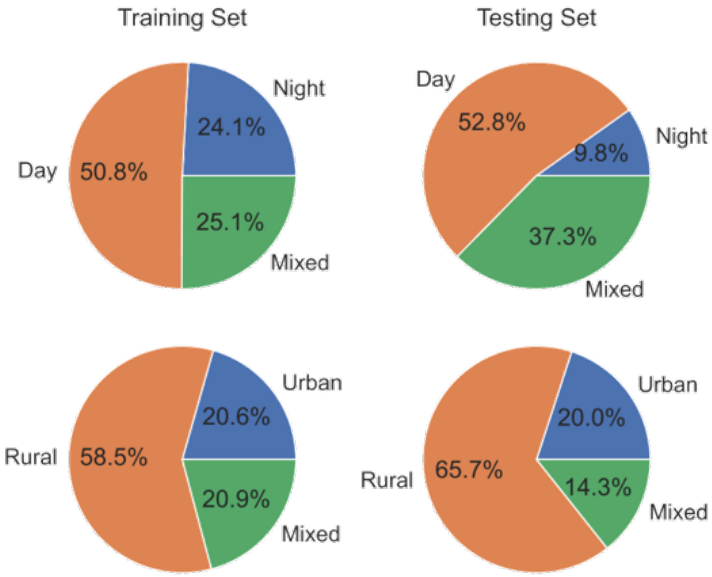


Figure 6.3: Time and Setting Variations. **Left:** Training Set. **Right:** Testing Set.

6.1.2 Simulation DL-VIO Parameters and Training Results

This section presents the DL-VIO model and training parameters, and its results over the simulation set. The training procedure follows the methods discussed in the previous chapter with the parameters detailed in Table 6.1. The selected learning rate warmup and scheduler produces the learning rate shown in Figure 6.6.

Table 6.1: Simulation DL-VIO Model and Training Parameters

Model Parameters	
Image Size	224×224
Patch Size	16
Embedding Dimensions	384
Heads	8
Depth	8
Training Parameters	
Epochs	30
Temporal Training Window	2
Batch Size	6
Learning Rate	3×10^{-5}
Loss Function	MSE
Optimizer	AdamW ($\beta_1 = 0.9,\ \ \beta_2 = 0.999,\ \ \gamma = 0.05$) [58]
Warmup	Linear (2 Epochs)
Scheduler	Cosine Decay with Warm Restarts ($2\times$ Decay after restarts)

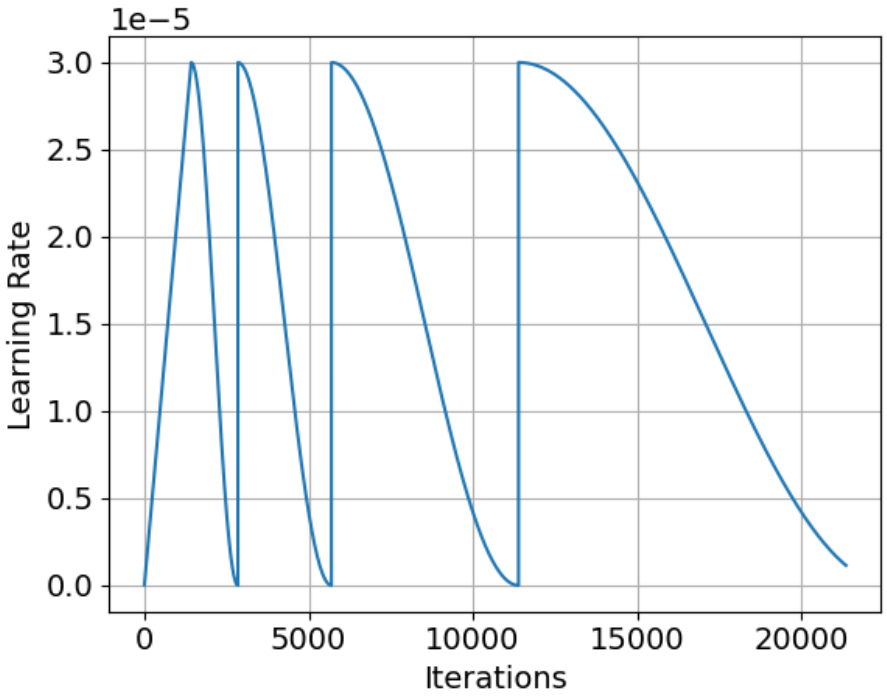


Figure 6.4: Simulation VIO Learning Rate

The training and validation RMSEs of the DL-VIO's translations and rotation outputs are shown in Figure 6.7. The results show that the training and validation RMSEs decay at approximately the same rates and magnitudes, which indicates that the network is able to learn necessary patterns from the training data with minimal overfitting. Furthermore, the dynamics of the RMSEs over the course of training follows a similar pattern to the learning rate dynamics. This is expected and is the intention of cosine decays and warm restarts where errors may periodically increase with each restart but decrease over the entire training period.

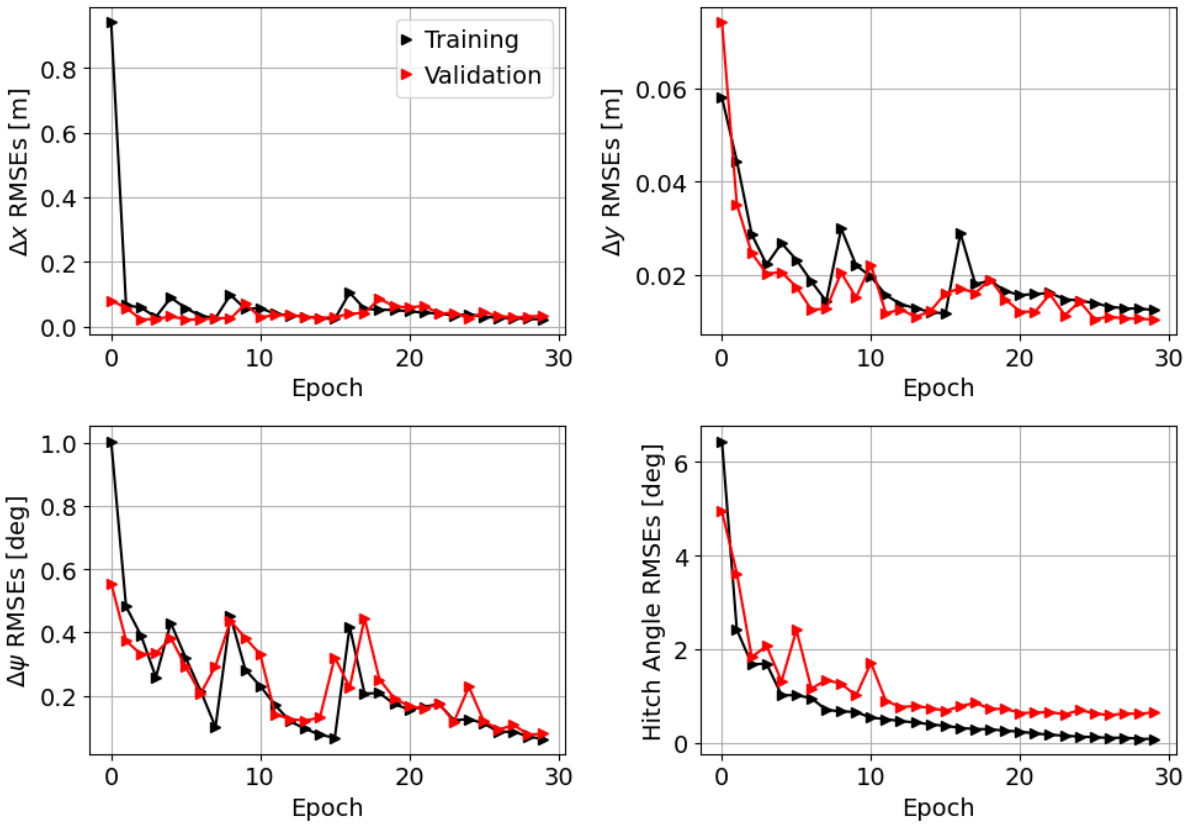


Figure 6.5: The training and validation RMSEs of the VIO network's outputs on simulation data.

6.1.3 Monte Carlo Simulations

Simulation Setup

Several Monte Carlo simulations are evaluated to test the performance of each dead reckoning system. For each case study and individual system, 100 Monte Carlo runs are performed. To capture the effects of various error sources each system may be subjected to, the stochastic variables for the simulations are vehicle model parameters, and IMU errors. Two sequences with distinct dynamics are selected from the testing set to be evaluated.

Vehicle parameters may vary up to defined percentage of their true values. Each parameter is then randomly sampled from a uniform distribution within their respective ranges. For a highly uncertain vehicle model, all parameters were varied up to 50% of their true values, and for a well known vehicle model, parameters were varied up to 5%

of their true values. Table 6.2 shows the nominal vehicle parameter values and their ranges.

Table 6.2: Vehicle Parameter True Values and Ranges

Parameters	Nomial	Low Uncertainty	High Uncertainty	Units
C_1	4×10^5	$(4.2 \times 10^5$ - $3.8 \times 10^5)$	$(6 \times 10^5$ - $2 \times 10^5)$	N/rad
$C_{2,3}$	2.25×10^5	$(2.138 \times 10^5$ - $2.362 \times 10^5)$	$(1.125 \times 10^5$ - $3.375 \times 10^5)$	N/rad
$C_{4,5}$	1.2×10^5	$(1.14 \times 10^5$ - $1.26 \times 10^5)$	$(6 \times 10^4$ - $1.8 \times 10^5)$	N/rad
m_{s1}	6493	(6168 - 6818)	(3246 - 9740)	kg
$m_{us,A1}$	570	(541.5 - 598.5)	(285 - 855)	kg
m_{us-A23}	785	(745.8 - 824.2)	(392.5 - 1178)	kg
a	1.384	(1.315 - 1.453)	(0.692 - 2.076)	m
b_1	3.616	(3.435 - 3.797)	(1.808 - 5.424)	m
b_2	4.886	(4.642 - 5.13)	(2.443 - 7.329)	m
c	4.251	(4.038 - 4.464)	(2.126 - 6.377)	m
R_{z1}	1.74	(1.653 - 1.827)	(0.87 - 2.61)	m
m_{s2}	3196	(3036 - 3356)	(1598 - 4794)	kg
$m_{us,A45}$	665	(631.8 - 698.2)	(332.5 - 997.5)	kg
d	7.303	(6.938 - 7.668)	(3.651 - 10.95)	m
f_1	13.124	(12.47 - 13.78)	(6.562 - 19.69)	m
f_2	14.412	(13.69 - 15.13)	(7.206 - 21.62)	m
R_{z2}	7.505	(7.13 - 7.88)	(3.752 - 11.26)	m

The IMU measurements are simulated using true linear accelerations and angular velocities reported from TruckSim® and adding a first-order Gauss-Markov (FOGM) bias and additive zero-mean white noise as shown in Equation (6.1),

$$\tilde{z}_k = z_k + b_k + \nu_k \tag{6.1}$$

where $\tilde{z}_k$ represents the accelerometer and gyroscope measurements, z_k represents the true signals, $\nu_k \sim \mathcal{N}(0, \sigma^2_{GWN})$ is additive GWN. The FOGM bias is defined in Equation (6.2).

$$b_k = e^{\frac{\Delta t}{\tau}} b_{k-1} + w_k \tag{6.2}$$

Where Δt is the measurement interval, τ is the correlation time of the bias, and $w_k \sim \mathcal{N}(0, \sigma^2_{GM})$ is the driving white noise of the bias defined by the Gauss-Markov variance shown in Equation (6.3).

$$\sigma^2_{GM} = \sigma^2_{ss}(1 - e^{-2\Delta t/\tau}) \tag{6.3}$$

Where σ^2_{ss} is the steady-state variance.

The simulated IMU errors parameters are chosen to emulate a low-cost IMU that represent common consumer-grade or automotive grade IMUs which are typically installed on ground vehicles. The simulation parameters are shown in Table 6.3. Note that these parameters apply across all accelerometer and gyroscope axes.

Table 6.3: Simulated IMU Parameters

Parameters	Automotive Grade Values	Units
τ_a	300	s
$\sigma_{ss,a}$	5×10^{-3}	m/s^2
$\sigma_{GWN,a}$	8×10^{-3}	m/s^2
τ_g	300	s
$\sigma_{ss,g}$	1.75×10^{-3}	rad/s
$\sigma_{GWN,g}$	7×10^{-4}	rad/s

Low Dynamic Scenario Results

First, a low dynamic scenario is studied. A short sequence of approximately 20 seconds performing two sharp turns is selected. The average driving speed for the sequence is approximately 8.3 m/s or 18 mph. Figure 6.6 shows the sequence trajectory, and Figure 6.7 shows the input steering angle, longitudinal velocity, yaw angle, and hitch angle for the maneuver.

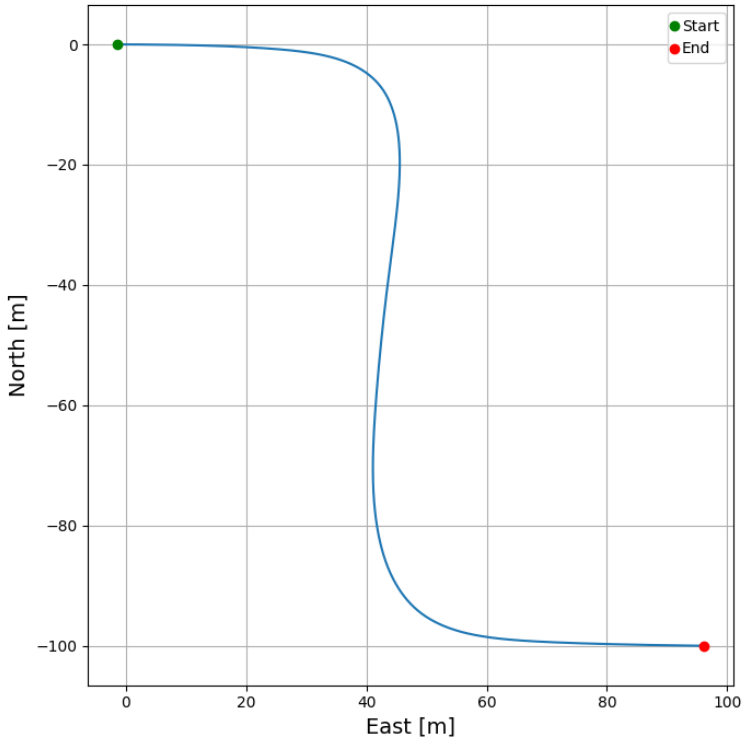


Figure 6.6: Low dynamic sequence North and East trajectory.

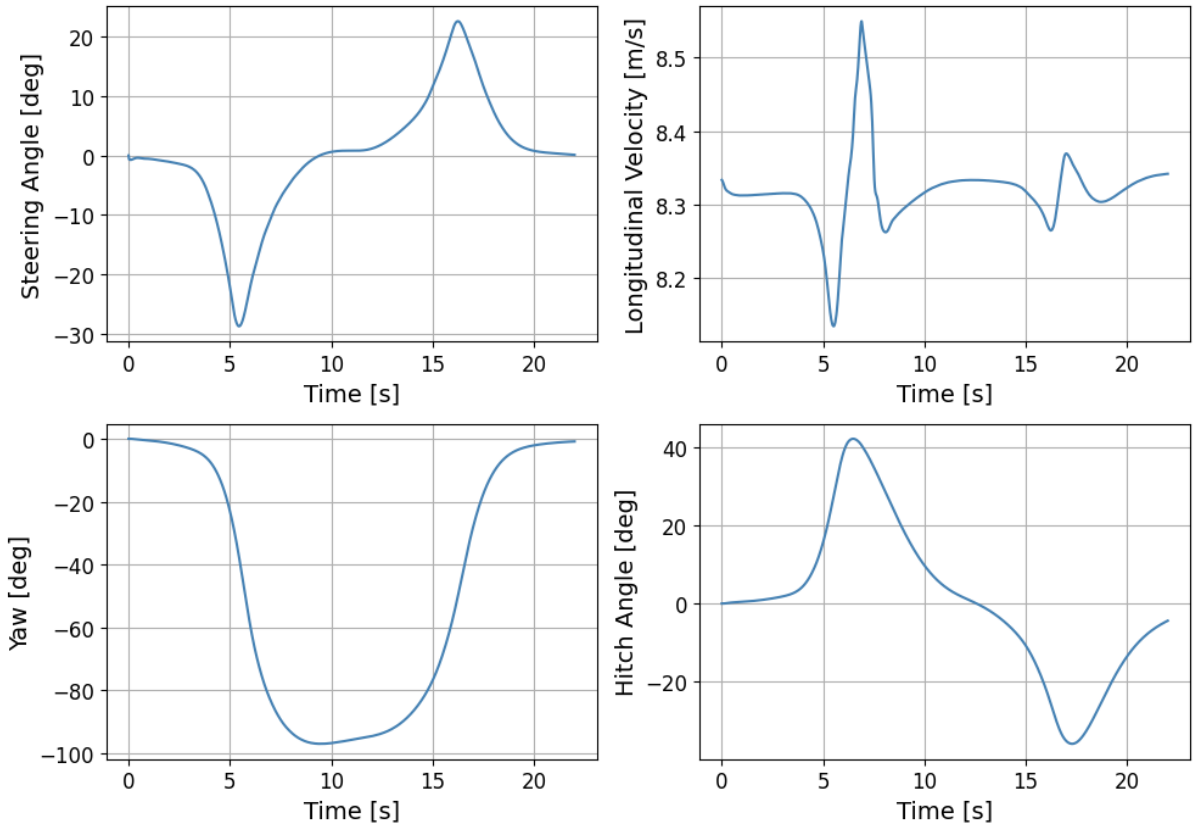
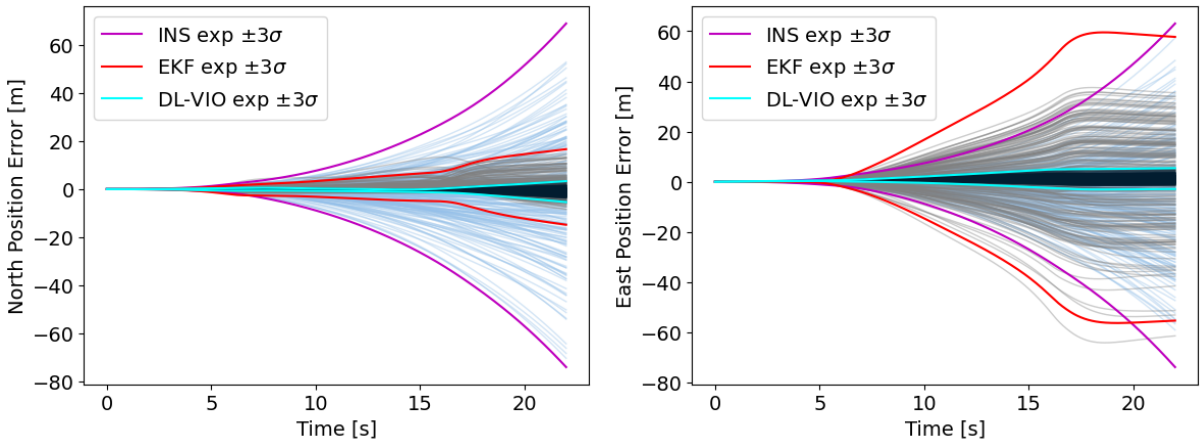


Figure 6.7: Low dynamic sequence steering angle, longitudinal velocity, yaw Angle, and hitch Angle

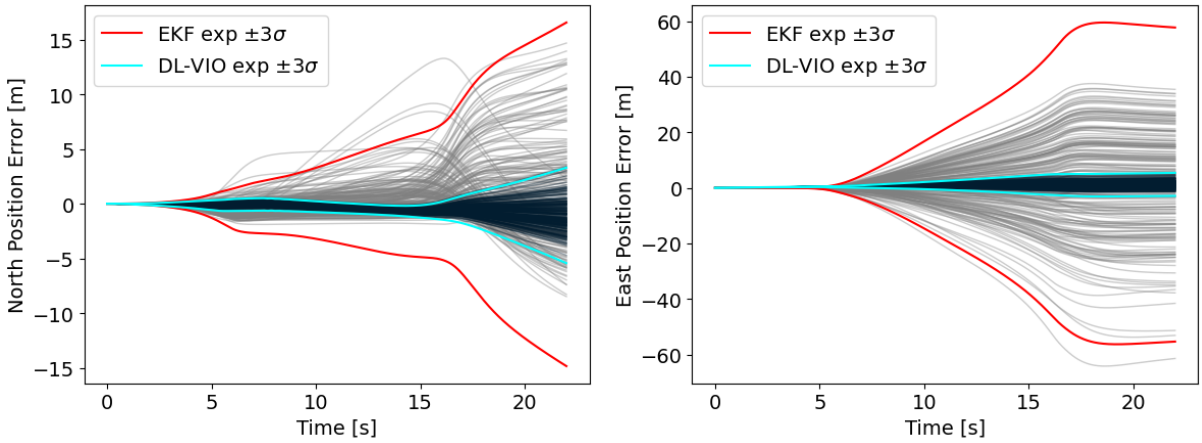
The first case study that is evaluated is the high uncertainty model parameters case. Figure 6.8 shows the North and East position errors from each system. Since the INS position errors are likely much greater than the other systems, the position errors from the model EKF (referred to as EKF) and the deep learning VIO (referred to as DL-VIO) are isolated and shown in Figure 6.9. Lastly, the yaw angle and hitch angle errors are shown in Figure 6.10. For this test case, the results show that the DL-VIO significantly outperforms the other systems in positioning, orientation, and hitch angle estimates. It is shown, that in the presence of model uncertainty, the EKF solutions are highly inaccurate. This is shown in the estimate errors for all of the states, and is especially evident in Figure 6.10 where the yaw and hitch angle errors are much larger than the other systems. The EKF is still able to outperform INS in positioning, but this is expected since low-cost, INS-based position errors are expected to be large from double integration of errors. The DL-VIO shows the lowest positioning errors with maximum errors falling below 5 and 10 meters for North and East positions, respectively.



