Data-Driven Models for Robust Egomotion Estimation

by

Brandon Wagstaff

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Institute for Aerospace Studies
University of Toronto

© Copyright 2023 by Brandon Wagstaff

Abstract

Data-Driven Models for Robust Egomotion Estimation

Brandon Wagstaff
Doctor of Philosophy
Graduate Department of Institute for Aerospace Studies
University of Toronto
2023

In many modern autonomy applications, robots are required to operate safely and reliably within complex environments, alongside other dynamic agents such as humans. To meet these requirements, localization algorithms for robots and humans must be developed that can maintain accurate pose estimates, despite being subjected to a range of adverse operating conditions. Further, the development of *self-localization* algorithms that enable mobile agents to maintain an estimate of their own pose is particularly important for improved autonomy. At the heart of self-localization is *egomotion estimation*, which is the process of determining the motion of a mobile agent over time using a stream of body-mounted sensor measurements. Body-mounted sensors such as cameras and inertial measurement units are self-contained, lightweight, and inexpensive, making them ideal candidates for self-localization. Traditional approaches to egomotion estimation are based on handcrafted models that achieve a high degree of accuracy while operating under a range of nominal conditions, but are prone to failure when the assumptions no longer hold. In this dissertation, we investigate how data-driven, or *learned*, models can be leveraged within the egomotion estimation pipeline to improve upon existing classical approaches. In particular, we develop a number of hybrid and end-to-end systems for inertial and visual egomotion estimation. The hybrid systems replace brittle components of classical egomotion estimators with data-driven models, while the end-to-end systems solely use neural networks that are trained to directly map from sensor data to egomotion predictions. We employ these data-driven systems for self-localization in pedestrian navigation, urban driving, and unmanned aerial vehicle applications. In these domains, we benchmark our systems on several real-world datasets, including a pedestrian navigation dataset that we collected at the University of Toronto. Our experiments demonstrate that, in challenging environments where classical estimation frameworks fail, data-driven systems are viable candidates for maintaining self-localization accuracy.

Epigraph

I have approximate answers, and possible beliefs, and different degrees of certainty about different things, but I'm not absolutely sure of anything.

Richard P. Feynman

To Sara. I couldn't have done it without you.

Acknowledgements

I'm incredibly thankful to my parents, who have been supportive throughout the many years I've been a student. I'm also grateful to my friends, who have brought so much joy along the way.

To all of the members of STARS Laboratory, you have made this journey such an incredible experience. I will cherish all of the memories we made along the way; including time spent at UTIAS, intramural soccer and baseball, lab socials, conferences abroad, and especially the late-night Green Room conversations. Special thanks goes to Valentin, who helped me find my footing and was a constant source of support.

Lastly, to Jonathan, I'm indebted to you for all of the support, guidance, and encouragement you have provided over the years.

Contents

Notation

a	:	Non-bold, non-capitalized symbols are real scalars.
$\mathbf{a}$	:	Bold, non-capitalized symbols are real column vectors.
$\mathbf{A}$	:	Bold, capitalized symbols are real matrices.
$\mathbf{I}_n$	:	An $n \times n$ identity matrix.
$\mathbf{0}_{n\times m}$	:	An $n \times m$ matrix of zeros. The subscript is optional as the dimensionality can often be inferred from context.
$\mathbf{1}_{n\times m}$	:	An $n \times m$ matrix of ones.
$\mathcal{N}(\boldsymbol{\mu}, \mathbf{R})$	:	Normally distributed random variable with mean vector $\boldsymbol{\mu}$ and covariance matrix $\mathbf{R}$.
$\mathbb{E}[\cdot]$	:	The expectation operator.
$\underrightarrow{\mathcal{F}}_a$	:	A reference frame in three dimensions.
$\mathbf{p}_a^{cb}$	:	A vector from point b to point c (denoted by the superscript) and expressed in $\underrightarrow{\mathcal{F}}_a$ (denoted by the subscript).
$\mathbf{R}_{ab}$	:	The 3×3 rotation matrix that transforms vectors from $\underrightarrow{\mathcal{F}}_b$ to $\underrightarrow{\mathcal{F}}_a$.
$\mathbf{T}_{ab}$	:	The 4×4 transformation matrix that transforms homogeneous points from $\underrightarrow{\mathcal{F}}_b$ to $\underrightarrow{\mathcal{F}}_a$.
$(\cdot)^\wedge$	:	An operator associated with the Lie algebra for rotations and poses. It produces a matrix from a column vector.
$(\cdot)^\vee$	:	The inverse operation of $(\cdot)^\wedge$.
$\dot{(\cdot)}$	:	The time derivative of a scalar, vector, or matrix.
$\mathbf{a} \oplus \mathbf{b}$	:	A composition operator that represents the addition operator for vectors and the product operator for rotations.
$\mathbf{a} \odot \mathbf{b}$	:	The Hadamard (or element-wise) product, $\odot$, is used for element-wise multiplication between two vectors or matrices.
$\pi(\cdot)$	:	The pinhole camera projection model that maps a 3D point to its 2D pixel coordinate within an image. See Equation (2.75) for its definition and Equation (2.76) for the inverse projection model, $\pi^{-1}(\cdot)$.

Chapter 1

Introduction

> You step into the Road, and if you don't keep your feet, there is no knowing where you might be swept off to.
>
> J.R.R. Tolkien

Localization—the process of determining the pose (i.e., the position and orientation) of a robot or other moving body relative to a fixed external reference frame—is fundamental to mobile autonomy. Robust localization is essential for safe and reliable navigation, a key requirement for the wide-scale deployment of autonomous robots in the near future and beyond (Cadena et al., 2016). In many situations, mobile robots must operate alongside other dynamic agents, such as humans, in complex environments. For these applications, algorithms for reliable localization of robots *and* humans will play a critical role in efficient and safe mobile autonomy solutions.