(a) North Error

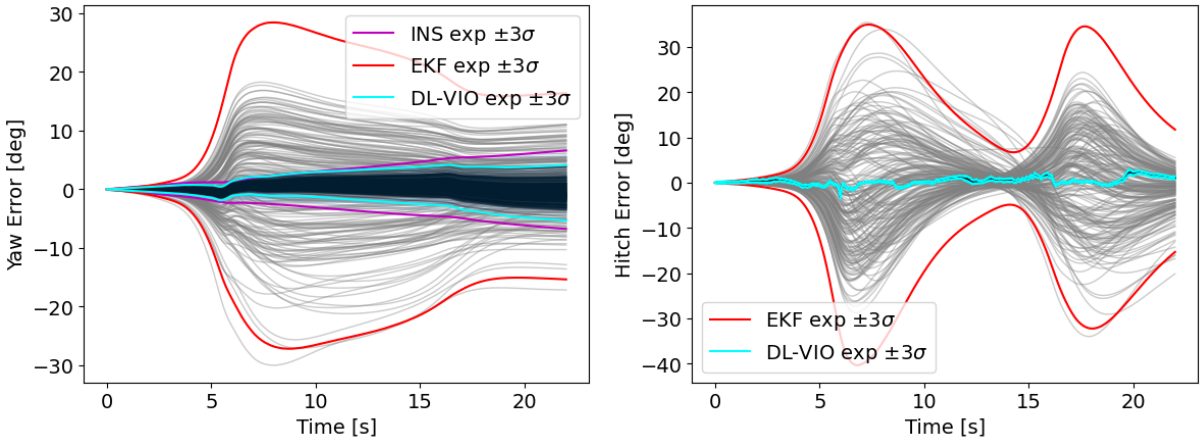
(b) East Error

Figure 6.8: Low dynamics sequence with high model uncertainty: North and East position errors



(a) North Error (b) East Error

Figure 6.9: Low dynamics sequence with high model uncertainty: EKF and DL-VIO North and East Position Errors

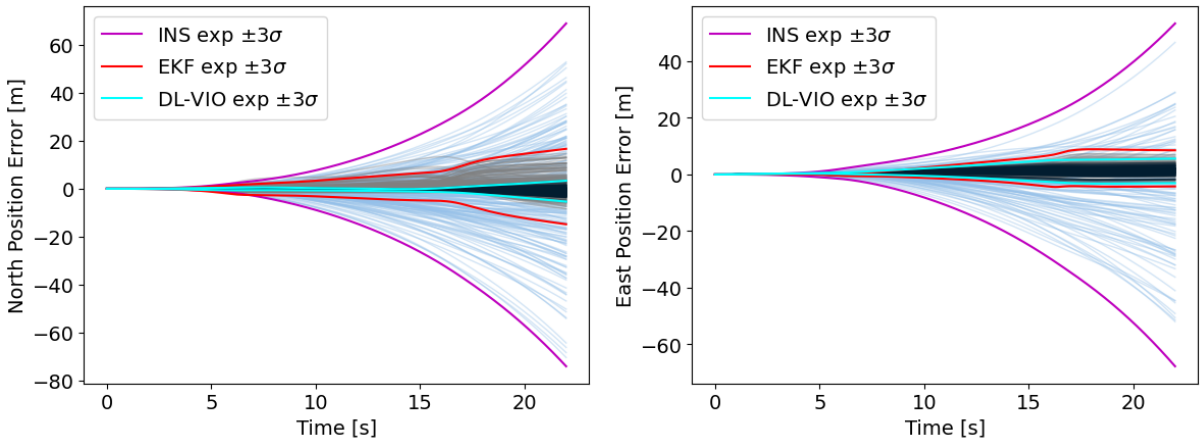


(a) Yaw Error (b) Hitch Error

Figure 6.10: Low dynamics sequence with high model uncertainty: Yaw and Hitch Angle Errors

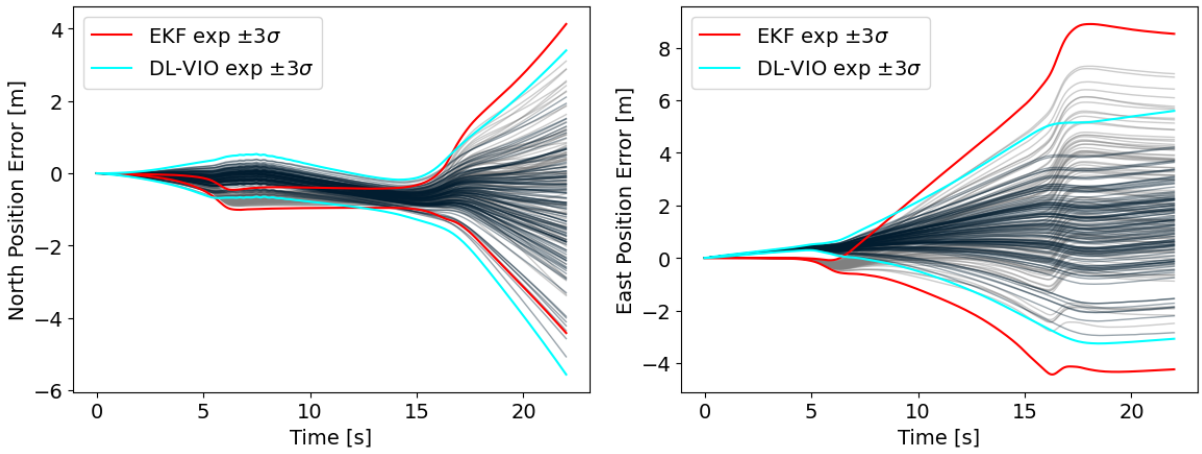
Next, the results are reevaluated for the case of low model uncertainty. These results represent a case where model parameters are known reasonably well. Note that since model parameters only affect the EKF solution, the only expected changes are for the EKF solution. Figure 6.11 shows the North and East position errors for the three systems, Figure 6.12 shows positions errors from the EKF and DL-VIO only, and Figure 6.13 shows the yaw and hitch angle errors. As expected, the performance of the EKF significantly improves. Position and yaw errors become more similar to those of the DL-VIO, reducing by tens of meters between the North and East positions,

and by over 10 degrees for yaw angle. However, Figure 6.13b shows that the DL-VIO is still able to predict the trailer hitch angle better than the EKF even with low model uncertainty. This highlights that even with a reasonably well known model, small parameter errors may cause large navigation and state estimation errors for the EKF.



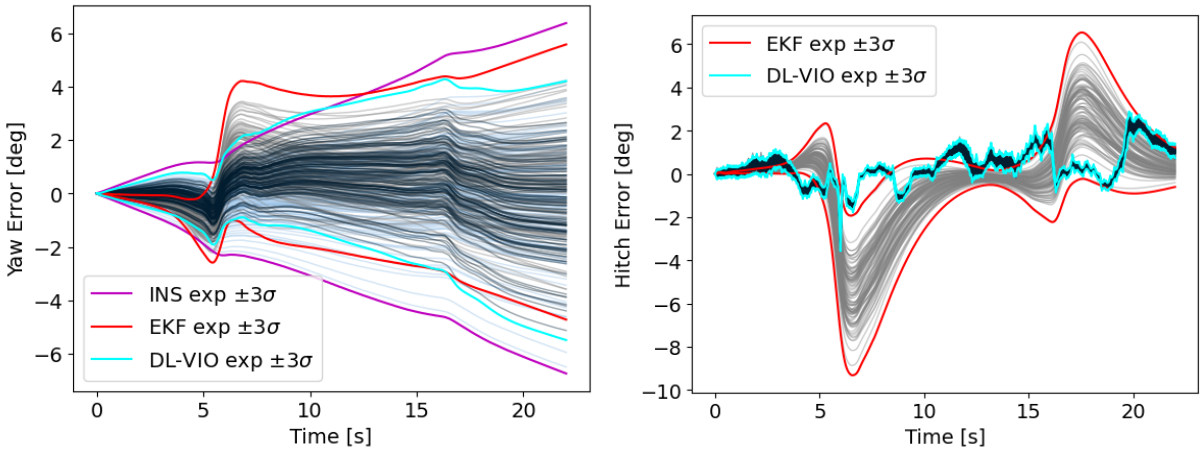
(a) North Error (b) East Error

Figure 6.11: Low dynamics sequence with low model uncertainty: North and East position errors



(a) North Error (b) East Error

Figure 6.12: Low dynamics sequence with low model uncertainty: EKF and DL-VIO North and East Position Errors



(a) Yaw Error

(b) Hitch Error

Figure 6.13: Low dynamics sequence with low model uncertainty: Yaw and Hitch Angle Errors

High Dynamic Scenario Results

Next, a high dynamic scenario is studied. For this scenario, the longest and most dynamic sequence from the simulation testing set is selected. The sequence is approximately one and a half minutes and includes high dynamic lane changes with an average driving speed of 12.5 m/s or 28 mph. The sequence trajectory is shown in Figure 6.14. The input steering angle, longitudinal velocity, and the resultant yaw and hitch angles for the sequence is shown in Figure 6.15.

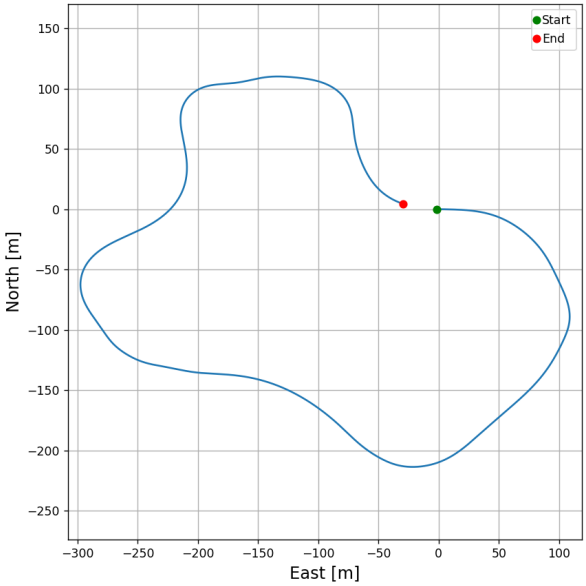


Figure 6.14: High dynamic sequence North and East Trajectory

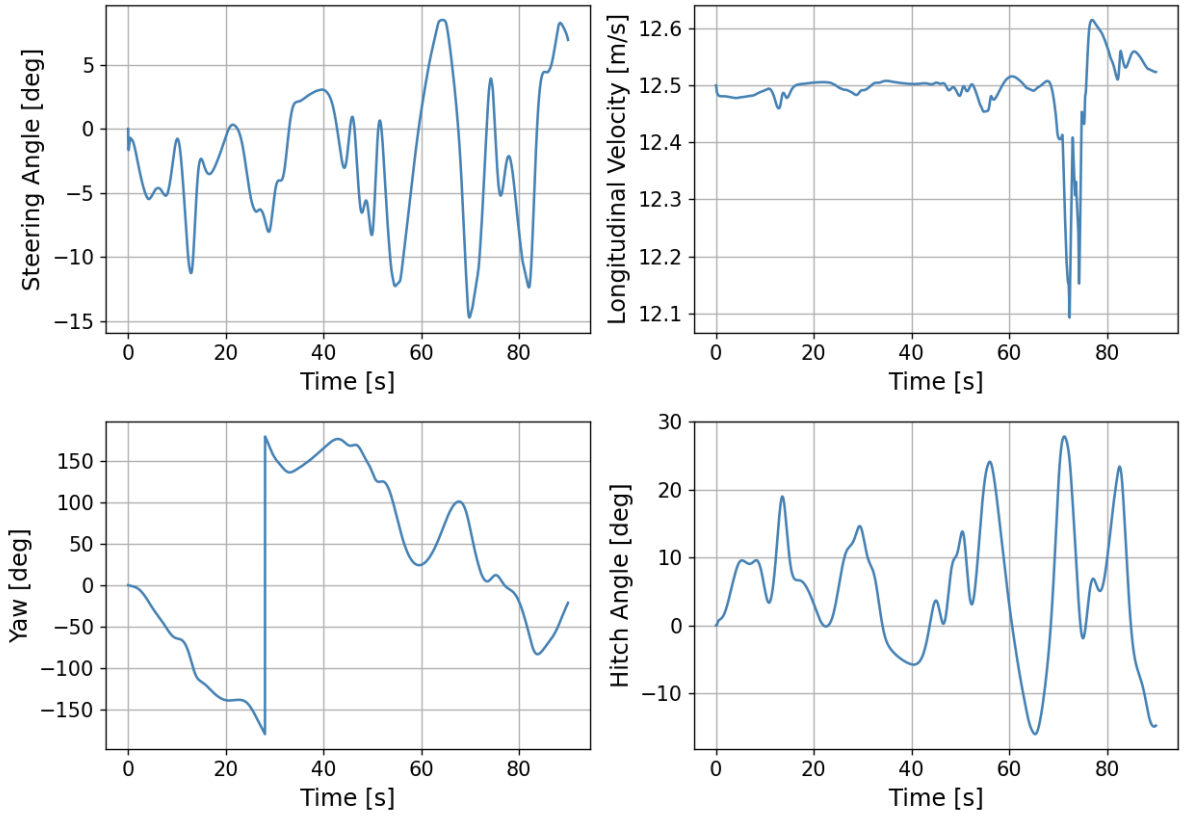
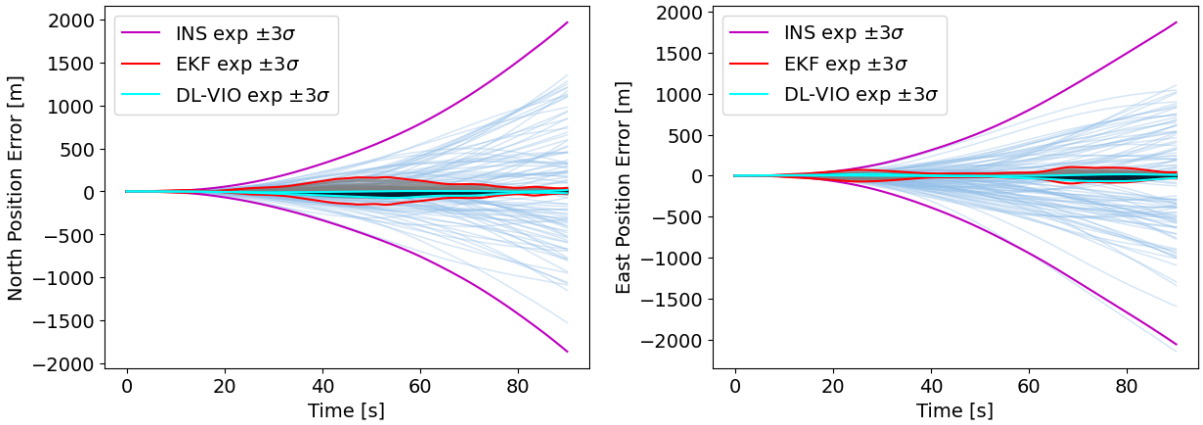


Figure 6.15: High dynamic sequence steering angle, longitudinal velocity, yaw angle, and hitch angle

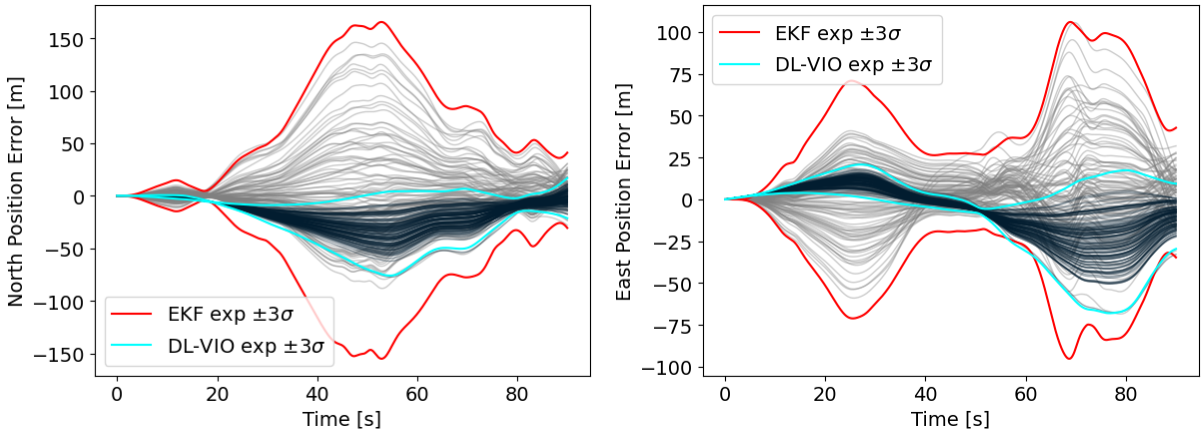
The case studies evaluated for the high dynamic scenario follow those conducted for the low dynamic scenario. First, the results using high model uncertainty are assessed. Figure 6.16 shows the North and East position errors for the three systems, Figure 6.17 shows positions errors from the EKF and DL-VIO only, and Figure 6.18 shows the yaw and hitch angle errors. It is shown that similar findings from the low dynamic scenario hold for the high dynamic scenario. The DL-VIO remains the superior dead reckoning navigation system compared to the other two systems evaluated. The errors induced by high model uncertainty for the EKF are amplified in the high dynamics case. This is especially evident in Figure 6.18b where it can be seen that the magnitude of maximum hitch angle errors directly correspond to the dynamics. Additionally, it is shown in Figure 6.17 that the maximum North and East position errors of the EKF exceeds the maximum position errors of the DL-VIO by approximately 100 meters. Furthermore, it is shown that the INS solution drifts significantly over time, reaching up to over 1000 meters of position error between the North and East positions. This is a common result for INS with low-cost sensors. It is important to highlight that the EKF is still able to maintain much better positioning than an INS even given poor model knowledge.



(a) North Error

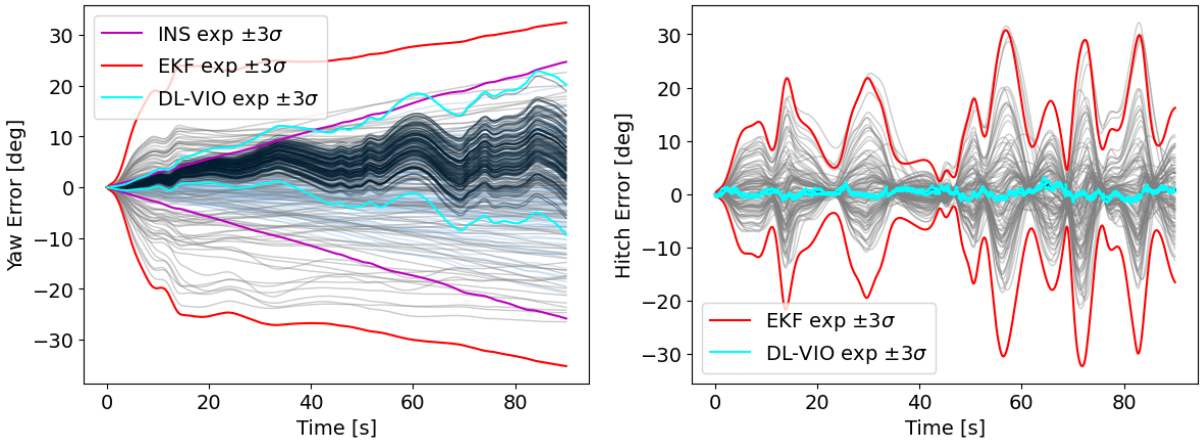
(b) East Error

Figure 6.16: High dynamics sequence with high model uncertainty: North and East Position Errors.



(a) North Error (b) East Error

Figure 6.17: High dynamics sequence with high model uncertainty: EKF and DL-VIO North and East Position Errors.

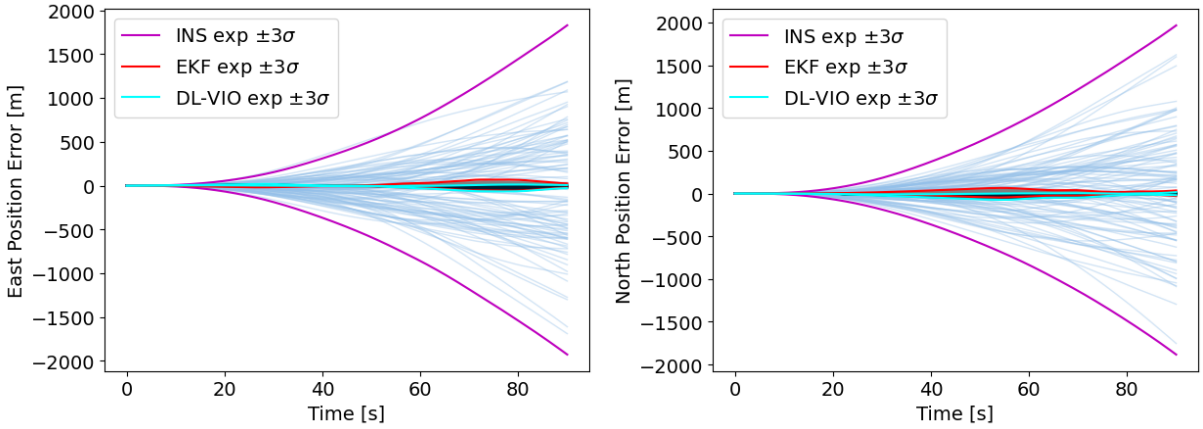


(a) Yaw Error (b) Hitch Error

Figure 6.18: High dynamics sequence with high model uncertainty: Yaw and Hitch Angle Errors.

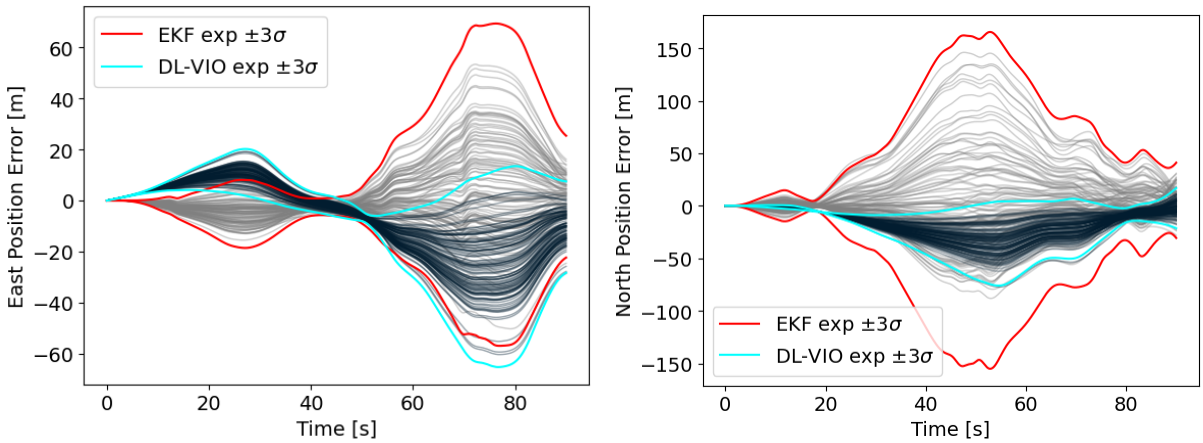
Lastly, the final case study evaluates the performance of the systems in the high dynamic scenario with low model uncertainty. Again, the only improvements expected are for the EKF. Figure 6.19 shows the North and East position errors for the three systems, Figure 6.20 shows positions errors from the EKF and DL-VIO only, and Figure 6.21 shows the yaw and hitch angle errors. The results indicate a significant improvement for the vehicle model EKF, showing a reduction in maximum position errors by approximately 100 meters compared to the previous case. The resultant yaw estimates from the EKF is also more closely aligned with the INS since the majority of

the errors for the system is due to the IMU's quality. Furthermore, the hitch angle estimates from the EKF is more accurate compared to the previous case, but remains less accurate than the hitch predictions from the DL-VIO. Moreover, the magnitude of the hitch angle errors correspond to the high dynamic maneuvers as observed in the low dynamic case. Overall, these results directly reflect the findings shown in the low dynamic scenario. The most significant discovery through the high dynamic evaluations is that the EKF errors for certain states, such as orientation and hitch angle, can be amplified in high dynamic maneuvers, especially if the model is not known accurately.



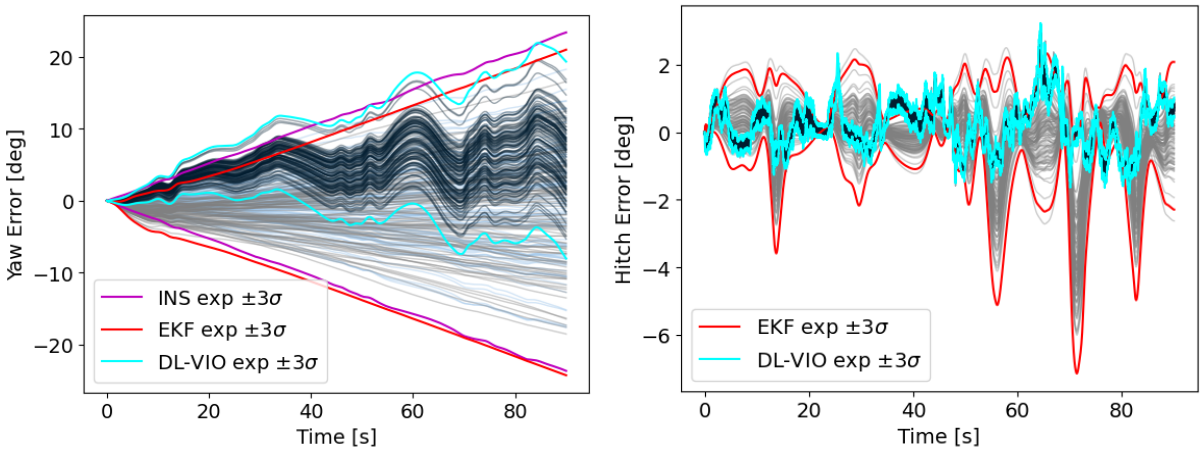
(a) North Error (b) East Error

Figure 6.19: High dynamics sequence with low model uncertainty: North and East Position Errors.



(a) North Error (b) East Error

Figure 6.20: High dynamics sequence with low model uncertainty: EKF and DL-VIO North and East Position Errors.



(a) Yaw Error

(b) Hitch Error

Figure 6.21: High dynamics sequence with low model uncertainty: Yaw and Hitch Angle Errors.

Ultimately, the simulation results show that the proposed DL-VIO consistently outperforms the model-based EKF with significant performance gains. Both systems are still able to maintain superior positioning when compared to the INS, but in the presence of model uncertainty, the solutions from the model-based EKF begin to degrade. Furthermore, the EKF errors are amplified at higher dynamics, especially if parameters are not known accurately. The DL-VIO shows the best performance of the three systems while being completely agnostic to perturbed vehicle parameters or higher dynamics.

6.2 Experimental Results

6.2.1 Experimental Setup

To validate the feasibility of the proposed tractor-trailer DL-VIO network, experimental evaluations are also performed. Experimental data is collected on a GAVLab class 8 test vehicle which is shown in Figure 6.22. Additional sensors equipped to the vehicle are a VN-100 IMU, two Triton 3.2 MP RGB monocular cameras, and two Honeywell eTALINs for the tractor and trailer units. The VN-100 IMU is mounted inside the tractor unit, and reports specific forces and angular rates at 40 Hz. The Triton

monocular cameras are mounted to the side mirrors facing backwards and are synchronized to record images at 10 Hz. The eTALINs are mounted in the tractor and trailer to provide ground truth measurements from both units. The eTALIN is a high performing navigation sensor which provides full PVA solutions at 150 Hz. Moreover, the vehicle's CAN bus reports the vehicle longitudinal velocity from wheel speeds, and the hand wheel angle, which is converted to the tire steer angle by a steering ratio. All sensors are recorded on a OnLogic Ubuntu PC using the Robot Operating System (ROS). Figure 6.23 shows the experimental setup.

Figure 6.22: GAVLab Class 8 Test Vehicle

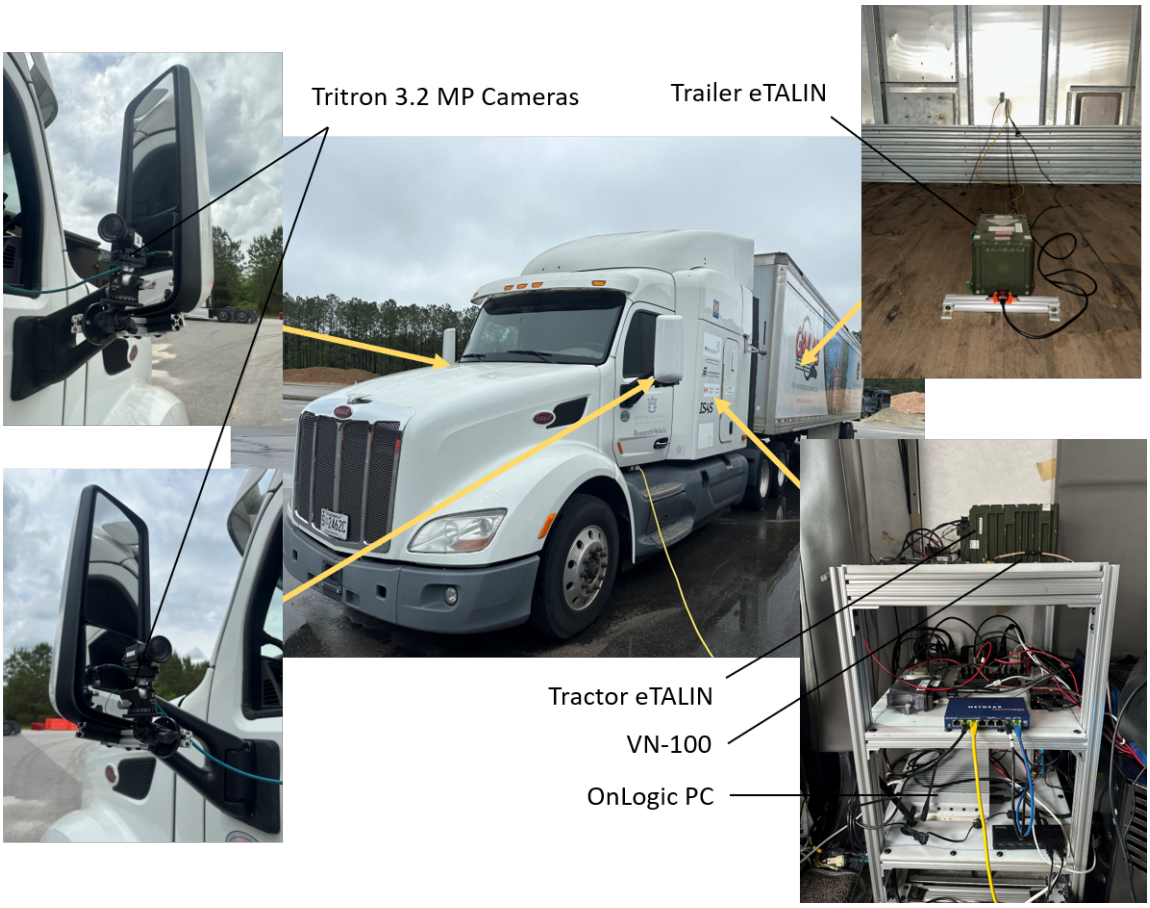


Figure 6.23: Test Vehicle Equipment Setup

The experimental data was collected throughout Auburn and Opelika, AL. All data was collected within the same day starting at the National Center for Asphalt Technology (NCAT) facility near Opelika, AL. Figure 6.24 shows the geographic plot of the collected data. In the figure, the vehicle travels in a clockwise direction over the entire loop trajectory. Six individual sequences are recorded throughout the course of the collection spanning approximately 20 minutes per sequence. The purpose of collecting individual sequences is to generate training and testing data from sequences that are completely unique. Furthermore, the data collected spans a variety of driving environments including rural roads, urban roads, and highways. All data is collected during daytime and no nighttime data was collected. Examples of camera data collected is shown in Figure 6.25. It is shown in the figure that the image data captures a variety of urban and rural environments and small lighting variations.

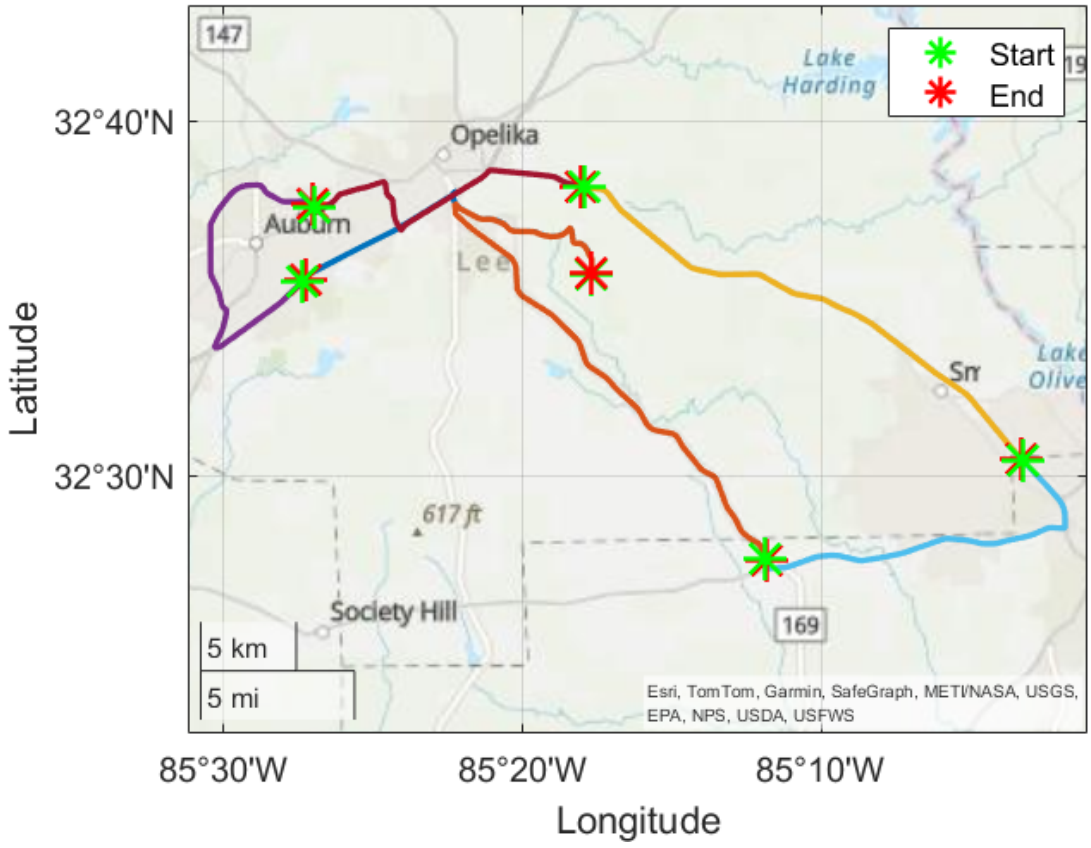


Figure 6.24: The six experimental sequences across Auburn and Opelika, AL.

Figure 6.25: Examples of the collected camera data across each sequence.

6.2.2 Experimental DL-VIO Parameters and Training Results

For the results presented in this chapter, the DL-VIO is trained on sequences one, three, and five, and tested on sequences two, four, and six. Note that the testing and training sequences are swapped for comparison and the results are shown in Appendix B. The results produced by swapping the datasets are slightly worse than those presented in this chapter, but the main contributions of the DL-VIO are mostly similar. For more details on the swapped dataset results, see Appendix B.

Figure 6.26 shows the training and testing sequences. A small validation set is randomly sampled from the testing set. The training and testing sets are organized this way so that each dataset contains diverse camera data and a wide range of dynamics. The network and training parameters for the experimental data are similar to those of the simulation network, but increases the temporal training window, and learning rate. Through manual tuning, these adjustments were found to improve the performance for the experimental data. Table 6.4 details these parameters. The training and validation RMSEs of the DL-VIO's outputs are shown in Figure 6.27. The training results show similar patterns to the simulation training results. The training and validation RMSEs are shown to be minimized throughout training at similar magnitudes and rates. Of note, the hitch angle training and validation RMSEs in Figure 6.27 are seen to slightly diverge throughout training. However, the actual performance on the testing set is shown to be reasonable, and this divergence is most likely due to a distribution mismatch in the training and validation sets due to their distinct sample sizes.

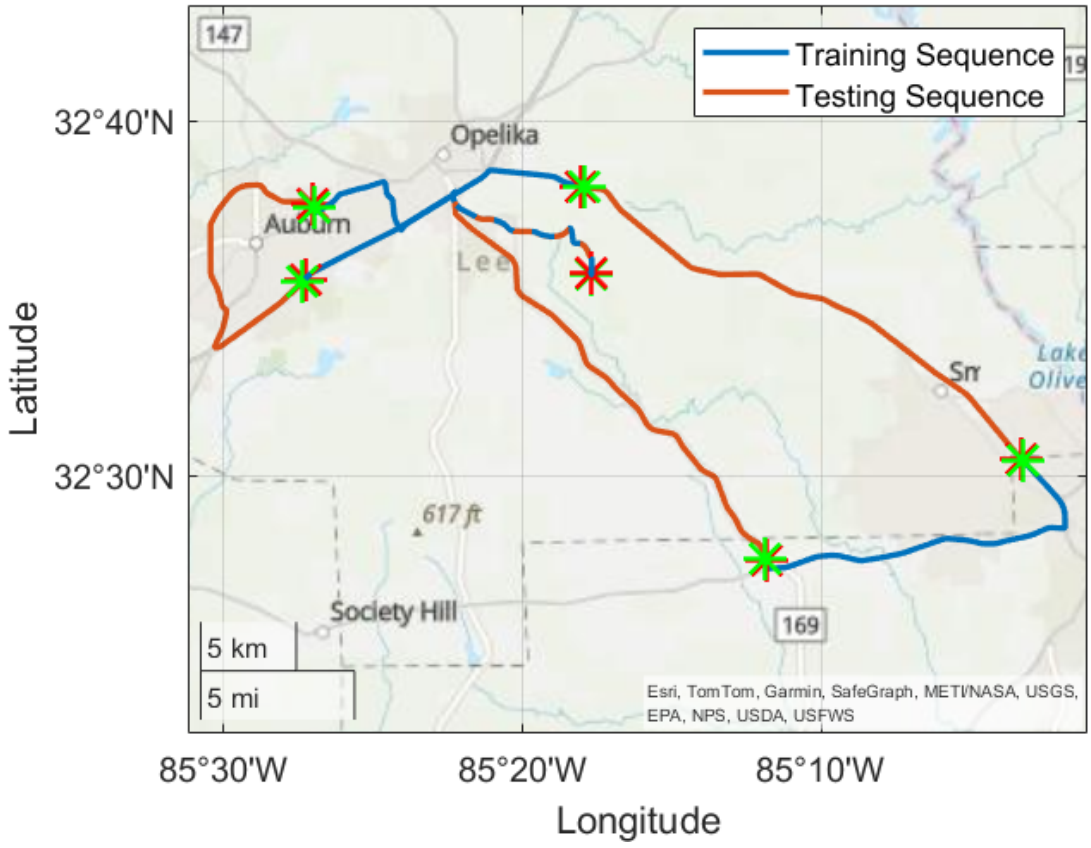


Figure 6.26: The experimental training and testing sequences for the DL-VIO network.

Table 6.4: Experimental DL-VIO Model and Training Parameters

Model Parameters	
Image Size	224×224
Patch Size	16
Embedding Dimensions	384
Heads	8
Depth	8
Training Parameters	
Epochs	30
Temporal Training Window	8
Batch Size	4
Learning Rate	1×10^{-4}
Loss Function	MSE
Optimizer	AdamW ($\beta_1 = 0.9,\ \ \beta_2 = 0.999,\ \ \gamma = 0.05$) [58]
Warmup	Linear (2 Epochs)
Scheduler	Cosine Decay with Warm Restarts ($2\times$ Decay after restarts)

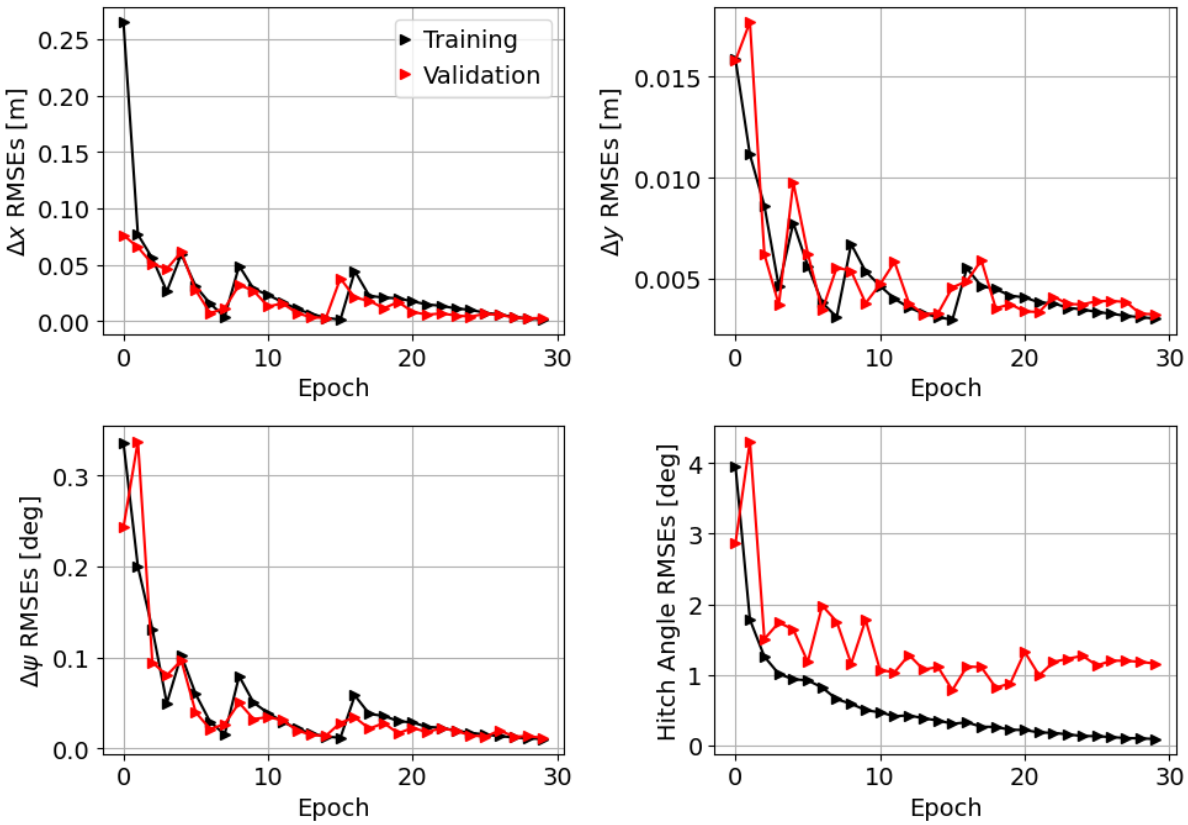


Figure 6.27: The training and validation RMSEs of the DL-VIO network's outputs on experimental data.

6.2.3 Experimental Results on Testing Sequences

This section presents the experimental results. For brevity, only the results for the the sequences in the testing set are shown in this section, and the remaining results for sequences in the training set are shown in Appendix A. The INS solution is also omitted to focus on the model-based EKF and the DL-VIO solutions, but the INS drift times are included for each sequence for comparison. Furthermore, a single GNSS correction update providing North position, East position, and yaw corrections is applied for each system approximately halfway into each sequence, and are indicated by a cyan dot in the presented figures. Given an accurate initialization and the GNSS correction, two distinct dead reckoning phases are created, each at approximately 10 minute intervals. The intent of the added correction is to better observe the solution drift characteristics of the dead reckoning systems over two distinct instances. The vehicle parameters used for the model-based EKF are the same as the nominal values defined in Table

6.2. These parameters are considered the nominal parameters of the experimental test vehicle and the results shown for EKF represents the system's performance given low model uncertainty.

Sequence 2 Results

Sequence 2 takes place mostly in Auburn, AL where the test vehicle exits US Interstate 85 (I-85) and enters through south Auburn. The trajectory then travels through the city towards north Auburn. The position, heading, and hitch angle estimates and errors for this sequence are shown in Figure 6.28, Figure 6.29, and Figure 6.30, respectively. The results show a benefit of the DL-VIO over the EKF for all states. For this sequence, the DL-VIO errors remain lower than the model-based EKF with nominal parameters across several metrics. Figure 6.31 shows the error statistics of each system for this sequence. The data suggests the DL-VIO network produces consistently lower RMSE, error STD, and max errors than the EKF. Of note, the maximum error of the hitch angle is higher for the DL-VIO. This is due to a specific spike for a single instance where the network is unable to make a reliable hitch angle prediction. The behavior is observed across all sequences for both hitch angle and yaw angle predictions. This is a common symptom of a pure measurement system such as the DL-VIO, where outliers may be more common than a smoothed solution from a filter.

In addition, the times it takes for each system to reach 1 meter, 5 meters, and 10 meters of absolute position error from the initial starting point are evaluated. Viewing these times reveals the system's performance from a much tighter scope. In reality, a solution with 1000 meters of error and one with 100 meters of error are both considered unusable for precise control and guidance applications. This metric observes how long each system is able to maintain a certain positioning accuracy from the initial point that a position fix becomes unavailable. Figure 6.32 shows the drift times for Sequence 2, and it is shown that the DL-VIO is able to maintain accurate positioning for much longer periods than the EKF. Improvements are on the scale of tens-of-seconds which may

be crucial for high precision control and guidance systems to leverage. This highlights a significant benefit of the DL-VIO over the EKF.

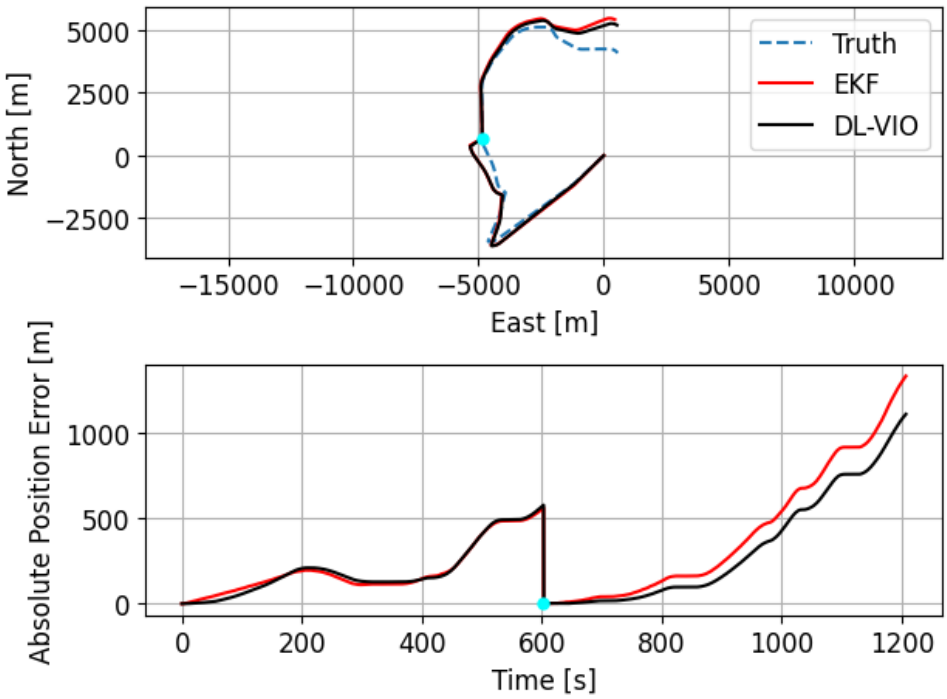


Figure 6.28: The position and position errors of the model-based EKF and DL-VIO network over Sequence 2.

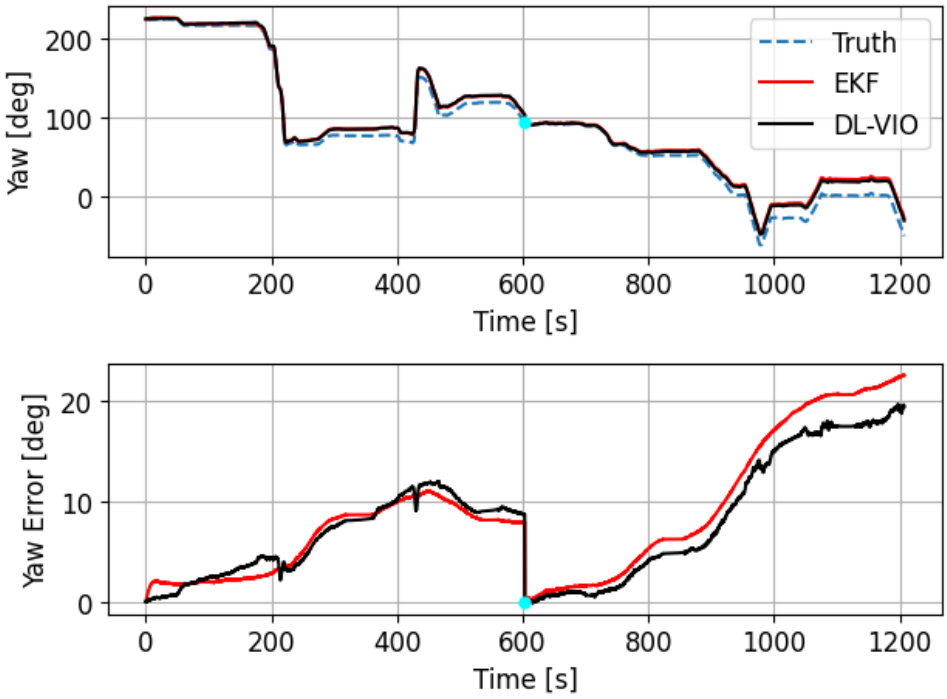


Figure 6.29: The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 2.

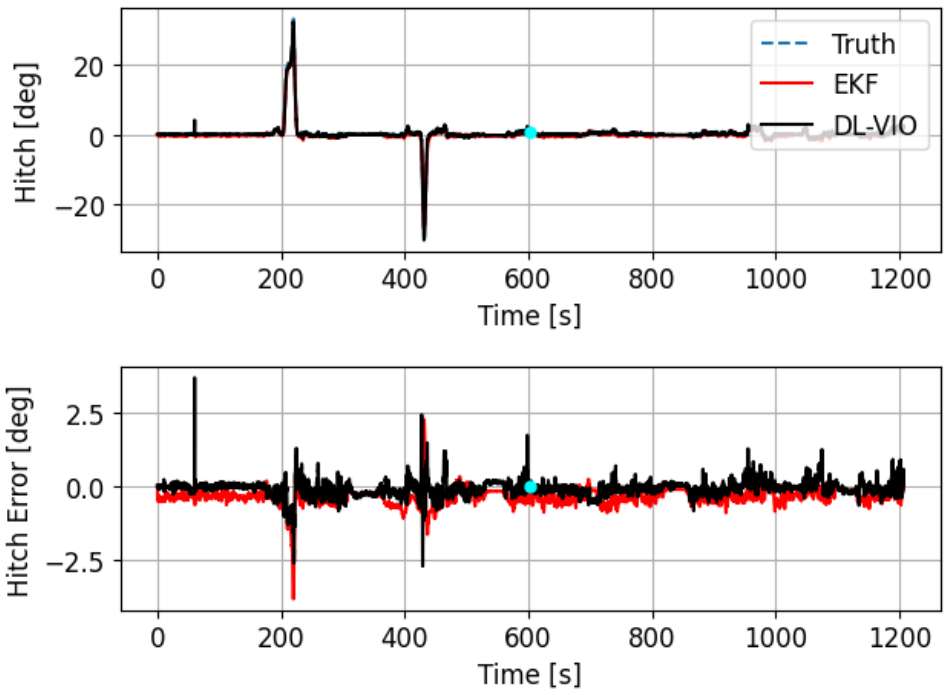


Figure 6.30: The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 2.