Self-localization is the ability of a robot to maintain an estimate of its own pose. A fundamental part of self-localization is *egomotion estimation*. Egomotion, or the motion of a body relative to its surrounding environment, is typically estimated with respect to a fixed origin, using measurements from sensors mounted on the body.[1] By relying only on a set of self-contained sensors, egomotion estimation does not require the presence of external infrastructure. This form of relative pose estimation is particularly useful when global navigation aids, such as global navigation satellite system (GNSS) signals, are unavailable (e.g., when indoors, underground, or sometimes when travelling through urban areas). Furthermore, egomotion estimation is a fundamental component of simultaneous localization and mapping (SLAM) systems (Durrant-Whyte and Bailey, 2006).

Among the sensors that are available for egomotion estimation, inertial measurement units (IMUs) and cameras are popular choices; they are inexpensive, lightweight, and capable of producing six degree-of-freedom (DOF) pose estimates. Body-mounted (or *strapdown*) inertial sensors provide specific force and angular velocity measurements, which can be integrated over time through a process known as *dead reckoning* to update the body pose relative to a previously known pose. Egomotion estimation through (inertial) dead reckoning is a localization strategy for human (i.e., pedestrian) navigation (Hou and Bergmann, 2020). Visual egomotion estimation (i.e., *visual odometry*), alternatively, is a well-known self-localization strategy used by a wide variety of autonomous systems (Aqel et al., 2016). Cameras are able to capture a rich visual representation of the environment at a high frame rate. Egomotion estimation is usually performed by detecting environmental landmarks in each camera image frame (i.e., visual features) and tracking the landmarks between frames. The joint estimation

[1] Herein, we use the term 'body' generically to represent any mobile agent, such as a vehicle, robot, or human.

of environmental landmark positions and camera motion defines the *structure from motion* (SfM) problem that appears frequently in computer vision literature; an optimal batch solution can be obtained using the bundle adjustment algorithm (Triggs et al., 1999).

Inertial and visual sensors can also be employed together in systems where data are fused online for improved egomotion estimation (Gui et al., 2015). Such visual-inertial systems have a number of advantages due to the complementary nature of the sensors. Visual measurements mitigate the significant error build-up that occurs during IMU-based dead reckoning, while the IMU measurements help maintain accuracy during periods when vision is degraded. Furthermore, the availability of metric information (i.e., distances relative to a known reference) in the IMU measurements allows for the true scale of the camera motion (and scene geometry) to be recovered. Without the inclusion of IMU measurements, (monocular) vision-only systems are only able to determine egomotion up to an unknown scale.

Algorithms for inertial, visual, and visual-inertial egomotion estimation are traditionally based on filtering or optimization frameworks that produce a probabilistic estimate of the relevant system *state* from a stream of (noisy) sensor measurements (Gui et al., 2015). The state estimate, at minimum, includes the body pose, but may also incorporate other useful quantities such as the body velocity and certain sensor parameters (e.g., calibration values). Traditional algorithms all rely on modelling assumptions to make the estimation problem tractable. Although these simplifications are reasonable when operating under 'nominal' conditions, there is an increasing need to deploy navigation systems in more challenging settings. Within these more complex environments (e.g., involving dynamic objects, poor illumination conditions, or sensors being subjected to rapid motions), assumptions may fail to hold, resulting in a failure to maintain an accurate state estimate over time.

This thesis investigates how data-driven, or *learned*, models can improve the robustness and accuracy of inertial and visual egomotion estimation algorithms. Herein, we identify key failure modes of classical systems and determine how we can augment or replace these *brittle* components with robust, data-driven alternatives. We develop and examine a number of self-localization algorithms based on inertial and visual egomotion estimation and demonstrate their utility (in both indoor and outdoor environments) for applications including human and robot localization.

1.1 Background and Motivation

Navigation, the field of study that focuses on the process of monitoring and controlling the movement of a body from one place to another, is a foundational part of mobility and mobile autonomous systems (Farrell, 2008). Being deeply rooted in seafaring[2], advances in navigation were initially motivated by the need to safely and accurately traverse large bodies of water, facilitating exploration and trade across the globe, and were tied to the tools and technologies available to navigators. These tools were generally developed to determine one's position relative to other known locations of interest. As early as the fifth millennium BCE, Indigenous seafarers of the Pacific Northwest travelling in dugout canoes used the coastline as a visual reference for self-localization. With the expansion of seafaring, spurred on by exploration and trade, more advanced tools for self-localization, including the astrolabe, compass, and marine chronometer, were designed. By the 20th century, techniques developed in the aviation industry led this field of navigation, with humans in the loop, to a point of maturity. The advent of space exploration—and the need for accurate navigation of unmanned systems—motivated the development of *self-navigation* techniques, which apply self-localization algorithms to track the state of a body (e.g., a rocket). A noteworthy example is the Kalman filter (Kalman, 1960), which, among other applications,

[2]Navigation stems from the Latin word *navigare*, meaning to sail, go by sea, steer a ship, which in turn comes from *navis*, or ship.

was used to estimate the trajectory of the Apollo 11 Command Module on its journey to and from the Moon in 1969 (McGee, 1985). In recent years, modern sensing advances have been made in the domains of aviation, space exploration, and robotics. Notably, global navigation satellite systems (GNSS) have been deployed by several countries, enabling absolute positioning worldwide in many situations. A GNSS receiver is now a standard instrument for navigation.

The coverage provided by absolute positioning aids, however, is not ubiquitous (in space or time). For example, systems such as GNSS are prone to experiencing periods of 'dropout' within urban canyons, inside buildings, or when moving underground or underwater. During dropout, the current position (or pose) of a body can be estimated through dead reckoning. Dead reckoning is a form of egomotion estimation that uses estimates of velocity and orientation over time to extrapolate changes in position (or pose) relative to a previously known position. Modern dead reckoning typically relies on an *inertial navigation system* (INS) for egomotion estimation. The physical INS consists of an IMU that incorporates three orthogonally mounted accelerometers (producing specific force readings) and three orthogonally mounted gyroscopes (producing angular velocity readings). These measurements are integrated through time (Woodman, 2007): orientation updates are produced via direct integration of the angular velocity measurements; position updates are produced via double integration of the specific force terms that have been transformed to the (gravity-aligned) world reference frame. Alignment relative to gravity is determined through the orientation estimate and is necessary to remove the effect of the Earth's gravity from the accelerometer measurements. An important characteristic of modern INSs is their ability to provide pose updates at a very high rate, usually between 100 Hz and 1,000 Hz.