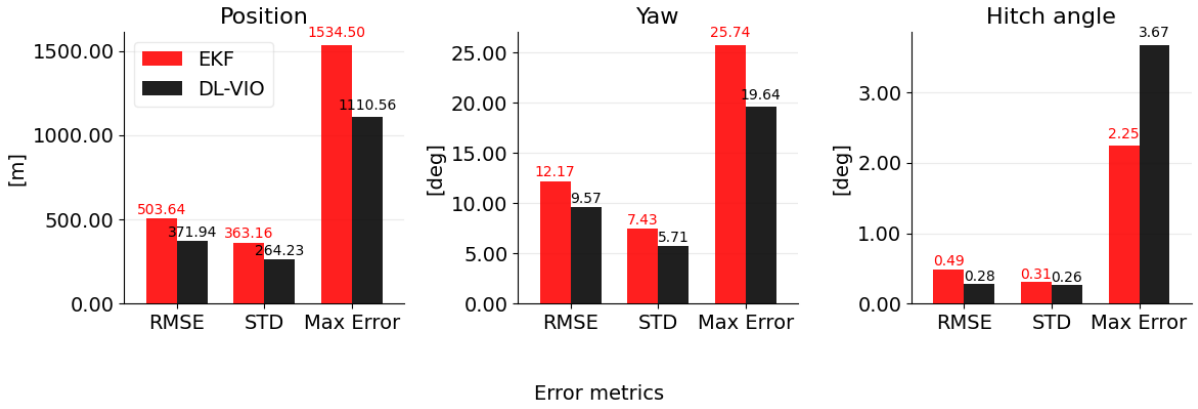


Figure 6.31: Error Statistics for Sequence 2

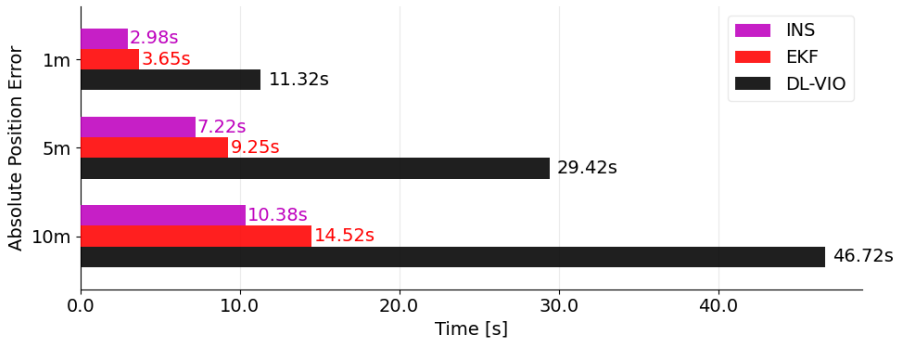


Figure 6.32: Drift Times for Sequence 2

Sequence 4 Results

Sequence 4 takes place entirely on US Highway 280 and contains no stops, unlike the other sequences. The test vehicle travels southeast-bound towards Columbus, GA, with relatively low dynamics. The position, heading, and hitch angle estimates and errors for this sequence are shown in Figure 6.33, Figure 6.34, and Figure 6.35, respectively. In this sequence, the DL-VIO greatly outperforms the model EKF, especially in the first dead reckoning phase. In the first phase, a small initial heading error of approximately two degrees causes the EKF position error to grow linearly, while the DL-VIO is able to maintain significantly lower position errors. It is noted that the DL-VIO solution does begin to drift quicker than the EKF from the 400 second to 600 second time interval. After the GNSS correction, it is shown that the EKF slightly outperforms the DL-VIO in both magnitude and drift rate, however, both system errors remain below 200 meters in the second dead reckoning phase. Figure 6.36 reinforces the benefits of the DL-VIO showing consistent and significant reductions in RMSE, error STD, and max errors compared to the EKF system. In addition, Figure 6.37 shows the position drift times for Sequence 4. Again, improvements over the EKF are shown on the scale of tens-of-seconds. For this sequence, the DL-VIO is able to maintain a position below 1 meter of absolute error for approximately 10 seconds, and remain below 10 meters of absolute error for over one minute.

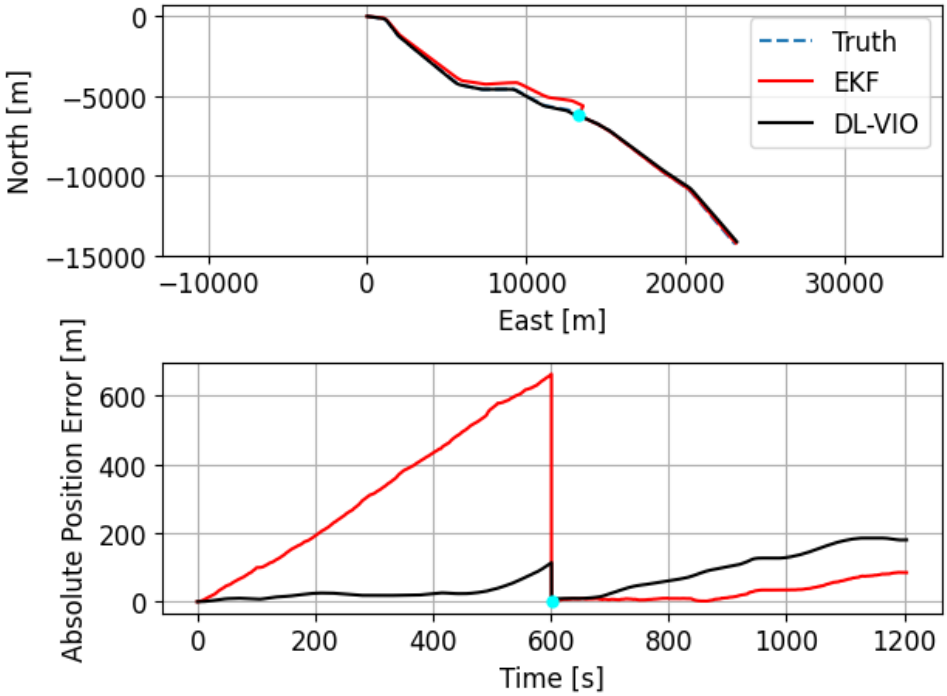


Figure 6.33: The position and position errors of the model-based EKF and DL-VIO network over Sequence 4.

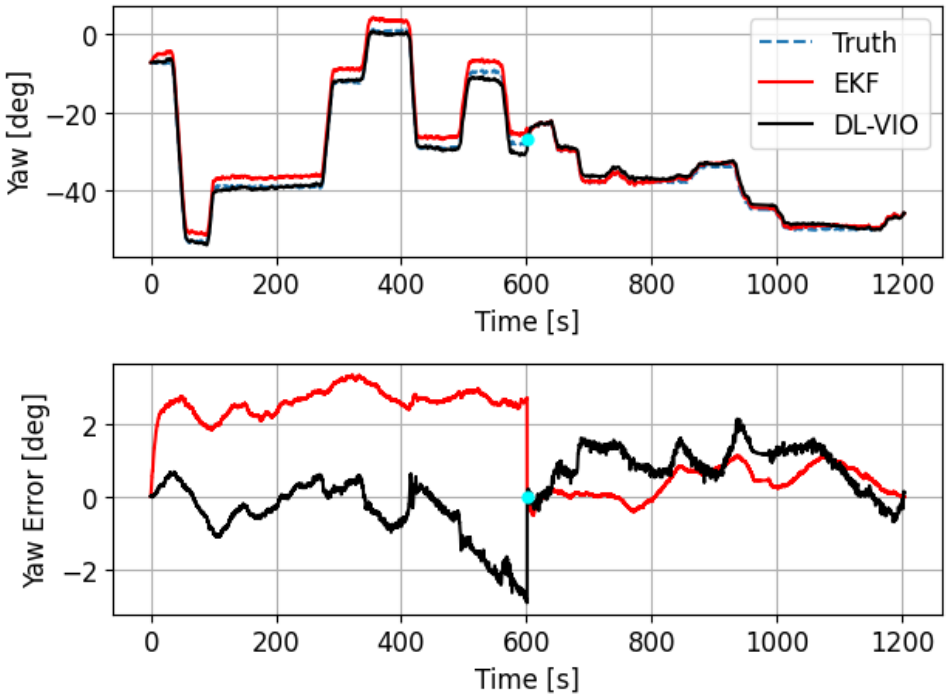


Figure 6.34: The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 4.

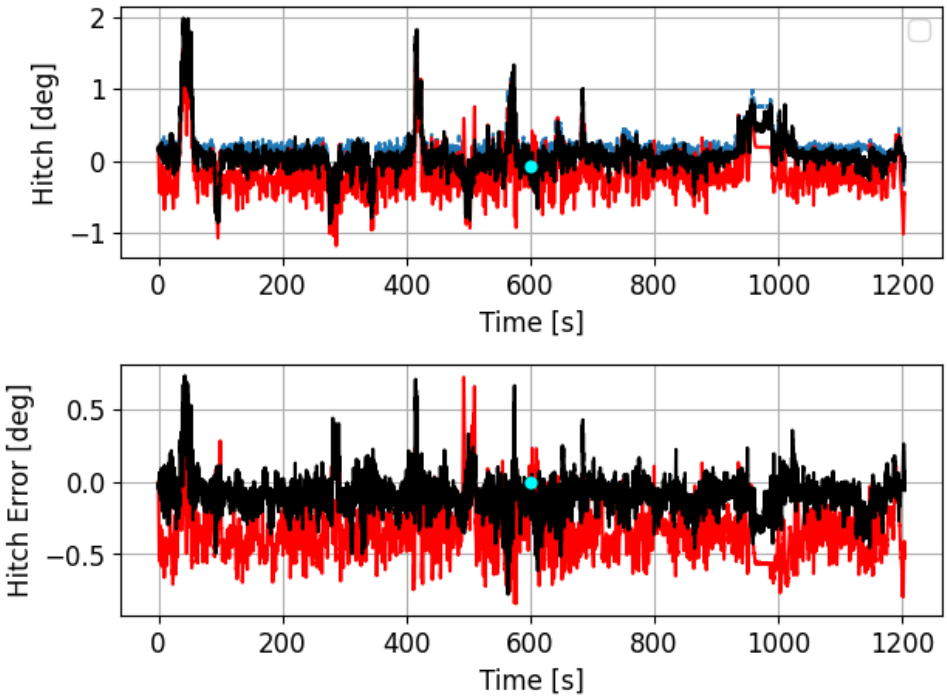


Figure 6.35: The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 4.

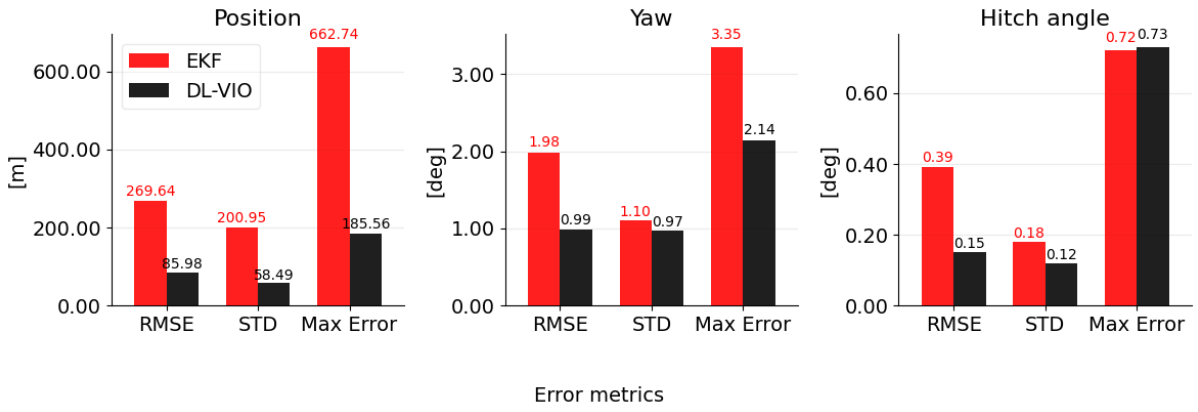


Figure 6.36: Error Statistics for Sequence 4

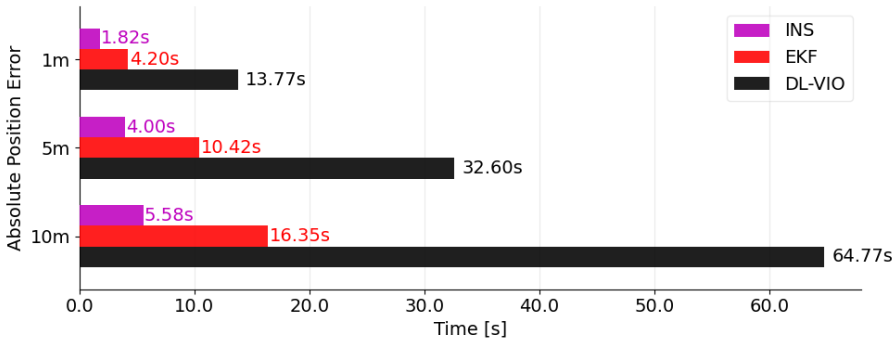


Figure 6.37: Drift Times for Sequence 4

Sequence 6 Results

Sequence 6 is the final sequence of the collection and mostly follows Alabama Highway 169 (AL-169) traveling back towards the NCAT facility. This sequence contains mostly rural features, and includes several sharp cornering turns as the test vehicle returns to the facility. The position, heading, and hitch angle estimates and errors for this sequence are shown in Figure 6.38, Figure 6.39, and Figure 6.40, respectively. Again, the DL-VIO produces superior positioning solutions compared to the EKF. However, for this sequence, larger errors are observed in the yaw estimate, especially during sharp turning scenarios. As shown in Figure 6.39, there are several instances where the DL-VIO yaw errors are large, especially towards the end of the sequence. These errors directly correspond to sharp turning scenarios such as 90-degree slow turns, and are most likely caused by the network overestimating or underestimating yawing motion under these scenarios. Since the yaw solution from the DL-VIO is constructed directly from changes in rotations, high errors in rotation predictions are added directly in the resulting yaw estimate. This was observed for this particular sequence, where the direct $\Delta\psi$ predictions from the network had approximately 3 to 5 degrees of error during these maneuvers. Irrespectively, the error statistics shown in Figure 6.41 suggest that overall the DL-VIO remains superior in positioning compared to the DL-VIO. However, the appropriate system under these circumstances may be dependent on a priority to reduce position drift or to maintain reasonable state estimates. Lastly, Figure 6.42 shows the drift times for Sequence 6. For this sequence, the DL-VIO is again shown to maintain accurate position for substantially longer periods than the EKF. Specifically, the DL-VIO is able to remain below 1 meter of absolute position error for over 40 seconds and below 10 meters of absolute error for over two minutes.

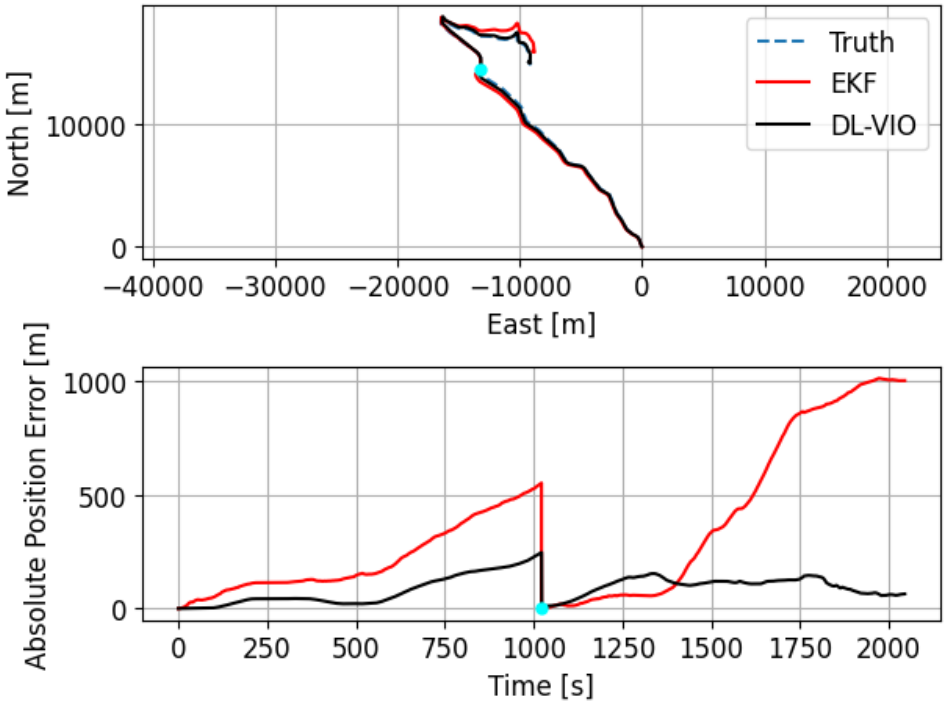


Figure 6.38: The position and position errors of the model-based EKF and DL-VIO network over Sequence 6.

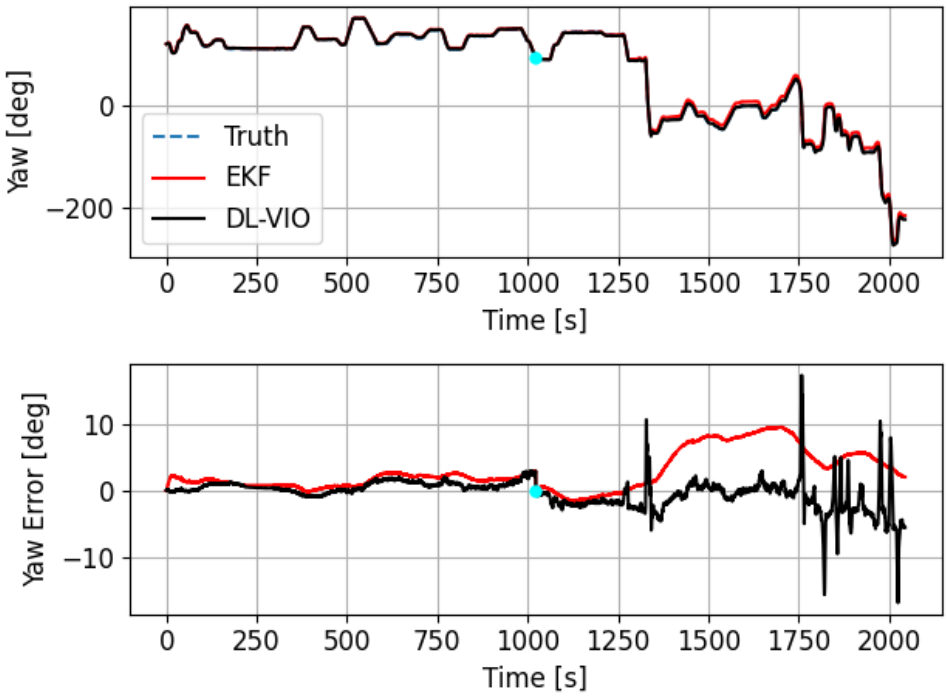


Figure 6.39: The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 6.

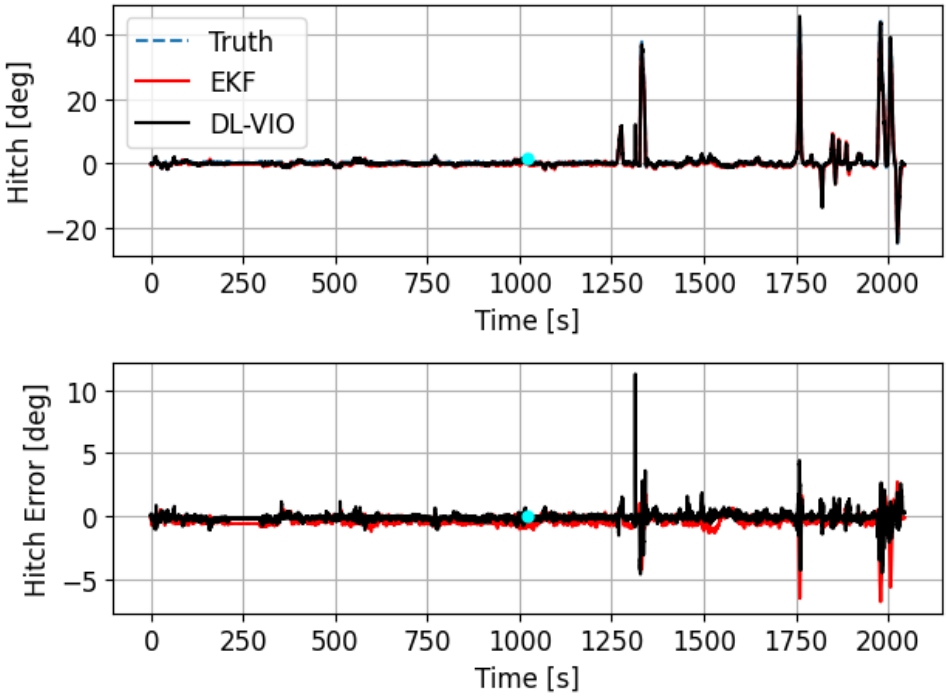


Figure 6.40: The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 6.

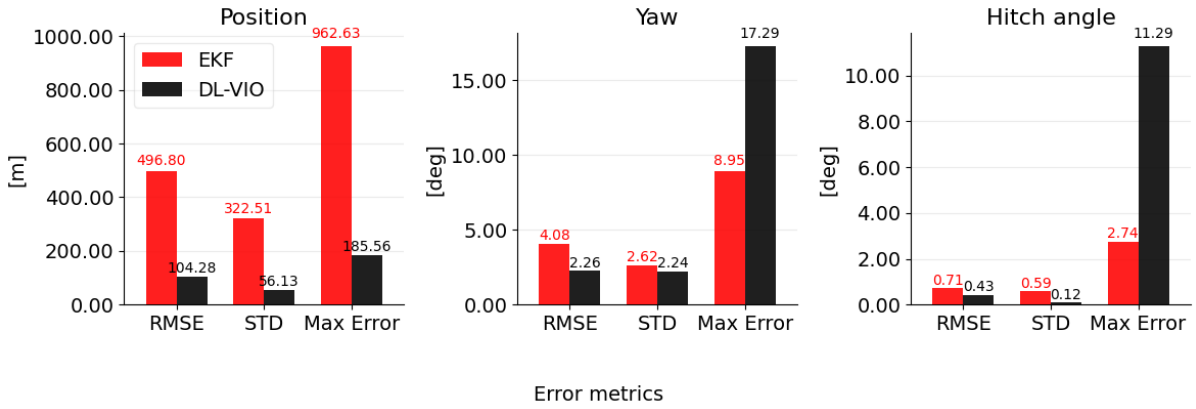


Figure 6.41: Error Statistics for Sequence 6

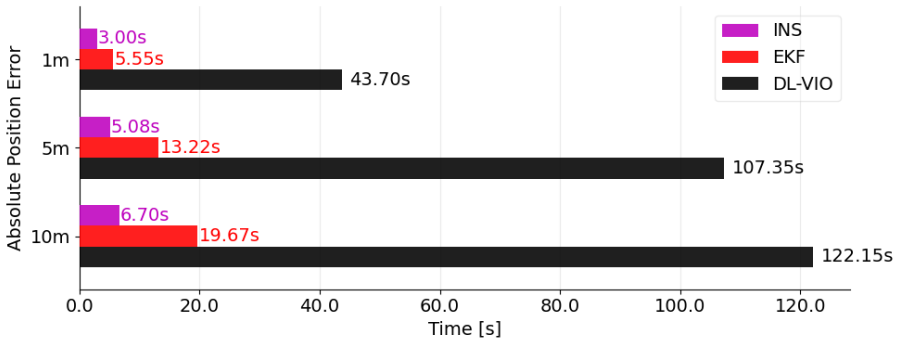


Figure 6.42: Drift Times for Sequence 6

Chapter 7

Conclusions and Future Works

7.1 Conclusions

In this thesis, a vehicle model based dead reckoning system, and a deep learning based dead reckoning system were designed and evaluated for GNSS independent navigation and state estimation of class 8 vehicles. A derivation for a tractor-trailer model was detailed and a navigation EKF that uses the model's state estimates was designed. Then, a deep learning alternative was proposed to address the shortcomings of the model-based method. A brief background on modern deep learning techniques was discussed, and these concepts were extended towards the design and evaluation of a deep learning model that ingests only sensor data to predict changes in vehicle odometry and hitch angle estimates. The proposed deep learning model is similar to traditional VIO, but it is able to ingests image, IMU, and CAN bus data to produce predictions without requiring direct information of the vehicle model, or sensor models (e.g., camera intrinsic and extrinsic parameters, IMU error model, camera to IMU orientation, etc.).

The navigation and state estimation performance of the model based method and deep learning method were evaluated through various Monte Carlo simulations. The Monte Carlo simulations reinforced the limitations of the model based method, showing large errors in both navigation and hitch angle states in the presence of model uncertainty. Furthermore, the simulations highlighted the effectiveness of the deep learning approach, and showed that the network was able to consistently outperform traditional INS and the model-based EKF given the same IMU errors.

Finally, the model-based EKF and DL-VIO network were evaluated and compared on experimental data. Data was collected throughout Auburn and Opelika, AL and split into sequences for training and testing purposes. Through experiments, the DL-VIO was shown to be a superior dead reckoning system to the model-based EKF, even when model parameters are well known. The neural network was able to often predict the position, orientation, and hitch angle with significantly lower errors than the EKF. However, it was found that under certain maneuvers such as sharp turns, the DL-VIO may struggle to produce accurate estimates and the EKF solution then becomes more reliable. It was found that perhaps the most valuable benefit of the DL-VIO was that the system showed significantly slower drift times than the EKF and was able to maintain reasonably accurate positioning for much longer periods of time.

Overall, the proposed deep learning method proves to be a beneficial dead reckoning alternative through the simulations and experiments conducted in this thesis. The deep learning approach shows potential to be powerful by design as it does not require specific knowledge about the system or sensors, unlike traditional dead reckoning systems. For changes in sensor setup or quality, the network may always be extended by retraining on new data. Nevertheless, the proposed deep learning approach is not without faults. At its core, it is still a dead reckoning system that will degrade over time. Additionally, the tested system produces pure measurements which are prone to outliers and erroneous errors. This is especially true for deep learning systems where it may fail to generalize to unseen data. Some methods to overcome these challenges are discussed next as areas of potential future work.

7.2 Future Works

The first approach to smooth out the measurements from the neural network is to fuse them in a filter context. Although, the network model was designed to fuse the different modalities internally, the system is a black box by design and users have little control over the quality and accuracy of the network's predictions beyond the training data and different training approaches. With a classical filter such as the KF or EKF,

measurements can be weighted, and the filter can facilitate how much influence it has over the states depending on the measurement uncertainty. This allows additional control of the measurements which can be critical to eliminating the outliers and high errors from the pure DL-VIO measurements. However, given the current predictions from the DL-VIO, creating a measurement model that relates to the states of the EKF used in this thesis is not trivial. The hitch angle prediction can be mapped directly to the state, but the changes in odometry are functions of the current state and a previous state which requires special handling. Furthermore, the absolute North, East, and yaw measurements that are constructed from the DL-VIO's predictions may not be mapped directly to the states without violating the Kalman filter's assumptions. This is because the absolute measurements are computed by accumulating the motion predictions over time, thus its errors are correlated in time. A simple approach to use the outputs from the DL-VIO is to convert the changes in motion to velocity and rate measurements. However, the benefits of this may be limited as the DL-VIO runs at a lower rate than most IMUs. Further investigations to developing a correct measurement model that effectively relates the DL-VIO outputs to the states would be greatly valuable.

Another possible research avenue is to extend the deep learning network's capabilities. While the best attempts were made to train and evaluate the network on diverse data, an extension to the dataset would almost always improve robustness. For example, no nighttime data was collected or evaluated experimentally, and it is most likely that the present network would struggle to produce accurate measurements under these conditions. In addition, the inputs from each modality were used in whole and no quality corruption or modality dropout was investigated. This makes the current trained network sensitive to sensor corruption and outages.

Finally, an extension to the sensor suite may also be explored. This work uses a limited sensor suite to predict changes in odometry and hitch angle, but additional sensors may extend what the network is able to observe and predict. A core contribution of the neural network approach is that it is able to ingest sensor data directly without

any external information about the system, thus adding additional sensors would only require redesigning how the input data is ingested and fused. Overall, given that the direct measurements from DL-VIO presented in this thesis shows promising results, developing additional features are likely to lead to performance gains.

References

[1] “Maps and data - average annual vehicle miles traveled by major vehicle category,” Alternative Fuels Data Center. https://afdc.energy.gov/data/10309 (accessed Jan. 7, 2026).

[2] National Center for Statistics and Analysis. (2025, August). Traffic safety facts 2023: A compilation of motor vehicle traffic crash data (Report No. DOT HS 813 738). National Highway Traffic Safety Administration.

[3] Groves, P. D., “Principles of GNSS, Inertial, and Multi-sensor Integrated Navigation Systems,” Industrial Robot, vol. 39, no. 3, p. ir.2012.04939caa.011, Apr. 2012, doi: 10.1108/ir.2012.04939caa.011.

[4] Soloviev, A., “GNSS-INS Integration. In Position, Navigation, and Timing Technologies in the 21st Century: Integrated Satellite Navigation, Sensor Systems, and Civil Applications,” Jade Morton, Y.T., van Diggelen, F., Spilker, J.J., Parkinson, B.W., Jr., Lo, S., Gao, G., Eds.; IEEE: Piscataway, NJ, USA, 2020; Volume 2.

[5] Farrel, J.A., The Global Positioning System & Inertial Navigation; McGraw-Hill Professional: New York, NY, USA, 1998; pp. 241–288.

[6] Wang, M.; Yu, P.; Li, Y., Performance analysis of GNSS/INS loosely coupled integration systems under GNSS signal blocking environment. E3S Web Conf.

[7] Maklouf, O.; Adwaib, A., Performance Evaluation of GPS/INS Main Integration Approach. Int. J. Aerosp. Mech. Eng. 2014, 8, 476–484.

[8] Zhang, Q., Niu, X., Zhang, H., Shi, C., Algorithm improvement of the low-end GNSS/INS systems for land vehicles navigation. Mathematical Problems in Engineering, 2013, Article 435286. https://doi.org/10.1155/2013/435286

[9] Travis, William and Bevly, David M,. "Navigation Errors Introduced By Ground Vehicle Dynamics," Online Journal of Space Communication: Vol. 5 : Iss. 9 , Article 13.

[10] Gillespie, T., "Fundamentals of Vehicle Dynamics," Society of Automotive Engineers, Warrendale, PA, 1992, ISBN: 1-56091-199-9.

[11] Travis, W., "Methods For Minimizing Navigation Errors Induced By Ground Vehicle Dynamics," Auburn University, Auburn, Alabama, USA, Master's Thesis, 2006.

[12] Li T. et al. (2010). "Ultra-tight Coupled GPS/Vehicle Sensor Integration for Land Vehicle Navigation." NAVIGATION, Vol. 57, No. 4, pp. 248-259.

[13] F. Jindra, "Handling Characteristics of Tractor-Trailer Combinations," 2024.

[14] Z. Brock, J. Nelson, and R. L. Hatton, "A Comparison of Lateral Dynamic Models for Tractor-Trailer Systems," in 2019 IEEE Intelligent Vehicles Symposium (IV), Paris, France: IEEE, Jun. 2019, pp. 2052–2059. doi: 10.1109/IVS.2019.8814286.

[15] M. F. J. Luijten, "Lateral Dynamic Behavior of Articulated Commercial Vehicles".

[16] S. M. Wolfe, "Heavy Truck Modeling and Estimation for Vehicle-to-vehicle Collision Avoidance Systems," Ph.D. dissertation, Ohio State University, 2014.

[17] X. Meng, H. Wang, and B. Liu, "A Robust Vehicle Localization Approach Based on GNSS/IMU/DMI/LiDAR Sensor Fusion for Autonomous Vehicles," in Sensors, vol. 17, no. 9, pp. 2140-2159, Sep. 2017, doi: 10.3390/s17092140.

[18] L. Wei, C. Cappelle and Y. Ruichek, ”Camera/Laser/GPS Fusion Method for Vehicle Positioning Under Extended NIS-Based Sensor Validation,” in IEEE Transactions on Instrumentation and Measurement, vol. 62, no. 11, pp. 3110-3122, Nov. 2013, doi: 10.1109/TIM.2013.2265476.

[19] Rodríguez-Martínez, E. A., Flores-Fuentes, W., Achakir, F., Sergiyenko, O., Murrieta-Rico, F. N. (2025). Vision-Based Navigation and Perception for Autonomous Robots: Sensors, SLAM, Control Strategies, and Cross-Domain Applications—A Review. Eng, 6(7), 153. https://doi.org/10.3390/eng6070153

[20] Mur-Artal, R., Montiel, J.M.M., Tardos, J.D.: ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE Trans. Robot. 31(5), 1147–1163 (2015). https://doi.org/10.1109/TRO.2015.2463671

[21] Mur-Artal, R., Tardos, J.D.: ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-d cameras. IEEE Trans. Robot. 33(5), 1255–1262 (2017). https://doi.org/10.1109/TRO.2017.2705103

[22] Campos, C., Elvira, R., Gomez, J.J., Montiel, J.M.M., Tard ´ os, ´J.D.: ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM. arXiv preprint arXiv: 2007.11898. Accessed 2021-03-15 (2020)

[23] Antoni, R., Marcus, A., Yun, C., Luca, C.: Kimera: an open-source library for real-time metric-semantic localization and mapping. In: IEEE Intl. Conf. on Robotics and Automation (ICRA). https://github.com/MIT-SPARK/Kimera. Accessed 2021-03-15 (2020)

[24] Qin, T., Li, P., Shen, S.: Vins-mono: a robust and versatile monocular visual-inertial state estimator. IEEE Trans. Robot. 34(4), 1004–1020 (2018). https://doi.org/10.1109/TRO.2018.2853729

[25] Usenko, V., Demmel, N., Schubert, D., St¨uckler, J., Cremers, D.: Visual-inertial mapping with non-linear factor recovery. arXiv (2019). https://doi.org/10.1109/LRA.2019.2961227

[26] Geneva, P., Eckenhoff, K., Lee, W., Yang, Y., Huang, G.: Openvins: a Research Platform for Visual-Inertial Estimation. In: 2020 IEEE International Conference on Robotics and Automation (ICRA), pp. 4666–4672 (2020). https://doi.org/10.1109/ICRA40945.2020.9196524

[27] Yuan, Y., Chen, Z., Li, K., Wang, W., & Zhao, H. (2025). SLAM-Former: Putting SLAM into One Transformer. arXiv preprint arXiv:2509.16909.

[28] C. Chen et al., “Selective Sensor Fusion for Neural Visual-Inertial Odometry,” in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA: IEEE, Jun. 2019, pp. 10534–10543. doi: 10.1109/CVPR.2019.01079.

[29] X. Wang, H. Luo, Z. Wang, J. Zheng, and X. Bai, “Self-supervised multi-frame depth estimation with visual-inertial pose transformer and monocular guidance,” Information Fusion, vol. 108, p. 102363, Aug. 2024, doi: 10.1016/j.inffus.2024.102363.

[30] Kurt, Y. B., Akman, A., & Alatan, A. A. (2024). Causal transformer for fusion and pose estimation in deep visual inertial odometry. arXiv preprint arXiv:2409.08769.

[31] Z. Ziaukas, M. Wielitzka, T. Ortmaier and J. -P. Kobler, ”Simultaneous Estimation of Steering and Articulation Angle in a Truck-Semitrailer Combination Solely Based on Trailer Signals,” 2019 American Control Conference (ACC), Philadelphia, PA, USA, 2019, pp. 2509-2514, doi: 10.23919/ACC.2019.8814348.

[32] L. Xu, E. Tseng, T. Pilutti, and S. Schondorf, ”Yaw Rate Based Trailer Hitch Angle Estimation for Trailer Backup Assist,” SAE Technical Paper 2017-01-0027, 2017, https://doi.org/10.4271/2017-01-0027.

[33] L. Chu, Y. Fang, M. Shang, J. Guo and F. Zhou, ”Estimation of Articulation Angle for Tractor Semi-trailer Based on State Observer,” 2010 International Conference on Measuring Technology and Mechatronics Automation, Changsha, China, 2010, pp. 158-163, doi: 10.1109/ICMTMA.2010.342.

[34] V. Josef, ”Trailer backing-up assistant using ultrasound sensors based control units to safely back-up the car with trailer,” 2016 17th International Conference on Mechatronics - Mechatronika (ME), Prague, Czech Republic, 2016, pp. 1-6.

[35] A. H. Korayem A. Khajepour, B. Fidan, “Hitch angle estimation of a towing vehicle with arbitrary configuration,” IEEE Trans. Intell. Transp. Syst., early access, Apr. 16, 2021, doi: 10.1109/TITS.2021.3071391.

[36] K. Olutomilayo and D. R. Fuhrmann, ”Estimation of Trailer-Vehicle Articulation Angle Using 2D Point-Cloud Data,” 2019 IEEE Radar Conference (RadarConf), Boston, MA, USA, 2019, pp. 1-6, doi: 10.1109/RADAR.2019.8835505.

[37] M. Bahramgiri, S. Nooshabadi, K. T. Olutomilayo and D. R. Fuhrmann, ”Automotive Radar-Based Hitch Angle Tracking Technique for Trailer Backup Assistant Systems,” in IEEE Transactions on Intelligent Vehicles, vol. 8, no. 2, pp. 1922-1933, Feb. 2023, doi: 10.1109/TIV.2022.3144896.

[38] T. Flegel, “Relative Position Vector Generation with Computer Vision for Vehicle Platooning Applications,” Master’s thesis, Auburn University, 2023.

[39] S. S. B.V. and A. Karthikeyan, ”Computer Vision based Advanced Driver Assistance System Algorithms with Optimization Techniques-A Review,” 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2018, pp. 821-829, doi: 10.1109/ICECA.2018.8474604.

[40] C. de Saxe and D. Cebon, “Measurement of articulation angle by image template matching,” Proc. Inst. Mech. Eng. D, J. Automobile Eng., vol. 233, no. 14, pp. 3801–3815, Dec. 2019, doi:10.1177/0954407019833819.

[41] C. Fuchs, F. Neuhaus, and D. Paulus, “3D pose estimation for articulated vehicles using Kalman-filter based tracking,” Pattern Recognit. Image Anal., vol. 26, no. 1, pp. 109–113, Jan. 2016, doi: https://doi.org/10.1134/S1054661816010077.

[42] M. Bahramgiri, S. Nooshabadi, K. T. Olutomilayo, and D. R. Fuhrmann, “Hitch Angle Estimation for Trailer Backup System—An Object Detection and Tracking Approach,” IEEE Trans. Instrum. Meas., vol. 71, pp. 1–15, 2022, doi: 10.1109/TIM.2022.3212737.

[43] A. Dahal et al., “DeepTrailerAssist: Deep Learning Based Trailer Detection, Tracking and Articulation Angle Estimation on Automotive Rear-View Camera,” in 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), Seoul, Korea (South): IEEE, Oct. 2019, pp. 2339–2346. doi: 10.1109/ICCVW.2019.00287.

[44] D. Arnström, “State Estimation for Truck and Trailer Systems using Deep learning,” Master’s thesis, Linköping University, 2018.

[45] T. Thawainin, T. Flegel and D. Bevly, ”Surveying CNNs for Estimating Trailer Articulation Angle using Monocular Cameras,” 2025 IEEE/ION Position, Location and Navigation Symposium (PLANS), Salt Lake City, UT, USA, 2025, pp. 654-663, doi: 10.1109/PLANS61210.2025.11028402.

[46] Anon., Department of Defense World Geodetic System 1984, National Imagery and Mapping Agency, TR8350.2, 3rd ed., 1997.

[47] Farrell, J. A., and M. Barth, The Global Positioning System and Inertial Navigation, New York: McGraw-Hill, 1999.

[48] Kaplan, E. D., et al., “Fundamentals of Satellite Navigation” in Understanding GPS Principles and Applications, 2nd ed., E. D. Kaplan, and C. J. Hegarty, (eds.), Norwood, MA: Artech House, 2006, pp. 21–65.

[49] Hata, K., & Savarese, S. (n.d.). CS231A Course Notes 1: Camera models. Stanford University. https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf

[50] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. Journal of Basic Engineering, 82(1), 35–45. https://doi.org/10.1115/1.3662552

[51] McGee, L. A., & Schmidt, S. F. (1985). Discovery of the Kalman filter as a practical tool for aerospace and industry (NASA Technical Memorandum 86847). NASA Ames Research Center.

[52] A. Géron, “Reccurent Nueral Networks” in *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. CA, 2017, ch. 14. Accessed July 28, 2025. [Online] Available: https://www.clc.hcmus.edu.vn/wp-content/uploads/2017/11/Hands_On_Machine_Learning_with_Scikit_Learn_and_TensorFlow.pdf

[53] Baheti, P. (2021, May 27). Activation functions in neural networks [12 types & use cases]. V7 Labs. https://www.v7labs.com/blog/neural-networks-activation-functions

[54] Stanford University. (n.d.). Neural networks part 1: Setting up the architecture. CS231n: Deep Learning for Computer Vision. https://cs231n.github.io/neural-networks-1/

[55] Stanford University. (n.d.). Neural networks part 2: Setting up the data and the loss. CS231n: Deep Learning for Computer Vision. https://cs231n.github.io/neural-networks-2/

[56] Stanford University. (n.d.). Neural networks part 3: Learning and evaluation. CS231n: Deep Learning for Computer Vision. https://cs231n.github.io/neural-networks-3/

[57] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12, 2121–2159.

[58] Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. In International Conference on Learning Representations (ICLR 2019). https://arxiv.org/abs/1711.05101

[59] Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In 3rd International Conference on Learning Representations (ICLR 2015). https://arxiv.org/abs/1412.6980

[60] Brownlee, J. (n.d.). A gentle introduction to learning rate schedulers. Machine Learning Mastery. https://machinelearningmastery.com/a-gentle-introduction-to-learning-rate-schedulers/

[61] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning (pp. 448-456).

[62] Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv preprint arXiv:1607.06450. https://arxiv.org/abs/1607.06450

[63] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929–1958.

[64] I. J. Goodfellow et al., “Generative Adversarial Networks,” Jun. 10, 2014, arXiv: arXiv:1406.2661. doi: 10.48550/arXiv.1406.2661.

[65] Stanford University. (n.d.). Convolutional neural networks: Architectures, convolution / pooling layers. CS231n: Deep Learning for Computer Vision. https://cs231n.github.io/convolutional-networks/

[66] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 770–778). https://doi.org/10.1109/CVPR.2016.90

[67] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 2261-2269, doi: 10.1109/CVPR.2017.243.

[68] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. -C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 4510-4520, doi: 10.1109/CVPR.2018.00474.

[69] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008). [https://arxiv.org/abs/1706.03762]

[70] Brown, T. B., et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901. https://arxiv.org/abs/2005.14165

[71] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (pp. 4171–4186). Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1423

[72] Gemini Team, (2024). Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. arXiv preprint arXiv:2403.05530. https://arxiv.org/abs/2403.05530

[73] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 10684–10695). IEEE.

[74] Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., & Liu, T. (2020). On layer normalization in the transformer architecture. In Proceedings of the 37th International Conference on Machine Learning (Vol. 119, pp. 10524–10533). PMLR. http://proceedings.mlr.press/v119/xiong20b.html

[75] A. Dosovitskiy et al., “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” Jun. 03, 2021, arXiv: arXiv:2010.11929. Accessed: Nov. 07, 2024. [Online]. Available: http://arxiv.org/abs/2010.11929

[76] P. Xu, X. Zhu, and D. A. Clifton, “Multimodal Learning With Transformers: A Survey,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 45, no. 10, pp. 12113–12132, Oct. 2023, doi: 10.1109/TPAMI.2023.3275156.

[77] Kenworth Truck Company. (2026). T880. https://www.kenworth.com/trucks/t880/

[78] Freightliner Trucks. (n.d.). The Fifth Generation Cascadia. https://www.freightliner.com/trucks/cascadia/

[79] C. Chen et al., “Selective Sensor Fusion for Neural Visual-Inertial Odometry,” in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA: IEEE, Jun. 2019, pp. 10534–10543. doi: 10.1109/CVPR.2019.01079.

[80] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition (pp. 248–255). IEEE. https://doi.org/10.1109/CVPR.2009.5206848

[81] G. Bertasius, H. Wang, and L. Torresani, “Is Space-Time Attention All You Need for Video Understanding?,” Jun. 09, 2021, arXiv: arXiv:2102.05095. doi: 10.48550/arXiv.2102.05095.

[82] Paszke, A., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems 32 (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[83] Mechanical Simulation Corporation. TruckSim. Mechanical Simulation Corporation, Ann Arbor, MI, 2011. Version 8.

[84] The MathWorks, Inc. (2023). RoadRunner (Version R2023b) [Computer software]. https://www.mathworks.com/products/roadrunner.html

[85] The MathWorks, Inc. (2023). Simulink (Version 10.8, R2023b) [Computer software]. https://www.mathworks.com/products/simulink.html

Appendix A

Experimental Results on Training Sequences

In this Appendix, the results from the training sequences (i.e., Sequences 1, 3, and 5) are shown. The results match closely with those of shown for the experimental testing sequences (i.e., Sequences 2, 4, and 6).

Sequence 1 Results

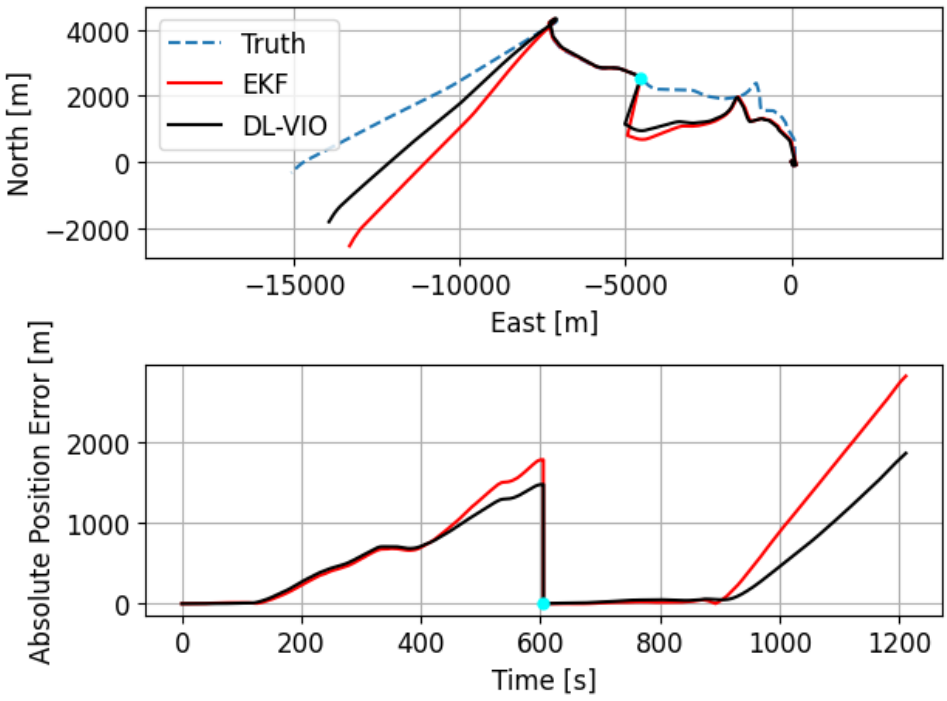


Figure A.1: The position and position errors of the model-based EKF and DL-VIO network over Sequence 1.

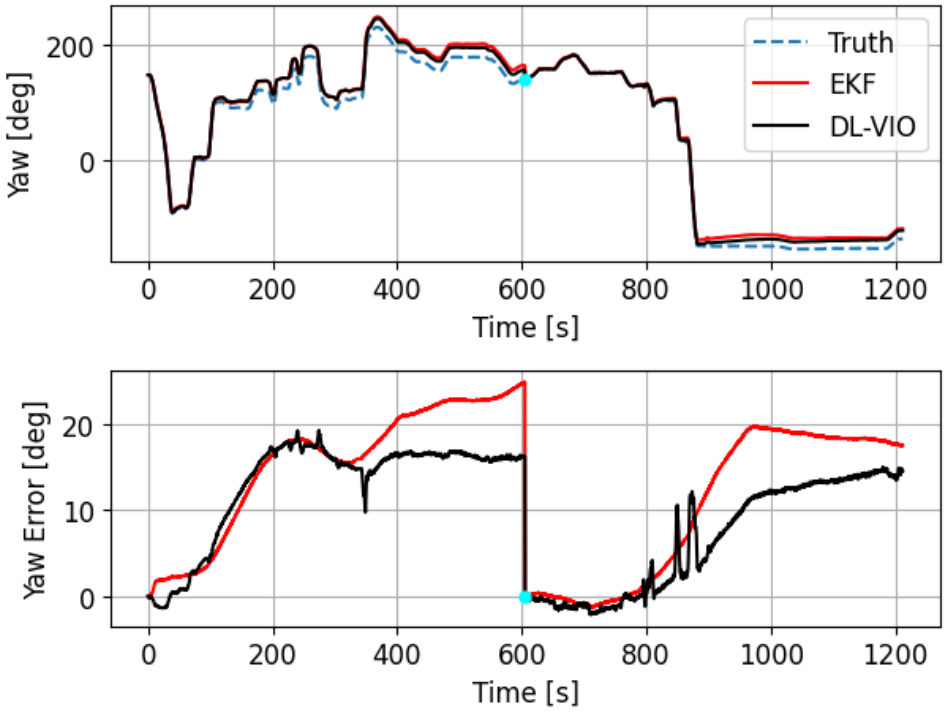


Figure A.2: The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 1.