Recent advancements in micro electro-mechanical systems (MEMS) technology have led to the availability of low-cost, lightweight IMUs, which are well suited for self-localization across a broad range of applications and environments. These low-cost IMUs, however, are subject to a high degree of noise and sensor drift. Continuous integration of the measurement noise results in position estimates that accumulate a position error that grows cubically with time (Skog et al., 2010b). Notably, this build-up of error can lead to navigation failure within a matter of seconds. To mitigate error growth, the INS must incorporate data from other measurement sources, in a process known as 'aiding,' whereby the state (positioning) estimate is constrained more precisely.

1.1.1 Aided Inertial Navigation

The process of correcting the inertial state estimate by incorporating measurements from additional sensors is called *aided* inertial navigation. Aiding is required to maintain accuracy over long periods of time or long distances. GNSS-aided navigation has been well studied (Farrell, 2008). As previously discussed, however, GNSS measurements are subject to dropout indoors and within urban environments (among others). Many localization systems, instead, rely on a suite of sensors that may include cameras, radars, lidars, and wheel encoders. All of these measurements are available for aided inertial navigation.

To use these measurements to improve the localization (or state) estimate, the relationship between each sensor measurement and the state variables of interest (e.g., position, velocity, or orientation) must be accurately modelled. For direct measurements of global position from GNSS, for example, this mapping is trivial. For other modalities, however, the relationship between raw sensor data and the state variables may be challenging to model. For example, in vision-aided navigation, modelling the entire distribution of possible sensor readings as a function of the state (i.e., position in the world) is not feasible. Instead, simplifying assumptions are necessary to make the problem tractable. In the case of visual aiding, specifically, simplifications are made by incorporating a *front-end* preprocessing stage that identifies and extracts *features* from images to reduce the dimensionality of the measurements. Features are stable environmental landmarks that have a consistent appearance when

Table 1.1: Comparison of some key characteristics of classical and learned visual and inertial egomotion estimation methods.

Characteristic	Classical Models	Learned Models
Required Domain Knowledge	High — models must be carefully crafted using domain knowledge	Low — models are learned from data
Number of Modelling Assumptions	High	Low
Number of Tunable Parameters	Many test-time parameters that affect performance	Few at test-time[1]
Flexibility	Low — constrained by modelling assumptions	High
Generalization Capability	High — but additional parameter tuning required	Low — with potential to improve
Accuracy	High — when modelling assumptions hold	Moderate — with potential to improve

[1] Learned models do have many hyperparameters that require manual tuning, but these only need to be determined prior to training. The resulting model, properly trained, requires little to no parameter tuning at test-time.

viewed under limited camera perspective changes. By observing the same features in multiple (nearby) images, feature correspondences can be determined and applied for egomotion estimation (Aqel et al., 2016).

Measurement models must be carefully crafted in order to faithfully represent the relationship between sensor outputs and the states in question. These 'handcrafted' models have a number of downsides. First, they require a significant amount of domain knowledge to create. Second, being sensor- and application-specific, handcrafted models often employ a number of tuning parameters whose values must be properly selected for the model to operate reliably under different conditions. Third, the number of simplifying assumptions used within these models can result in brittle behaviour, where localization is accurate under nominal conditions but prone to failure when the assumptions are violated, sometimes even only slightly. These limitations must be addressed in order for egomotion estimation to be a viable form of self-localization in modern applications, where algorithms must be robust to failure in a wide operating range.

1.1.2 Data-Driven Egomotion Estimation

The emergence of deep learning (Goodfellow et al., 2016) has resulted in a surge of data-driven systems being proposed for a range of applications. In the field of computer vision, for example, deep learning has improved the state of the art in a number of tasks such as image-based object classification and detection (Voulodimos et al., 2018). Inspired by these recent successes, a nascent application involves the use of deep learning within visual and inertial localization pipelines. Work in this area generally falls within one of two broad categories: *end-to-end* systems that involve the complete replacement of the localization system with one or more neural networks, or *hybrid* systems that merge data-driven models with existing localization pipeline components.

End-to-end learned systems for localization obviate the need for carefully designed (handcrafted) models. Instead, neural networks parameterize a direct mapping from sensor to state, and are trained (through supervised or self-supervised learning) to minimize a loss function across a training dataset. Recently, end-to-end systems have been proposed for camera relocalization (Kendall et al., 2015), SLAM (Teed and Deng, 2021), and visual (Wang et al., 2017), inertial (Chen et al., 2018), and visual-inertial (Clark et al., 2017) egomotion estimation. Table 1.1 lists a number of key characteristics that differentiate these end-to-end systems from their classical counterparts. Notably, end-to-end systems require less domain knowledge to build, rely on fewer modelling assumptions, and have fewer tunable parameters in general. Importantly, owing to the large modelling capacity

of neural networks (i.e., the ability to fit a wide variety of functions), the flexibility of end-to-end systems is superior: neural networks can be trained to model highly complex relationships that do not have known analytical solutions. Accordingly, there is a strong motivation to utilize these networks within robust self-localization algorithms. There are, however, some limitations that have prevented their widespread deployment. Namely, under nominal operating conditions where classical modelling assumptions *do* hold, end-to-end systems generally have been unable to match the accuracy of classical approaches. In addition, generalization to new data (i.e., environments) beyond the initial training distribution continues to be an outstanding issue for neural networks.

Alternatively, hybrid systems augment (rather than replace) classical estimators with learned components. For example, as part of the visual egomotion estimation pipeline, Yang et al. (2020) couple a learned depth estimator with an optimization-based back-end; Peretroukhin et al. (2017) incorporate a learned model to estimate illumination direction and limit orientation drift; and Clement and Kelly (2018) and Tomasi et al. (2021) use learned models within the front-end of their visual localization pipelines to improve image quality. By combining learning with classical techniques, these hybrid algorithms aim to retain the interpretability and transferability of traditional estimation frameworks while leveraging the capacity and flexibility of data-driven models to improve accuracy and robustness. We draw inspiration from the success of these hybrid approaches and follow a similar strategy of preserving well-modelled classical components while replacing the brittle components with learned models.