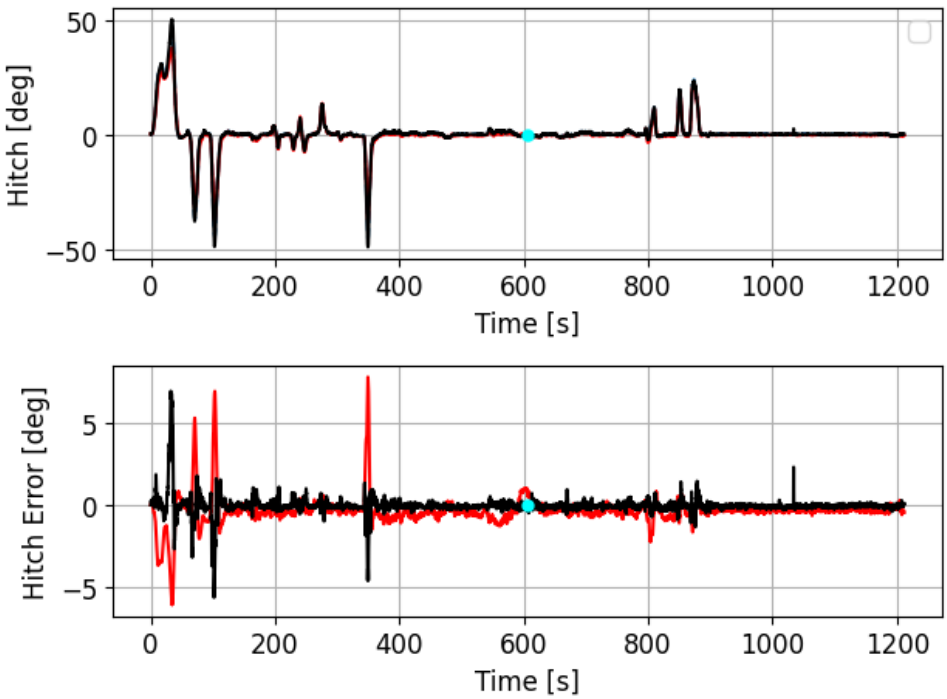


Figure A.3: The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 1.

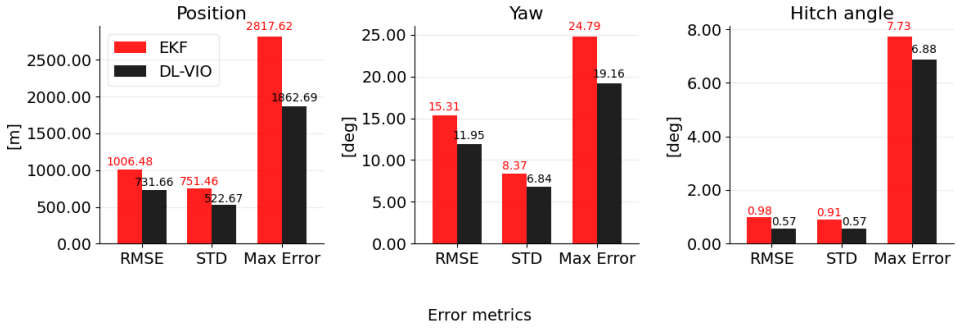


Figure A.4: Error Statistics for Sequence 1

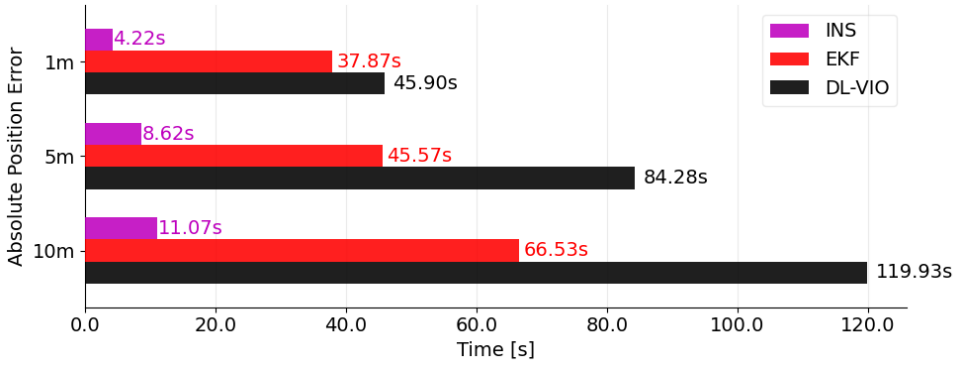


Figure A.5: Drift Times for Sequence 1

Sequence 3 Results

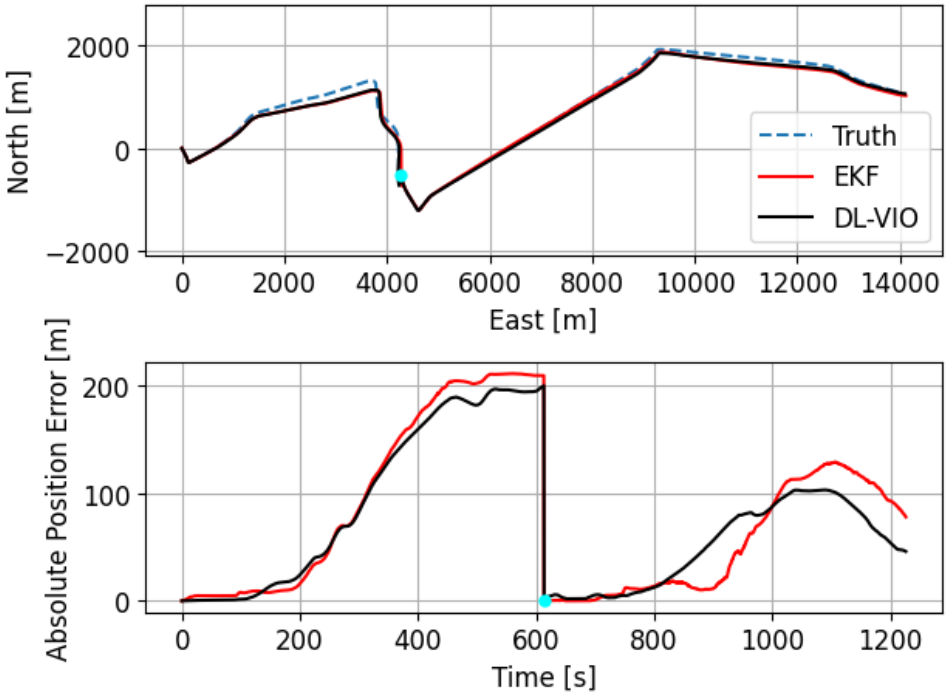


Figure A.6: The position and position errors of the model-based EKF and DL-VIO network over Sequence 3.

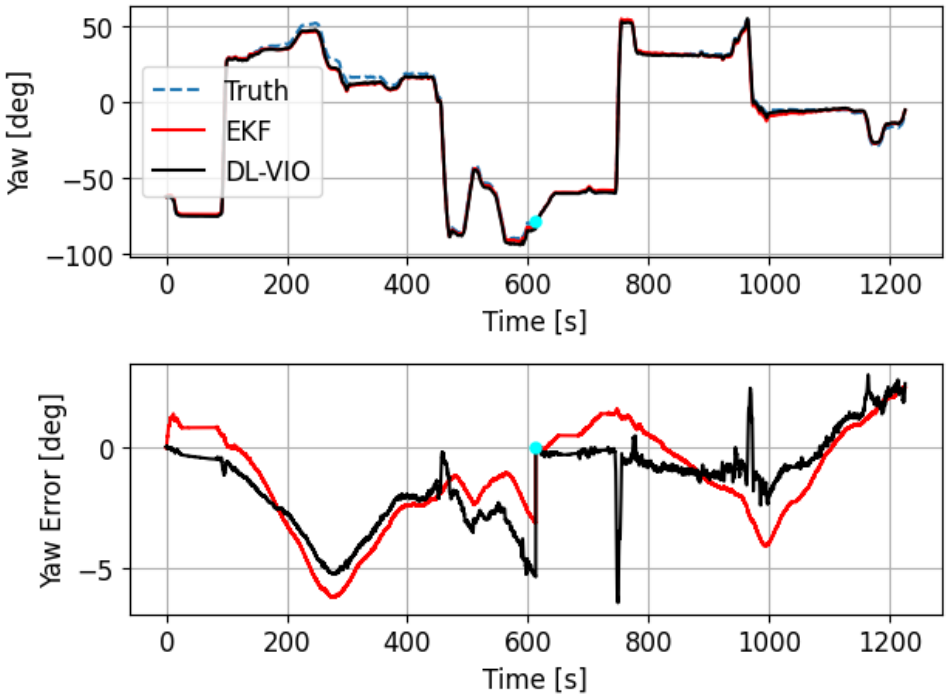


Figure A.7: The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 3.

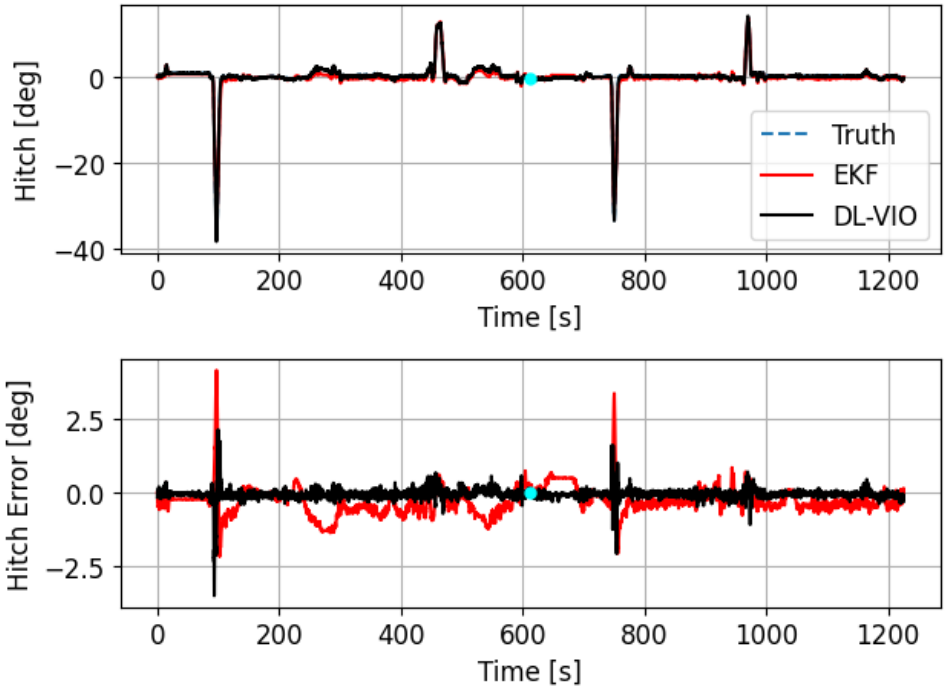


Figure A.8: The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 3.

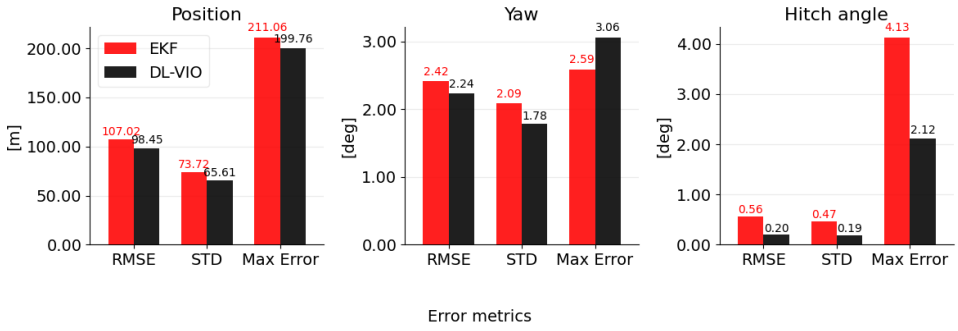


Figure A.9: Error Statistics for Sequence 3

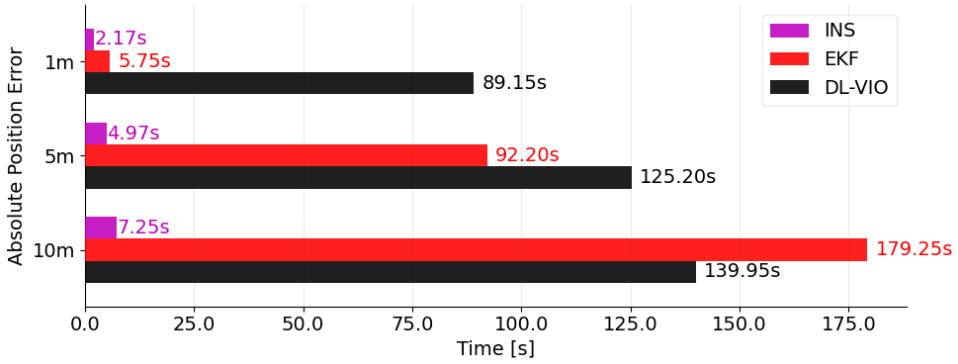


Figure A.10: Drift Times for Sequence 3

Sequence 5 Results

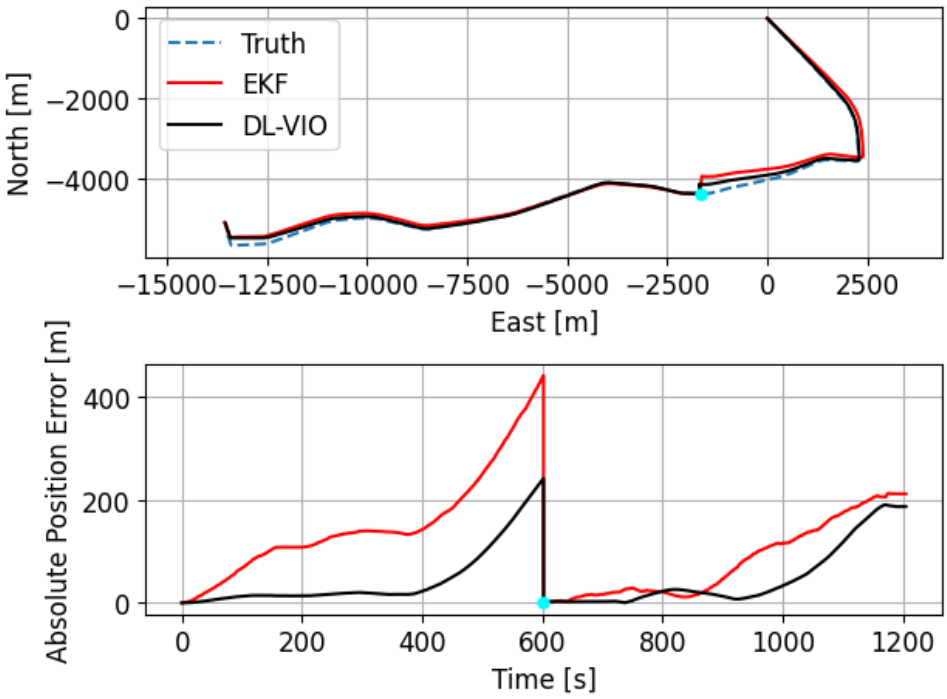


Figure A.11: The position and position errors of the model-based EKF and DL-VIO network over Sequence 5.

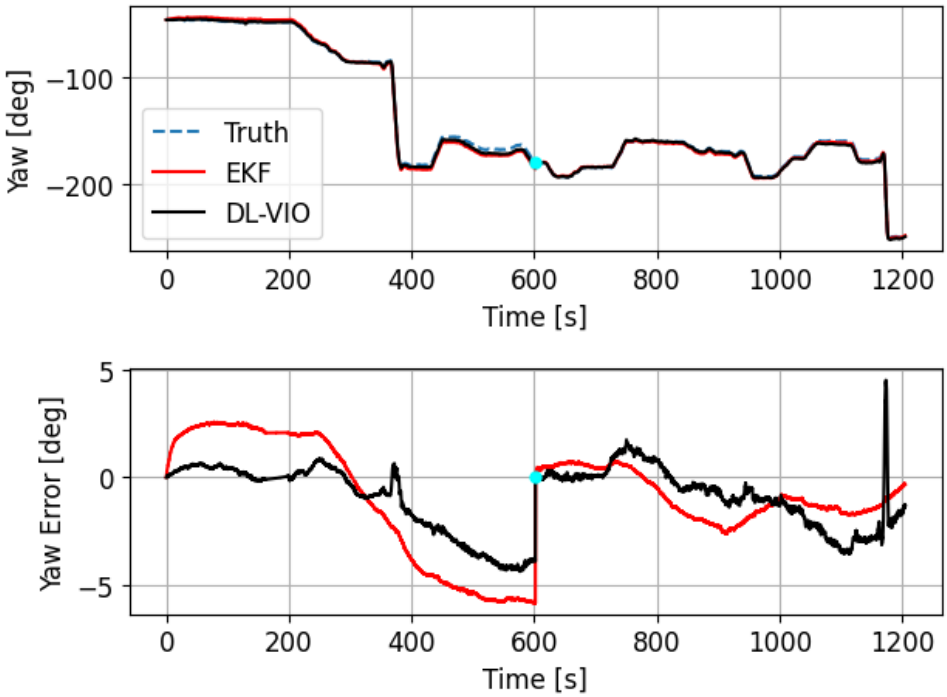


Figure A.12: The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 5.

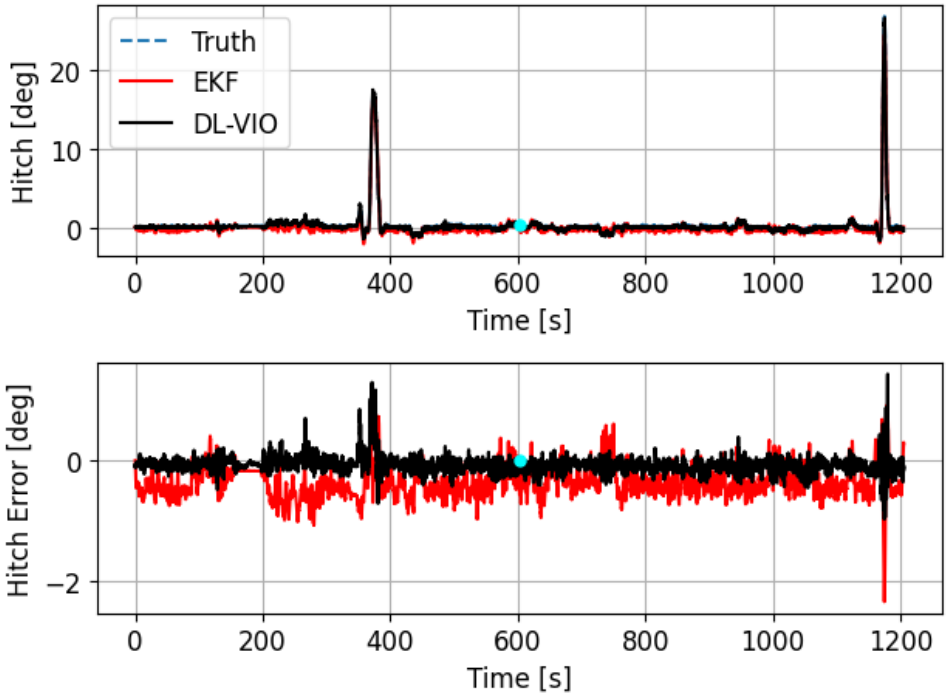


Figure A.13: The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 5.

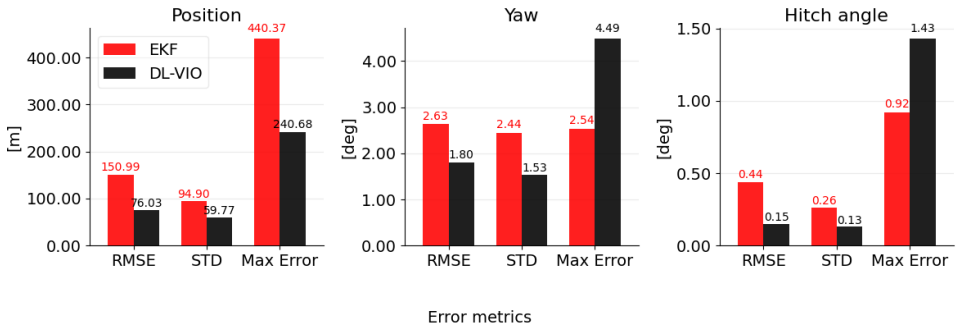


Figure A.14: Error Statistics for Sequence 5

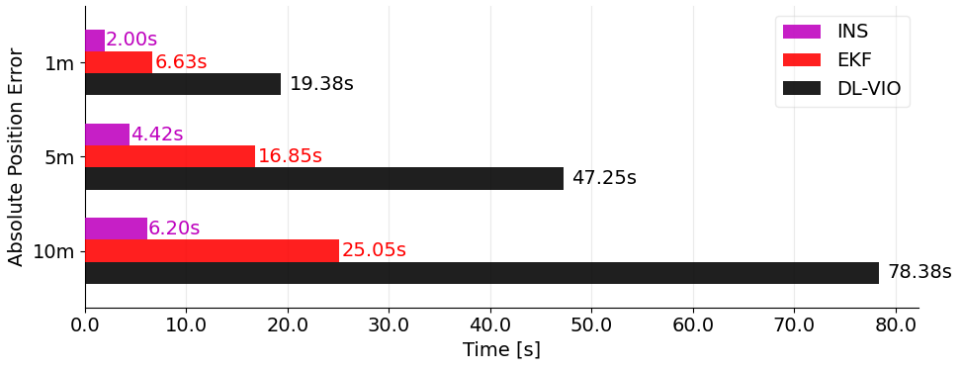


Figure A.15: Drift Times for Sequence 5

Appendix B

Swapped Training and Testing Results

In this Appendix, the training and testing datasets used for the main results are swapped for comparison. For the following results, the training set is composed of Sequences 2, 4, and 6 and the testing set is composed of Sequences 1, 3, and 5.

It was observed that by swapping the datasets, the overall performance of the DL-VIO became worse for several of the sequences. This is especially evident in the position and orientation error statistics. The hitch angle results vary depending on the sequence but is mostly similar to the results shown using the original datasets. However, it is important to note that although the overall position and orientation performance generally degraded, the DL-VIO is still able to maintain positioning accuracy for much longer periods of time than the EKF. For certain sequences (e.g., Sequences 2, 3, and 5), it was found that the times to reach certain positioning errors increased despite the overall position and orientation solutions degrading. The observed differences in the results are most likely due to a distribution mismatch between the datasets. Further investigation is necessary to determine the source of the mismatch, and what representations in the training data is required to optimally train the model.

B.1 Results on Testing Sequences

Sequence 1 Results

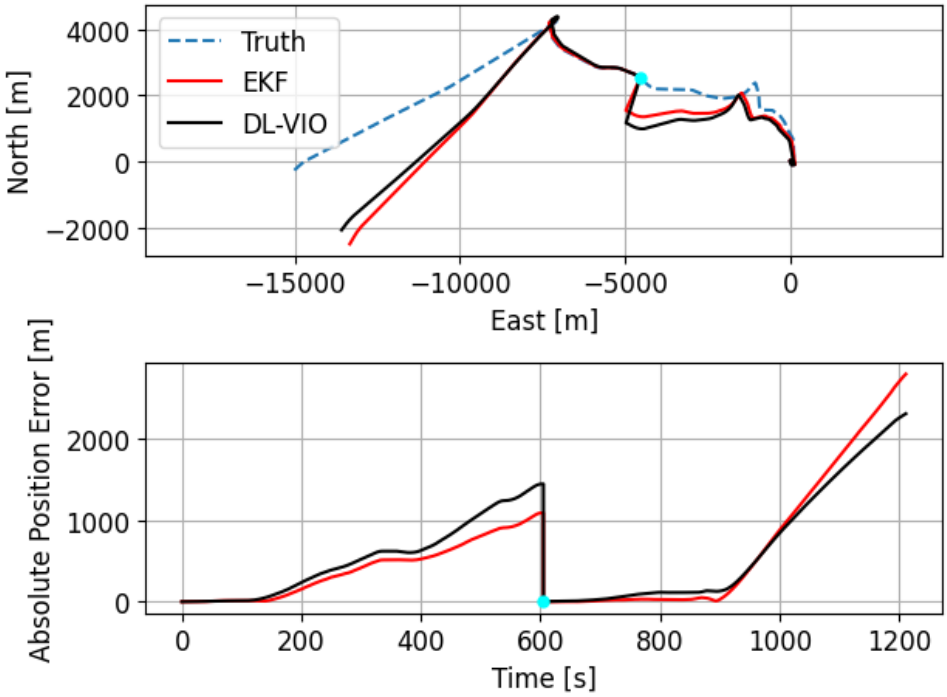


Figure B.1: The position and position errors of the model-based EKF and DL-VIO network over Sequence 1.

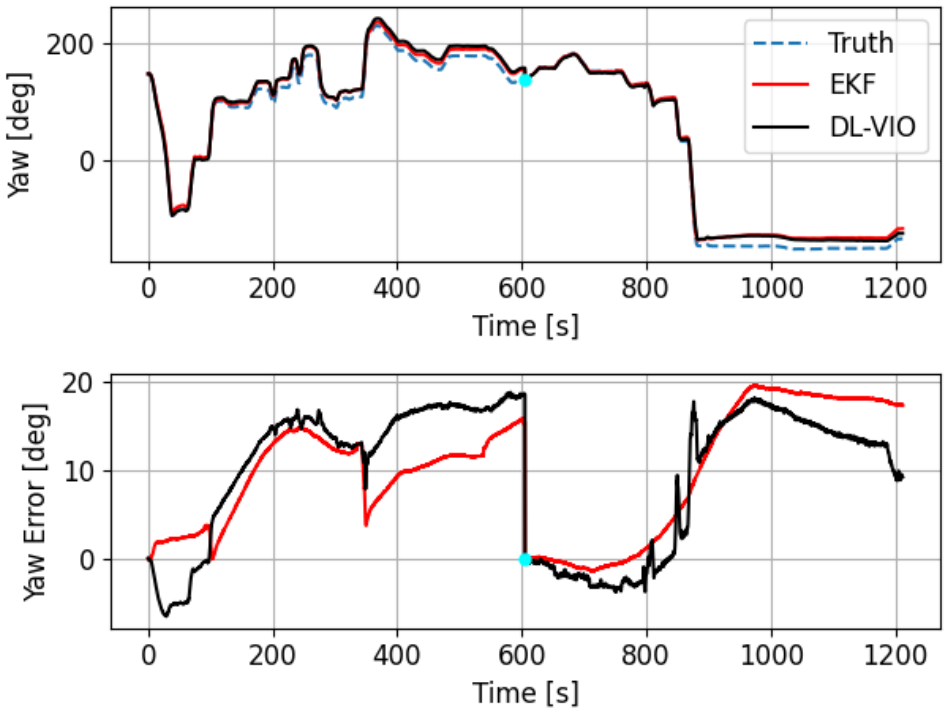


Figure B.2: The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 1.

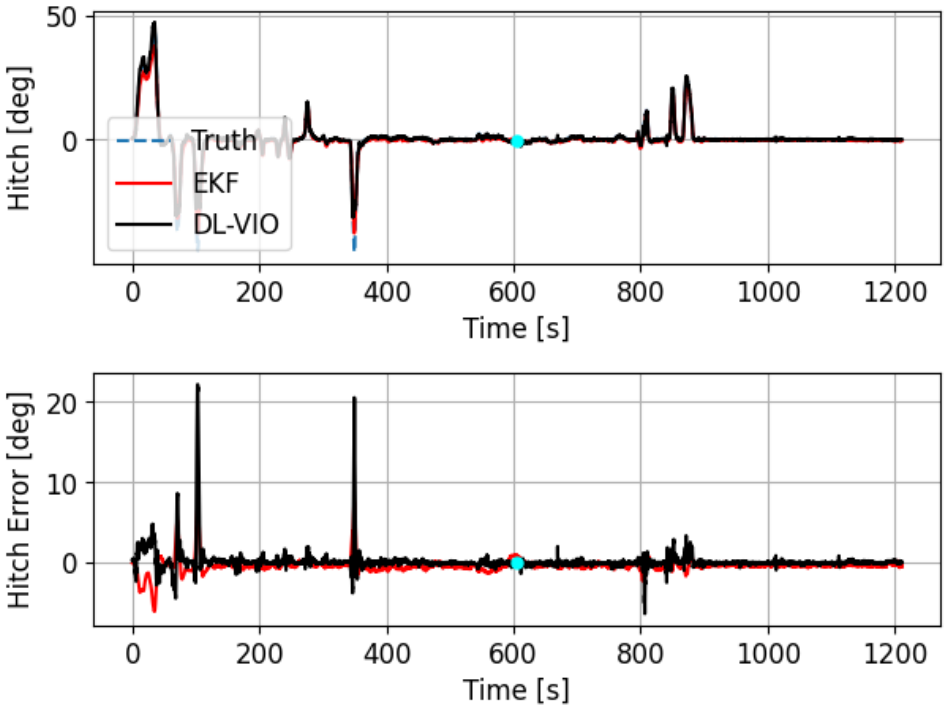


Figure B.3: The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 1.

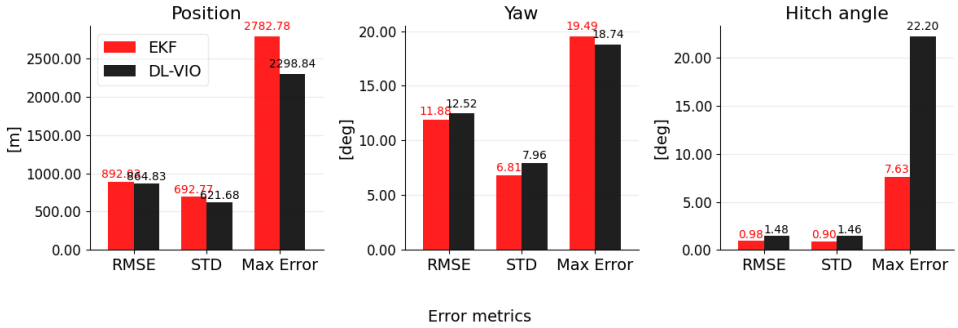


Figure B.4: Error Statistics for Sequence 1

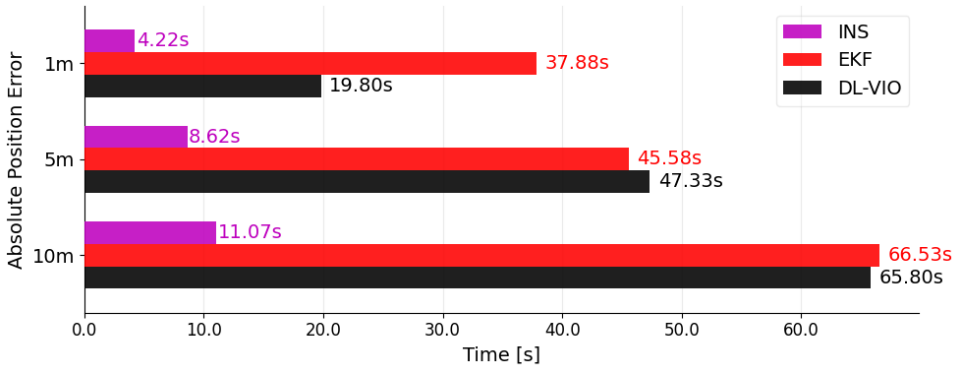


Figure B.5: Drift Times for Sequence 1

Sequence 3 Results

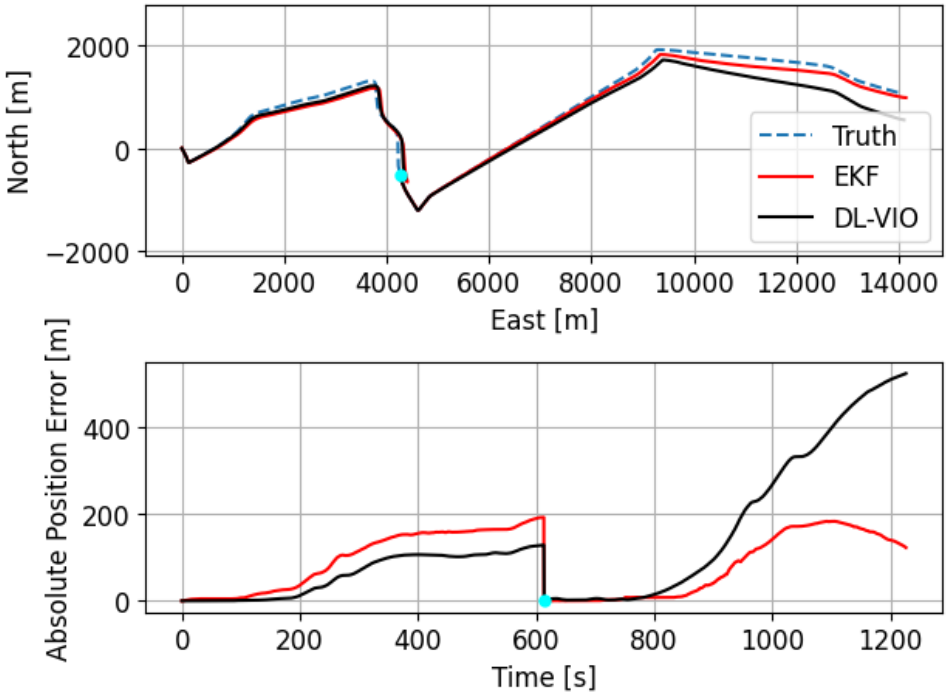


Figure B.6: The position and position errors of the model-based EKF and DL-VIO network over Sequence 3.

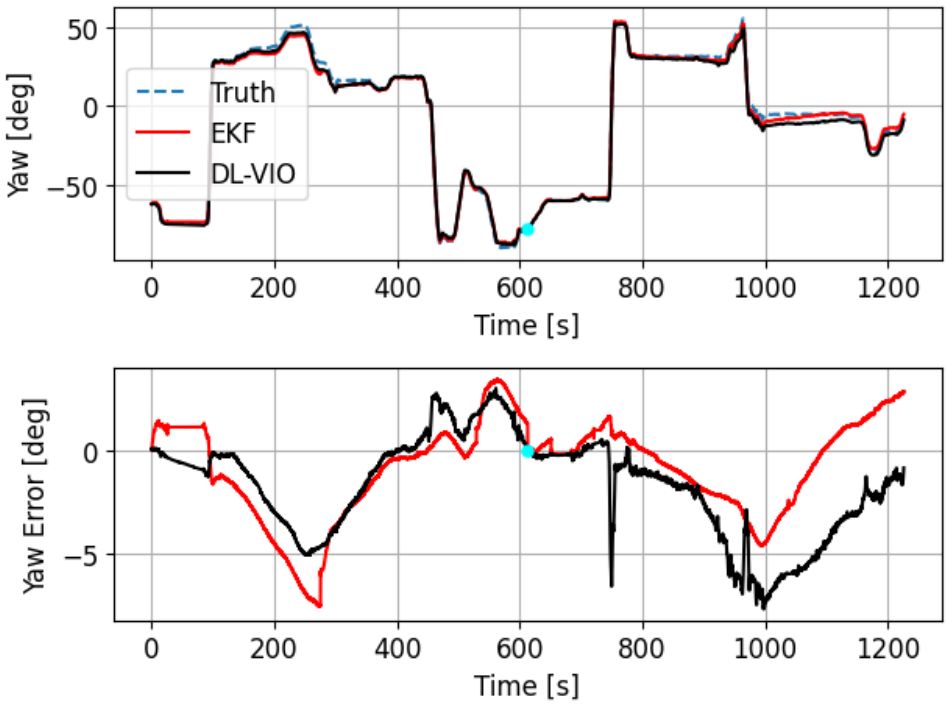


Figure B.7: The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 3.

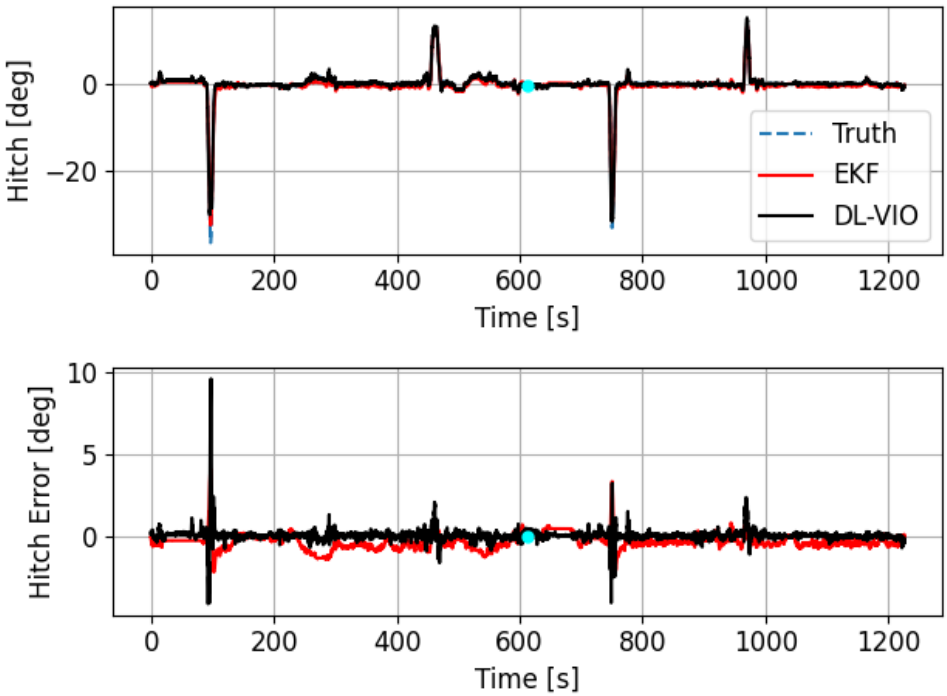


Figure B.8: The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 3.

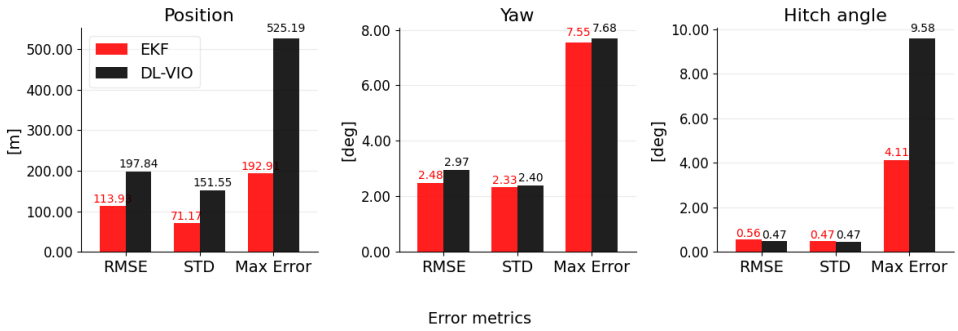


Figure B.9: Error Statistics for Sequence 3

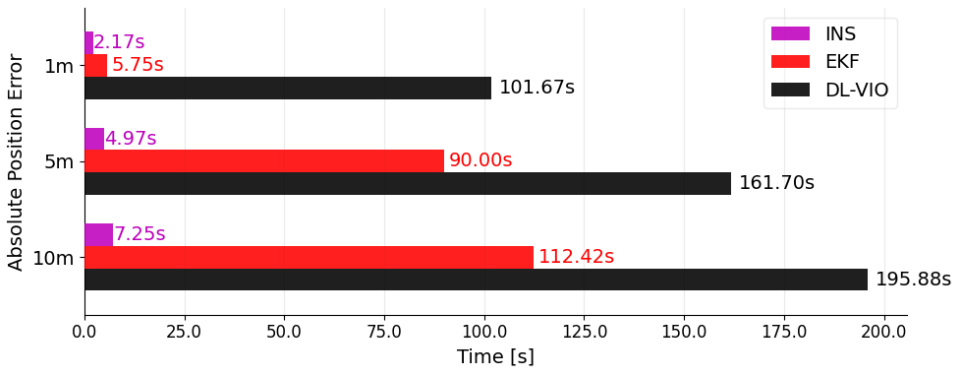


Figure B.10: Drift Times for Sequence 3

Sequence 5 Results

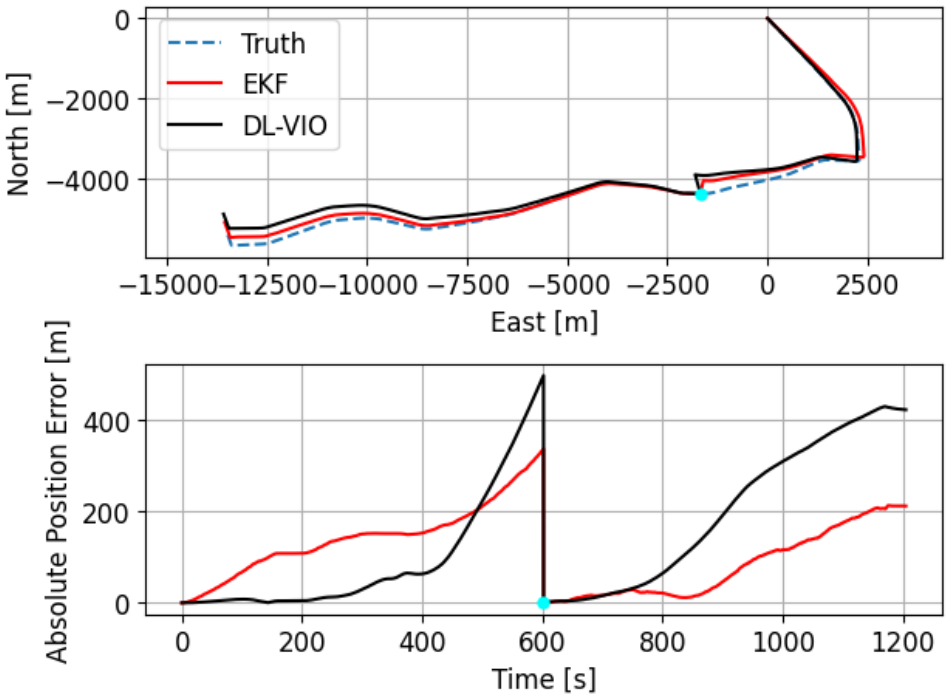


Figure B.11: The position and position errors of the model-based EKF and DL-VIO network over Sequence 5.

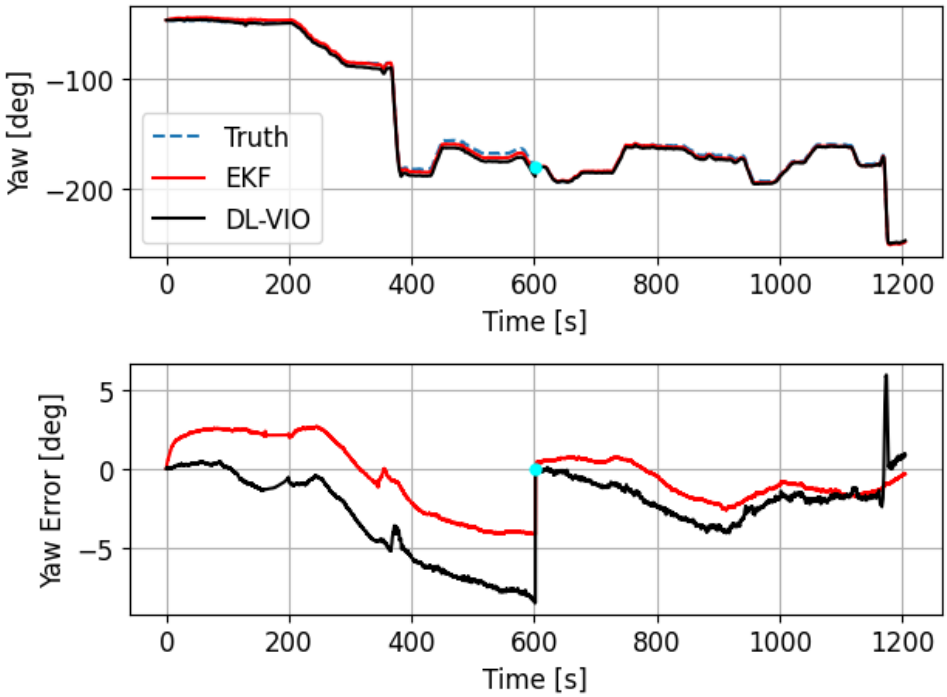


Figure B.12: The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 5.

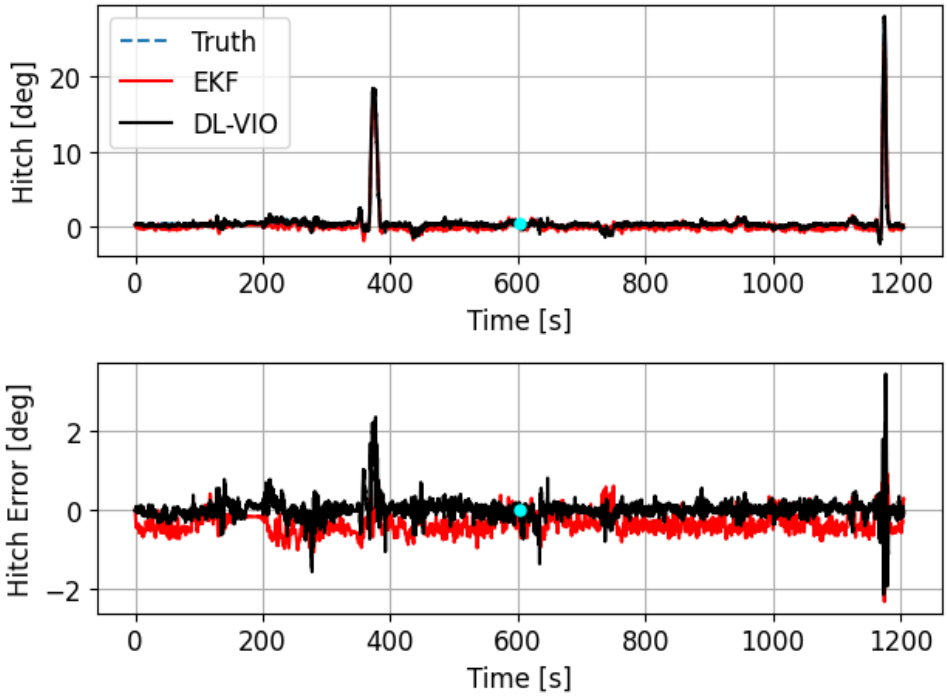


Figure B.13: The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 5.

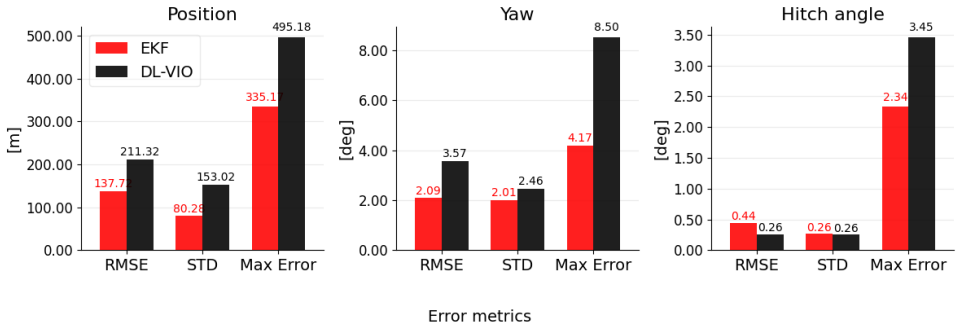


Figure B.14: Error Statistics for Sequence 5

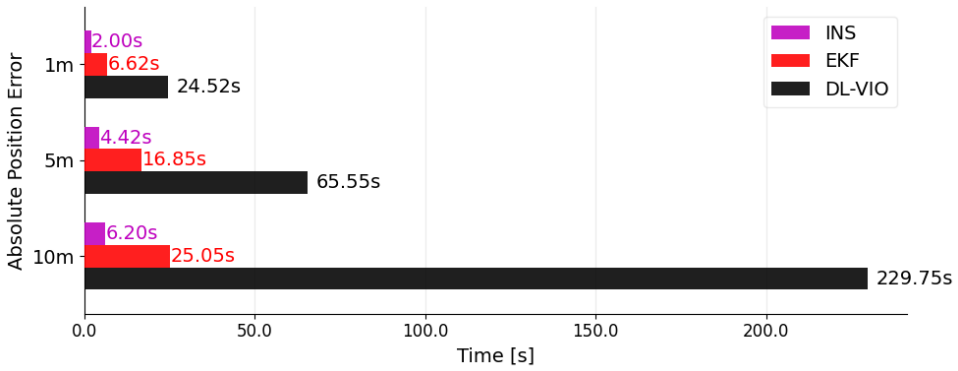


Figure B.15: Drift Times for Sequence 5

B.2 Results on Training Sequences

Sequence 2 Results

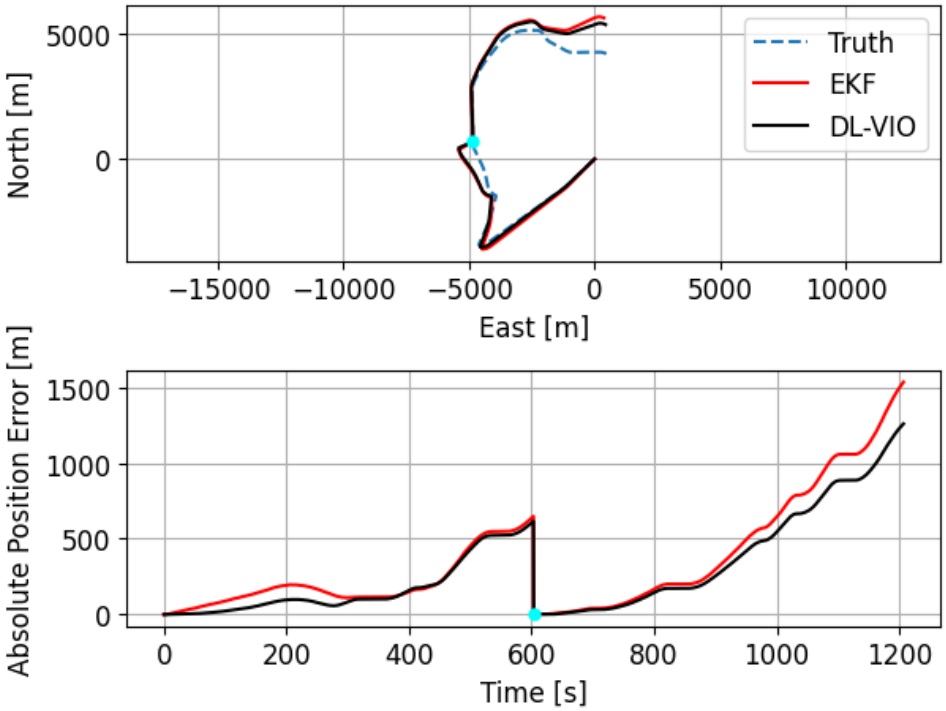


Figure B.16: The position and position errors of the model-based EKF and DL-VIO network over Sequence 1.