1.2 Thesis Contributions

Herein, we investigate how data-driven models can be incorporated into classical egomotion estimation pipelines to improve their overall performance. By leveraging the capacity and flexibility of learned models, our goal is to design *robust* egomotion estimators that are accurate under a wide range of challenging conditions where classical systems fail. Figure 1.1 depicts the general structure of the systems described in this thesis. We focus on self-localization for two key navigation applications: pedestrian navigation and autonomous vehicle navigation. Our approach is based on the aided inertial navigation paradigm. We present two data-driven models: a learned zero-velocity update model for aided pedestrian navigation (Foxlin, 2005) and a learned egomotion measurement model for self-localization of autonomous vehicles. Finally, we describe a novel visual-inertial estimator that uses our learned egomotion measurement model for aiding. We place an emphasis on self-supervised learning techniques, which facilitate lifelong learning by allowing our models to be retrained as new data is acquired. Our contributions are summarized below:

1. **Robust zero-velocity detection for foot-mounted inertial navigation**
   The zero-velocity-aided INS is a well-studied and popular solution for IMU-based pedestrian navigation (Foxlin, 2005; Nilsson and Handel, 2014). During bipedal locomotion, a foot-mounted IMU repeatedly passes through a stationary phase, at which point the pose estimation error can be corrected by applying a 'zero-velocity update.' When correctly identified by a *zero-velocity detector*, which uses IMU measurements to estimate when the foot is stationary, the zero-velocity updates can significantly improve localization accuracy. However, false-positive detections cause the length of the wearer's trajectory to be underestimated, while false-negative detections (i.e., missed detections) lead to rapid and unbounded error growth.

   We present two novel, data-driven techniques for detecting zero-velocity events to improve zero-velocity-aided inertial localization. Our first technique augments a classical zero-velocity detector by incorporating

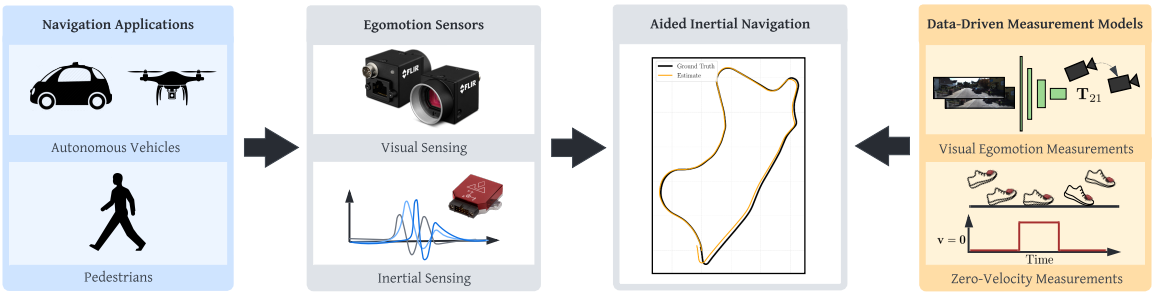


Figure 1.1: An overview of our approach for robust egomotion estimation through aided inertial navigation. We investigate how data-driven, or *learned*, measurement models can augment the baseline INS. Specifically, we develop a learned zero-velocity measurement model for aided pedestrian navigation and a learned visual egomotion measurement model for autonomous vehicle localization.

a motion classifier that adaptively updates a threshold parameter within the detector. Our second technique uses a recurrent neural network to classify zero-velocity events from raw inertial data. This is in contrast to the majority of zero-velocity detection methods that rely on statistical hypothesis testing. We demonstrate that our data-driven detectors outperform traditional handcrafted detectors on challenging indoor navigation datasets consisting of walking, running, and stair-climbing motions.

2. **Improvements to the learned structure from motion pipeline**
   Recent efforts have been made to formulate a data-driven solution to structure from motion (SfM), by using convolutional neural networks (CNNs) to parameterize a direct mapping from pixels to scene depth and camera motion. When training this *learned SfM* system, consisting of depth and egomotion neural networks, a self-supervised loss formulation (Zhou et al., 2017) has become increasingly popular, mainly because it obviates the requirement for ground truth labels and can facilitate lifelong learning. Despite their popularity, the existing egomotion networks have failed to yield the same level of accuracy as classical estimators.

   We make several contributions to the learned SfM pipeline to improve its overall accuracy, robustness, and generalizability. First, as a proof-of-concept approach to motivate the self-supervised training pipeline, we extend the method of Peretroukhin and Kelly (2018) by training a self-supervised, *deep pose correction* network that can remove systematic errors from a classical visual egomotion estimator. Second, we investigate how to address the scale ambiguity in monocular systems that has previously been shown to be detrimental to the self-supervised training procedure (Bian et al., 2019). We do so by providing a method for self-supervised scale recovery that forces the measured camera height (over the ground plane) be similar to an a priori known camera height. Our final contribution in this domain is a novel network architecture that properly *couples* the depth and egomotion networks; this *tightly coupled* framework facilitates a mutually consistent notion of scale between the two networks and significantly improves convergence during training as well as accuracy and generalizability at test time.

3. **A self-supervised, differentiable Kalman filter for visual-inertial egomotion estimation**
   Finally, having made substantial improvements to the learned SfM pipeline, we investigate how this system can be incorporated as a robust measurement model for a hybrid vision-aided INS. Building on the work of Li and Waslander (2020), our approach uses a differentiable Kalman filter with an IMU-based process model and a robust, neural-network-based relative-pose-measurement model. Through the data

efficiency of self-supervised learning, we show that our system significantly outperforms the original supervised system, while enabling online retraining. Further, we demonstrate the robustness of our system within visually degraded environments. Notably, we find that in cases where classical estimators consistently diverge, our estimator does not suffer from a significant reduction in accuracy.