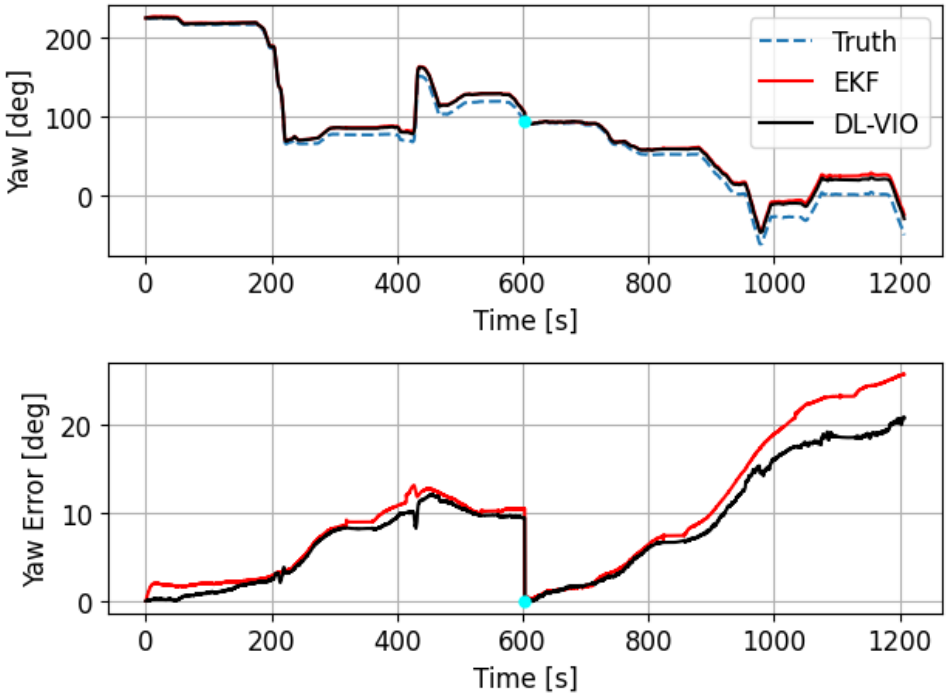


Figure B.17: The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 2.

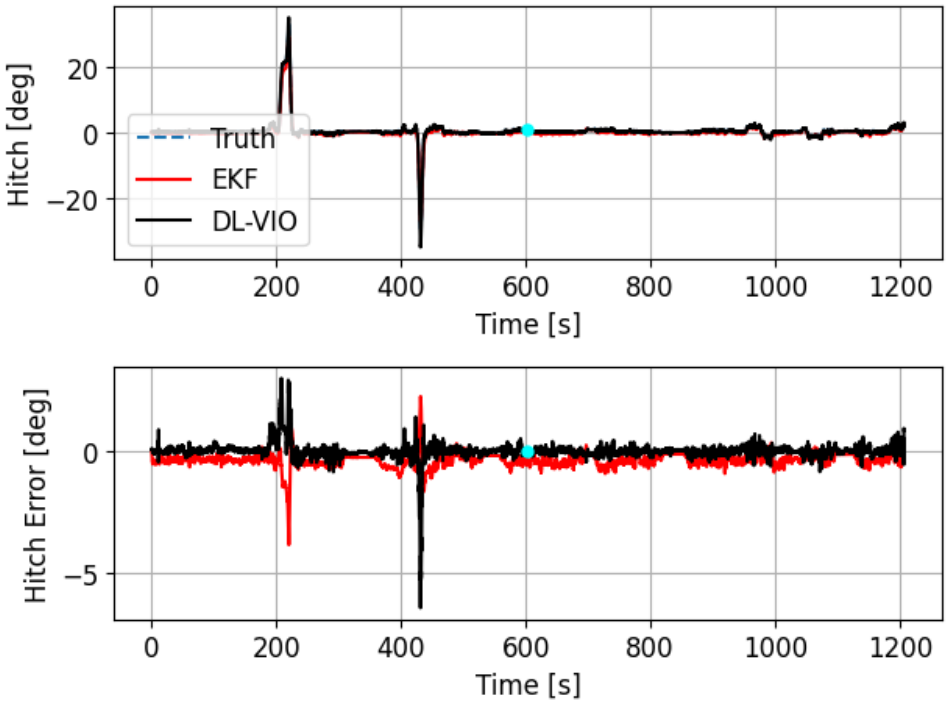


Figure B.18: The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 2.

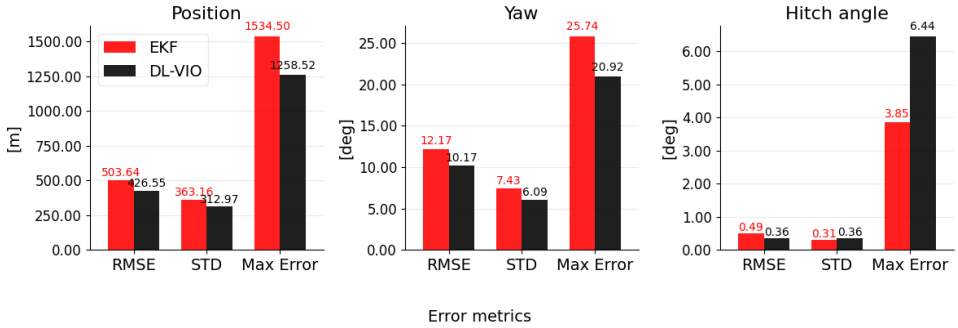


Figure B.19: Error Statistics for Sequence 2

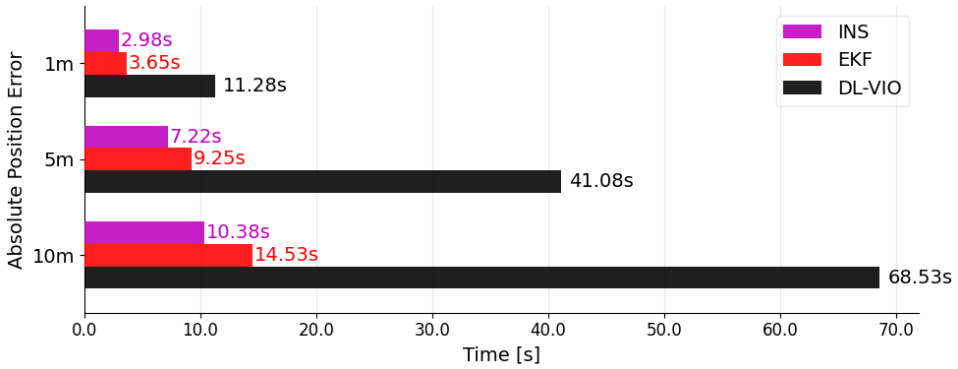


Figure B.20: Drift Times for Sequence 2

Sequence 4 Results

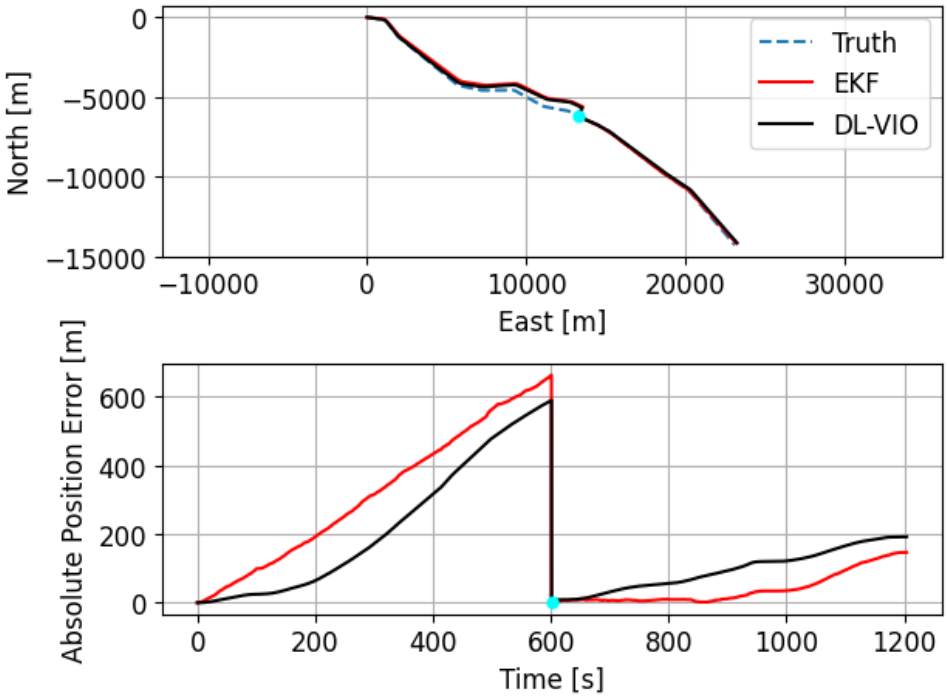


Figure B.21: The position and position errors of the model-based EKF and DL-VIO network over Sequence 4.

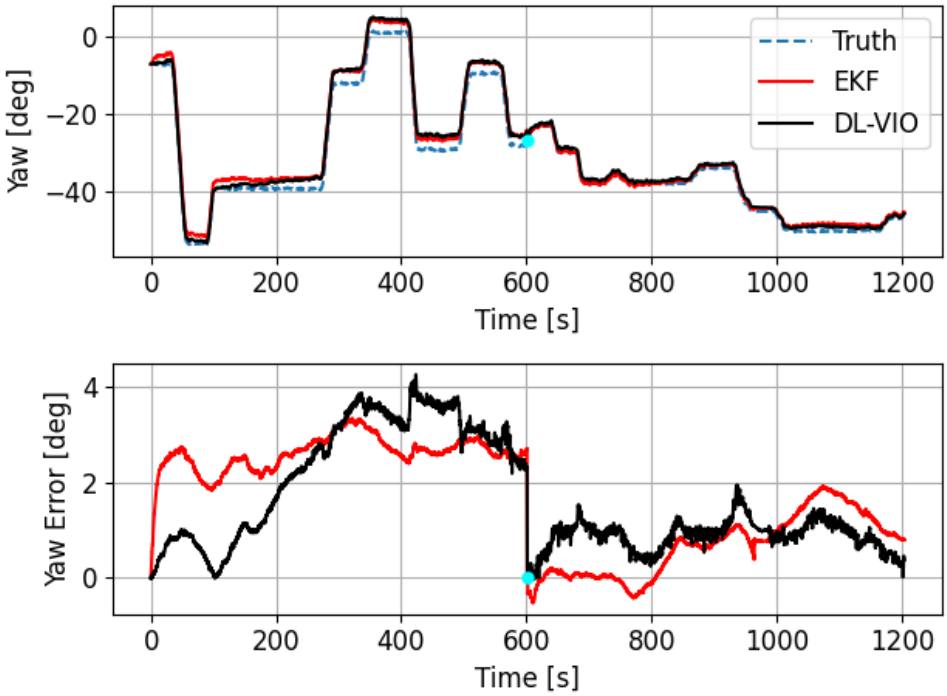


Figure B.22: The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 4.

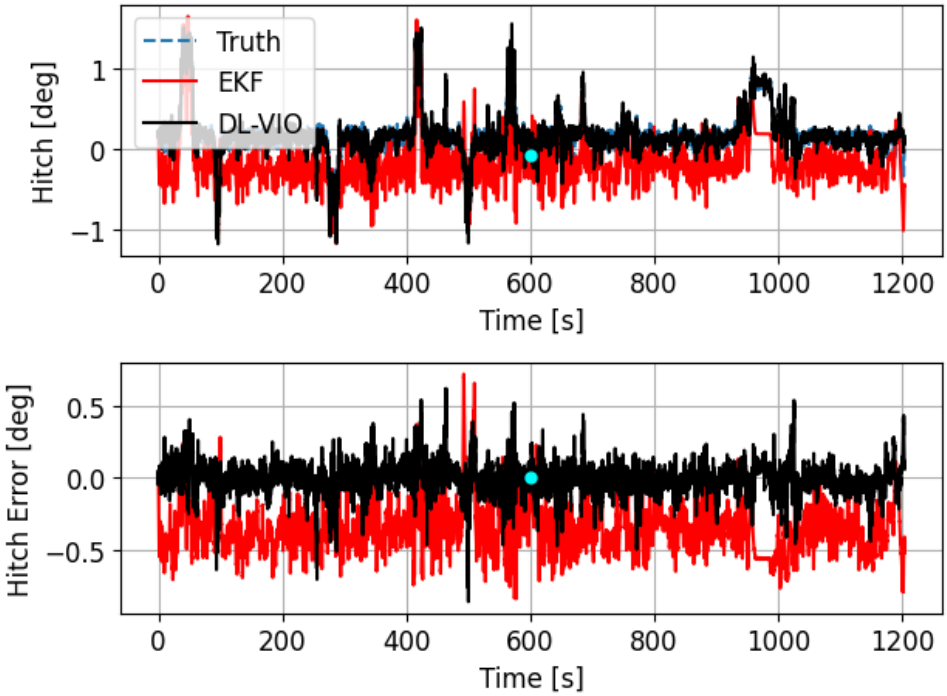


Figure B.23: The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 4.

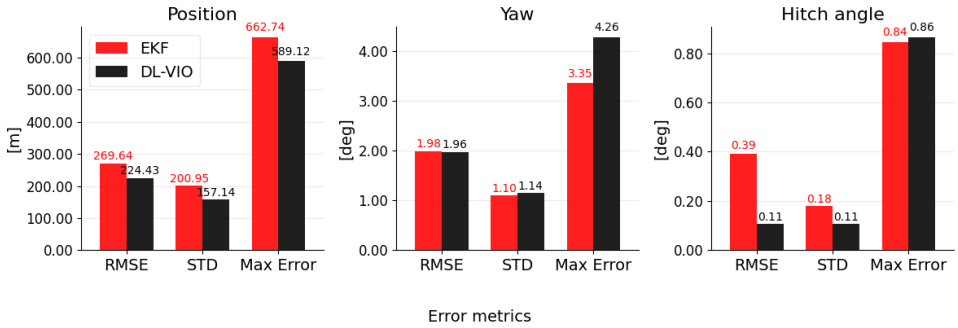


Figure B.24: Error Statistics for Sequence 4

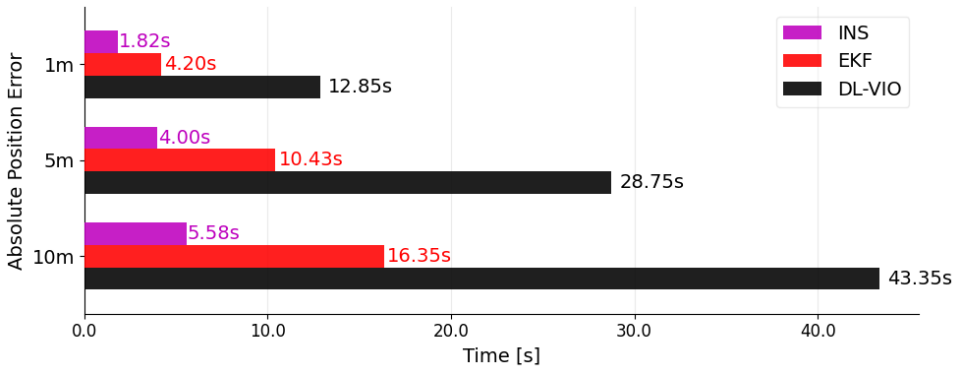


Figure B.25: Drift Times for Sequence 4

Sequence 6 Results

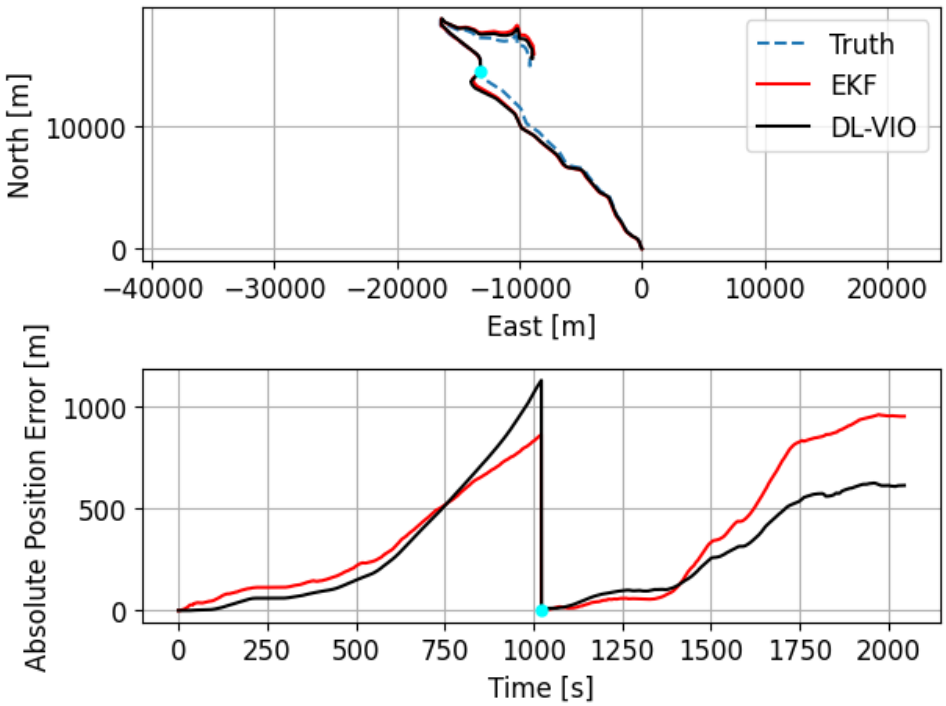


Figure B.26: The position and position errors of the model-based EKF and DL-VIO network over Sequence 6.

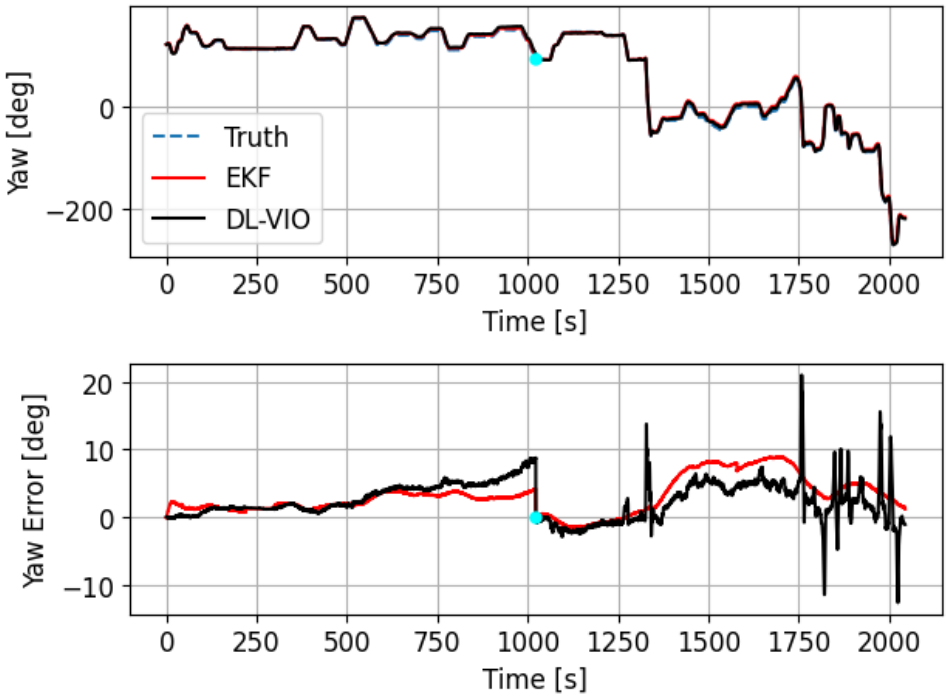


Figure B.27: The yaw and yaw errors of the model-based EKF and DL-VIO network over Sequence 6.

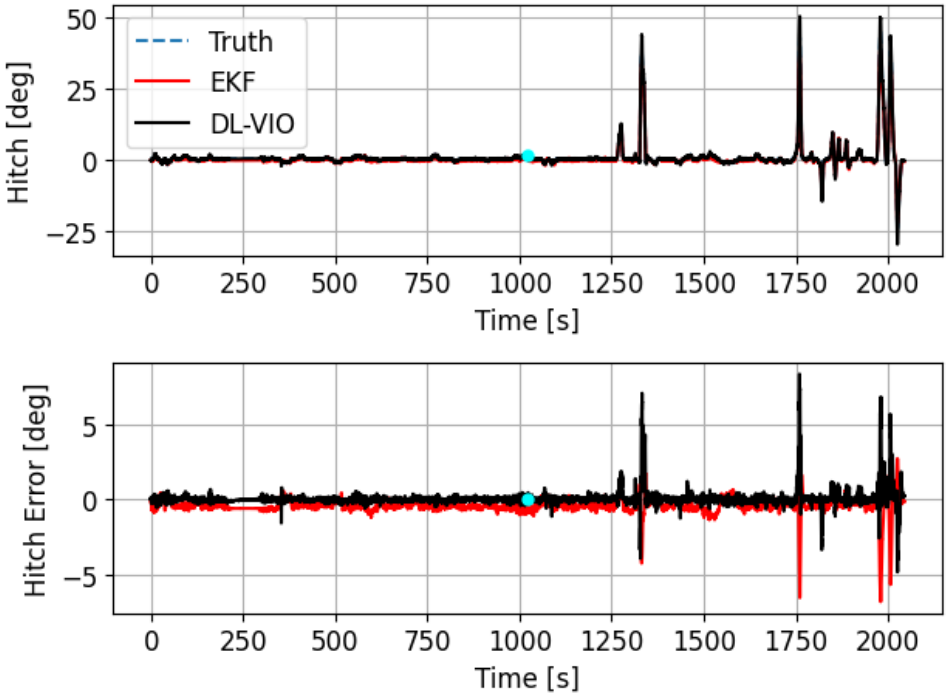


Figure B.28: The hitch and hitch errors of the model-based EKF and DL-VIO network over Sequence 6.

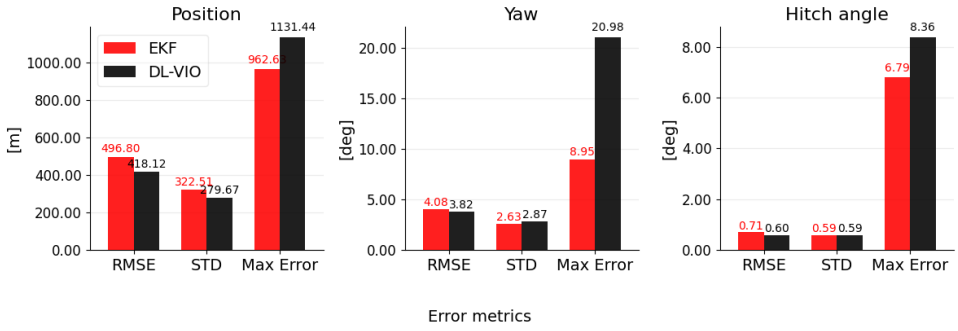


Figure B.29: Error Statistics for Sequence 6

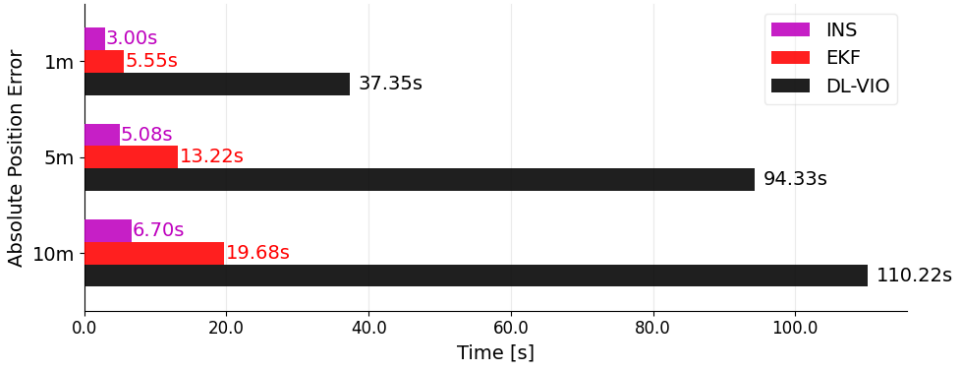


Figure B.30: Drift Times for Sequence 6