The remainder of the thesis is organized as follows. In Chapter 2, we provide the background and context for our contributions. In Chapter 3, we introduce our zero-velocity-aided INS and demonstrate the advantages of aided navigation using data-driven zero-velocity updates. Chapter 4 provides an overview of visual egomotion estimation techniques and introduces the learned SfM system, which consists of two networks: one for predicting depth and one for predicting egomotion. The two subsequent chapters discuss our own improvements to the learned SfM pipeline: Chapter 5 describes our solution for resolving the scale ambiguity inherent in monocular systems; Chapter 6 demonstrates how proper coupling of the depth and egomotion network structures is crucial for accuracy and robustness. Finally, in Chapter 7, we describe our self-supervised framework for training a hybrid visual-inertial egomotion estimator that incorporates an IMU-based process model and a learned visual measurement model.

1.3 Associated Publications

This thesis is comprised of work from the following first-author publications:

1. Wagstaff, B., Peretroukhin, V., and Kelly, J. (2017). Improving foot-mounted inertial navigation through real-time motion classification. In *Proc. IEEE Int. Conf. Indoor Position. Indoor Navig. (IPIN)*

2. Wagstaff, B. and Kelly, J. (2018). LSTM-based zero-velocity detection for robust inertial navigation. In *Proc. IEEE Int. Conf. Indoor Position. Indoor Navig. (IPIN)*

3. Wagstaff, B., Peretroukhin, V., and Kelly, J. (2019). Robust data-driven zero-velocity detection for foot-mounted inertial navigation. *IEEE Sens. J.*, 20(2):957–967

4. Wagstaff, B., Peretroukhin, V., and Kelly, J. (2020). Self-supervised deep pose corrections for robust visual odometry. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2331–2337

5. Wagstaff, B. and Kelly, J. (2021). Self-supervised scale recovery for monocular depth and egomotion estimation. In *Proc. Conf. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, pages 2620–2627

6. Wagstaff, B., Peretroukhin, V., and Kelly, J. (2022a). On the coupling of depth and egomotion networks for self-supervised structure from motion. *IEEE Robot. Autom. Lett. (RAL)*, 7(3):6766 – 6773

7. Wagstaff, B., Wise, E., and Kelly, J. (2022b). A self-supervised, differentiable Kalman filter for uncertainty-aware visual-inertial odometry. In *Proc. IEEE 2022 Conf. Adv. Intell. Mechatron. (AIM)*

Chapter 2

Background

There are several fundamental concepts that underlie the contributions described in this thesis. We begin with some mathematical preliminaries, focussing on how rigid-body rotations (a basic concept in 3D pose estimation) can be represented. Then, we provide a brief introduction to the field of state estimation, where we discuss the extended Kalman filter. Following this, we outline our mathematical models for inertial and visual sensors. Finally, we discuss the essentials of deep learning.

2.1 Preliminaries

Here, we introduce some of the key concepts that are useful for describing the *pose* (consisting of the position and orientation) of a rigid body in space. Where applicable, our notation is largely consistent with that of Barfoot (2017).

2.1.1 Coordinate Frames

To describe the position of an object in 3D (Euclidean) space, we must introduce the concept of a *reference frame*. Any point $\mathbf{p} = [x, y, z]^\top$ can be expressed within a coordinate, or reference frame $\underrightarrow{\mathcal{F}}$, which consists of an orthogonal triad of vectors. In Figure 2.1, the vectors from the origin of two coordinate frames, $\underrightarrow{\mathcal{F}}_i$ and $\underrightarrow{\mathcal{F}}_v$, to a 3D point $\mathbf{p}$ are $\underrightarrow{\mathbf{r}}^{pi}$ and $\underrightarrow{\mathbf{r}}^{pv}$, respectively. These vectors, without being expressed in any particular reference frame, are related through

$$\underrightarrow{\mathbf{r}}^{pi} = \underrightarrow{\mathbf{r}}^{vi} + \underrightarrow{\mathbf{r}}^{pv}. \tag{2.1}$$

Each vector can be expressed in any reference frame. Choosing to express the vectors in $\underrightarrow{\mathcal{F}}_i$ gives

$$\mathbf{r}_i^{pi} = \mathbf{r}_i^{vi} + \mathbf{r}_i^{pv}, \tag{2.2}$$

$$= \mathbf{r}_i^{vi} + \mathbf{C}_{iv}\mathbf{r}_v^{pv}. \tag{2.3}$$

In this expression, $\mathbf{C}_{iv}$ is a 3×3 matrix that rotates coordinates from $\underrightarrow{\mathcal{F}}_v$ to $\underrightarrow{\mathcal{F}}_i$ (without accounting for the translation between the origins of the two reference frames).

Our goal will be to estimate the pose of a rigid body with respect to a fixed, world navigation frame using inertial and/or visual sensing. The measurements produced by each sensor are expressed within the sensor's

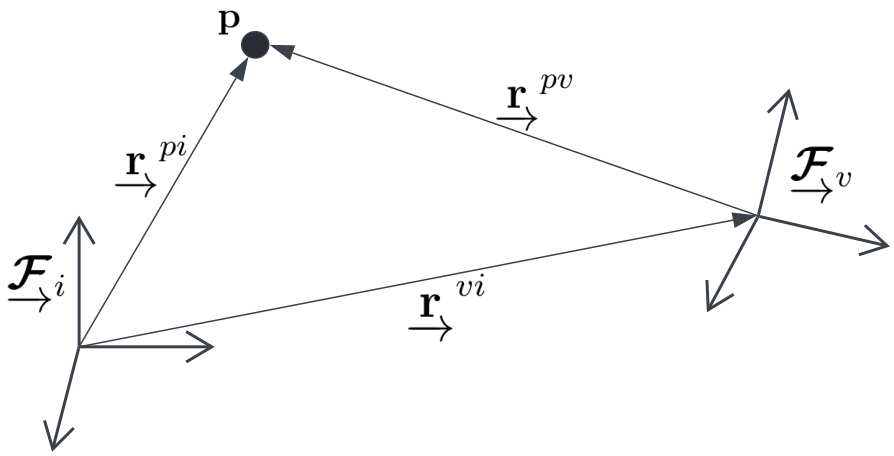


Figure 2.1: Visualization of the world and vehicle reference frames, along with a 3D point $\mathbf{p}$ whose location can be expressed with respect to either frame.

own reference frame, which is attached to (and moves with) the sensor. We will use three reference frames:

- the **world reference frame** $\underrightarrow{\mathcal{F}}_i$, which is treated as an inertial reference frame (i.e., non-accelerating and non-rotating) whose vertical (z) axis is aligned with the direction of gravity;[1]
- the **vehicle reference frame** $\underrightarrow{\mathcal{F}}_v$, which we assume (for convenience) is aligned with the IMU reference frame, and therefore is the frame in which the specific force and angular velocity measurements are expressed;
- the **camera reference frame** $\underrightarrow{\mathcal{F}}_c$, which is located at the optical centre of the camera, with its z-axis aligned with the camera's optical axis.

2.1.2 Rotation Parameterizations

The 3×3 rotation matrices used to describe the relative orientation of one reference frame with respect to another are members of the matrix Lie group $\mathrm{SO}(3)$, or the *special orthogonal* group in three dimensions. The four group properties of $\mathrm{SO}(3)$ are:

$$\text{Closure:} \quad \mathbf{C}_1\mathbf{C}_2 \in \mathrm{SO}(3) \mid \forall\, \mathbf{C}_1, \mathbf{C}_2 \in \mathrm{SO}(3), \tag{2.4a}$$

$$\text{Associativity:} \quad \mathbf{C}_1(\mathbf{C}_2\mathbf{C}_3) = (\mathbf{C}_1\mathbf{C}_2)\mathbf{C}_3 \mid \forall\, \mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3 \in \mathrm{SO}(3), \tag{2.4b}$$

$$\text{Identity:} \quad \mathbf{C}\mathbf{I}_3 = \mathbf{I}_3\mathbf{C} = \mathbf{C} \mid \forall\, \mathbf{C} \in \mathrm{SO}(3), \tag{2.4c}$$

$$\text{Invertibility:} \quad \mathbf{C}^{-1} \in \mathrm{SO}(3) \mid \forall\, \mathbf{C} \in \mathrm{SO}(3). \tag{2.4d}$$

As a Lie group, $\mathrm{SO}(3)$ is also a differentiable manifold. Additionally, elements of $\mathrm{SO}(3)$ are *noncommutative* ($\mathbf{C}_1\mathbf{C}_2 \neq \mathbf{C}_2\mathbf{C}_1$) and *orthonormal* ($\mathbf{C}_{21} = \mathbf{C}_{12}^{-1} = \mathbf{C}_{21}^{\top}$). When applying the group action of $\mathrm{SO}(3)$ (i.e., matrix

[1] Note that in reality, owing to the rotation of the earth, this frame is non-inertial, but the approximation has a negligible effect on our systems over the time scales involved.

multiplication) to 3D (Euclidean) vectors, it is both a *length-preserving* ($\mathbf{C}^\top\mathbf{C} = \mathbf{C}\mathbf{C}^\top = \mathbf{I}_3$) and *orientation-preserving* ($\det(\mathbf{C}) = 1$) transform.[2]

The $\mathrm{SO}(3)$ group has an associated Lie algebra, a vector space consisting of the set of skew symmetric matrices $\mathfrak{so}(3) = \{\boldsymbol{\Phi} = \boldsymbol{\phi}^\wedge \in \mathbb{R}^{3\times3} \mid \boldsymbol{\phi} \in \mathbb{R}^3\}$. Here, the *skew-symmetric operator*, $(\cdot)^\wedge$ is applied, which operates on a 3D vector $\boldsymbol{\phi} \in \mathbb{R}^3$ as

$$\boldsymbol{\phi}^\wedge = \begin{bmatrix}\phi_1\\ \phi_2\\ \phi_3\end{bmatrix}^\wedge = \begin{bmatrix}0 & -\phi_3 & \phi_2\\ \phi_3 & 0 & -\phi_1\\ -\phi_2 & \phi_1 & 0\end{bmatrix}. \tag{2.5}$$

The skew-symmetric operator can be used to compute the cross product between vectors $\mathbf{a}$ and $\mathbf{b}$,

$$\mathbf{a}\times\mathbf{b} = \mathbf{a}^\wedge\mathbf{b} = -\mathbf{b}^\wedge\mathbf{a}. \tag{2.6}$$

The Lie algebra represents the tangent space of the Lie group at the identity, and a vector in the Lie algebra, $\boldsymbol{\phi}$, can be mapped to $\mathrm{SO}(3)$ through the exponential map

$$\mathbf{C} = \exp\left(\boldsymbol{\phi}^\wedge\right) = \sum_{n=0}^{\infty}\frac{1}{n!}(\boldsymbol{\phi}^\wedge)^n \approx \mathbf{I} + \boldsymbol{\phi}^\wedge. \tag{2.7}$$

The Lie algebra vector, or *rotation vector* (Barfoot, 2017), is closely associated with the axis-angle representation of rotations that parameterizes a rotation as $\boldsymbol{\phi} = \phi\,\mathbf{a}$. This vector encodes a rotation by the angle ϕ about the unit-norm axis $\mathbf{a}$. Rodrigues' rotation formula defines the relationship between a rotation matrix $\mathbf{C} \in \mathrm{SO}(3)$ and the axis-angle representation,

$$\mathbf{C} = \cos\phi\mathbf{I} + (1-\cos\phi)\mathbf{a}\mathbf{a}^\top + \sin\phi\mathbf{a}^\wedge, \tag{2.8}$$

$$= \mathbf{I} + \sin\phi\mathbf{a}^\wedge + (1-\cos\phi)\mathbf{a}^\wedge\mathbf{a}^\wedge, \tag{2.9}$$

which is equivalent to the exponential map. In contrast to the rotation matrix representation, the axis-angle representation is constraint-free (whereas elements of $\mathrm{SO}(3)$ have six constraints imposed by the orthogonality condition, $\mathbf{C}\mathbf{C}^\top = \mathbf{I}$). The tradeoff, however, is that any three-parameter rotation representation gives a mapping to $\mathrm{SO}(3)$ that is *surjective* but *non-injective*. The inverse mapping (Solà et al., 2018), called the logarithmic map, is therefore only defined when restricting the domain to $0 < \phi < \pi$,

$$\boldsymbol{\phi} = \log(\mathbf{C})^\vee = \frac{\phi\,(\mathbf{C}-\mathbf{C}^\top)^\vee}{2\sin\phi}, \tag{2.10}$$

where

$$\phi = \cos^{-1}\left(\frac{\mathrm{tr}(\mathbf{C})-1}{2}\right). \tag{2.11}$$

Here, the $(\cdot)^\vee$ operator is the inverse of the $(\cdot)^\wedge$ operator. Note that when $\phi = 0$ and $\phi = \pi$, $\boldsymbol{\phi}$ is undefined in

[2] A rotation matrix with a determinant of -1 is part of the orthogonal group, but is an *improper* rotation: it rotates a vector about an axis, while also reflecting it about a plane perpendicular to the axis of rotation. The set of orthogonal matrices having a positive determinant ($+1$) make up the *special* subgroup, and consist of *proper* rotations only.

Equation (2.10). When $\phi \approx 0$, a first-order approximation can be used instead to improve numerical stability:

$$\boldsymbol{\phi} \approx (\mathbf{C} - \mathbf{I})^{\vee}. \tag{2.12}$$

Poisson's kinematical equation, or the *strapdown* equation (Stevens et al., 2015), defines the time derivative of the rotation matrix,

$$\dot{\mathbf{C}}(t) = \mathbf{C}(t)\,\boldsymbol{\omega}(t)^{\wedge}. \tag{2.13}$$

Here, $\boldsymbol{\omega}(t)$ is the instantaneous angular velocity of the system expressed within the local (body or vehicle) frame.

The Unit Quaternion Representation

An alternative rotation representation is the *unit quaternion*. Within this section we provide a brief introduction to quaternions and define a number of quaternion operations that are relevant to this thesis. We generally follow the notation from Solà (2015) and refer to their work for additional details.

Quaternions are composed of a real number component and three imaginary components,

$$Q = a + bi + cj + dk, \tag{2.14}$$

where $\{a, b, c, d\}$ are real numbers and $\{i, j, k\}$ are imaginary (unit) numbers that have the following properties:

$$i^2 = j^2 = k^2 = ijk = -1, \quad ij = -ji = k, \quad jk = -kj = i, \quad ki = -ik = j. \tag{2.15}$$

Herein, we represent a quaternion as a vector $\mathbf{q} \in \mathbb{R}^4$ with the real component, q_w, preceding the imaginary component, $\mathbf{q}_v$:

$$\mathbf{q} = \begin{bmatrix} q_w & \mathbf{q}_v^{\top} \end{bmatrix}^{\top} = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^{\top}. \tag{2.16}$$

The quaternion product operator, $\otimes$, compounds two quaternions together to yield another quaternion

$$\mathbf{r} = \mathbf{p} \otimes \mathbf{q} = \begin{bmatrix} p_w q_w - \mathbf{p}_v^{\top}\mathbf{q}_v \\ p_w \mathbf{q}_v + q_w \mathbf{p}_v + \mathbf{p}_v \times \mathbf{q}_v \end{bmatrix}. \tag{2.17}$$

Alternatively, the quaternion product can be evaluated through matrix multiplication using the bracket operators $[\mathbf{q}]_L$ and $[\mathbf{q}]_R$, defined as

$$[\mathbf{q}]_L = q_w \mathbf{I} + \begin{bmatrix} 0 & -\mathbf{q}_v^{\top} \\ \mathbf{q}_v & \mathbf{q}_v^{\wedge} \end{bmatrix}, \quad [\mathbf{q}]_R = q_w \mathbf{I} + \begin{bmatrix} 0 & -\mathbf{q}_v^{\top} \\ \mathbf{q}_v & -\mathbf{q}_v^{\wedge} \end{bmatrix}, \tag{2.18}$$

which map a quaternion to a 4×4 matrix. These operators are used for quaternion multiplication through

$$\mathbf{q}_1 \otimes \mathbf{q}_2 = [\mathbf{q}_1]_L \mathbf{q}_2 = [\mathbf{q}_2]_R \mathbf{q}_1. \tag{2.19}$$

The set of unit quaternions ($\|\mathbf{q}\| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} = 1$) forms a Lie group with elements covering the unit-sphere, S^3, embedded within $\mathbb{R}^4$ (Solà et al., 2018). Similar to $\mathrm{SO}(3)$, there is an exponential map from an

axis-angle rotation $\boldsymbol{\phi} \in \mathbb{R}^3$ to a unit quaternion. The exponential map is defined as

$$\mathbf{q} = \exp(\frac{\boldsymbol{\phi}\,\mathbf{a}}{2}) = \begin{bmatrix} \cos\frac{\phi}{2} \\ \mathbf{a}\sin\frac{\phi}{2} \end{bmatrix}, \ \mathbf{q} \in S^3. \tag{2.20}$$

There is a slight abuse of notation here, since in reality the rotation vector is represented as a pure quaternion (a quaternion with $q_w = 0$) prior to applying the exponential mapping (see Solà et al. (2018) for additional details). The unit quaternion can be used to rotate a 3D vector from $\mathbf{x} \in \mathbb{R}^3$ to $\mathbf{x}'$, through

$$\begin{bmatrix} 0 \\ \mathbf{x}' \end{bmatrix} = \mathbf{q} \otimes \begin{bmatrix} 0 \\ \mathbf{x} \end{bmatrix} \otimes \mathbf{q}^*. \tag{2.21}$$

The rotation operation requires the quaternion conjugate $\mathbf{q}^* = \begin{bmatrix} q_w & -\mathbf{q}_v^\top \end{bmatrix}^\top$. For unit quaternions, this conjugate is equivalent to the quaternion inverse, $\mathbf{q}^* = \mathbf{q}^{-1}$. Note that the rotation operation preserves the Euclidean length of the vector when it is transformed from $\mathbf{x}$ to $\mathbf{x}'$ (which requires that the real component of the output quaternion remains zero).

There is also a direct transformation from $\mathbf{q}$ to an equivalent rotation matrix, $\mathbf{C}$. Following the same notation as Solà (2015), we use $\mathbf{C}\{\mathbf{q}\}$ to represent the rotation matrix built from $\mathbf{q}$,

$$\mathbf{q} \otimes \mathbf{x} \otimes \mathbf{q}^* \longleftrightarrow \mathbf{C}\{\mathbf{q}\}\mathbf{x}, \quad \mathbf{C}\{\mathbf{q}\} = (q_w^2 - \mathbf{q}_v^\top \mathbf{q}_v)\mathbf{I}_3 + 2\mathbf{q}_v\mathbf{q}_v^\top + 2q_w\mathbf{q}_v^\wedge. \tag{2.22}$$

Note that the group of unit quaternions forms a *double cover* of $\mathrm{SO}(3)$—by observing Equation (2.22), the double cover is apparent through $\mathbf{C}\{\mathbf{q}\} = \mathbf{C}\{-\mathbf{q}\}$.

The time derivative of the unit quaternion is

$$\dot{\mathbf{q}}(t) = \frac{1}{2}\mathbf{q}(t) \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}(t) \end{bmatrix}, \tag{2.23}$$

where $\boldsymbol{\omega}(t)$ is the instantaneous angular velocity vector. This equation, analogous to Equation (2.13), is also known as Poisson's kinematical equation (in quaternion form). See Stevens et al. (2015) for its derivation.

2.1.3 Rigid-Body Transformations

A straightforward way of expressing the pose of a rigid body is in the form of a *homogeneous pose matrix* or *homogeneous transformation matrix*. A transformation matrix $\mathbf{T}_{iv}$ applies a translation and rotation to the homogeneous representation of a 3D vector, $\begin{bmatrix} \mathbf{r}^\top & 1 \end{bmatrix}^\top$, effectively implementing Equation (2.2) in matrix form,

$$\begin{bmatrix} \mathbf{r}_i^{pi} \\ 1 \end{bmatrix} = \mathbf{T}_{iv} \begin{bmatrix} \mathbf{r}_v^{pv} \\ 1 \end{bmatrix}, \quad \mathbf{T}_{iv} = \begin{bmatrix} \mathbf{C}_{iv} & \mathbf{r}_i^{vi} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}. \tag{2.24}$$

Transformation matrices are members of the special Euclidean Lie group, $\mathrm{SE}(3)$, defined as

$$\mathrm{SE}(3) = \left\{ \mathbf{T} = \begin{bmatrix} \mathbf{C} & \mathbf{r} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \mid \mathbf{C} \in \mathrm{SO}(3), \mathbf{r} \in \mathbb{R}^3 \right\}. \tag{2.25}$$

The inverse can be determined by

$$\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{C}^\top & -\mathbf{C}^\top \mathbf{r} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}. \tag{2.26}$$

The Lie algebra of $\mathrm{SE}(3)$ is $\mathfrak{se}(3) = \left\{ \boldsymbol{\Xi} = \boldsymbol{\xi}^\wedge \in \mathbb{R}^{4\times 4} \mid \boldsymbol{\xi} \in \mathbb{R}^6 \right\}$. Note that, in this context, $(\cdot)^\wedge$ is an overloaded version of the skew-symmetric operator,

$$\boldsymbol{\xi}^\wedge = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\phi} \end{bmatrix}^\wedge = \begin{bmatrix} \boldsymbol{\phi}^\wedge & \boldsymbol{\rho} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}, \quad \boldsymbol{\phi}, \boldsymbol{\rho} \in \mathbb{R}^3. \tag{2.27}$$

The exponential mapping from $\boldsymbol{\xi}$ to an element in $\mathrm{SE}(3)$ is defined as

$$\exp\left(\boldsymbol{\xi}^\wedge\right) = \begin{bmatrix} \sum_{n=0}^{\infty} \frac{1}{n!} (\boldsymbol{\phi}^\wedge)^n & \mathbf{J}_\ell \boldsymbol{\rho} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}, \tag{2.28}$$

$$\approx \mathbf{I}_4 + \boldsymbol{\xi}^\wedge. \tag{2.29}$$

Here, the $\mathbf{J}_\ell$ term,

$$\mathbf{J}_\ell = \sum_{n=0}^{\infty} \frac{1}{(n+1)!} (\boldsymbol{\phi}^\wedge)^n, \tag{2.30}$$

is the left Jacobian of $\mathrm{SO}(3)$ with its inverse

$$\mathbf{J}_\ell^{-1} = \frac{\phi}{2} \cot \frac{\phi}{2} \mathbf{I} + \left(1 - \frac{\phi}{2} \cot \frac{\phi}{2}\right) \mathbf{a}\mathbf{a}^\top - \frac{\phi}{2} \mathbf{a}^\wedge. \tag{2.31}$$

The inverse has singularities at $\phi = 2\pi m$, with m being a positive integer.

Transformation matrices are well suited for propagating the pose of a rigid body through time using newly acquired egomotion measurements. Starting from the previous body pose $\mathbf{T}_{iv_{k-1}}$ at the discrete timestep $k-1$, the egomotion estimate between the previous timestep and the current timestep, $\mathbf{T}_{v_k v_{k-1}}$, can be compounded to produce an updated pose at timestep k with

$$\mathbf{T}_{iv_k} = \mathbf{T}_{iv_{k-1}} \mathbf{T}_{v_k v_{k-1}}^{-1}, \tag{2.32}$$

$$= \mathbf{T}_{iv_{k-1}} \mathbf{T}_{v_{k-1} v_k}. \tag{2.33}$$

2.2 Estimation Theory

To obtain an optimal estimate of the state of a dynamic system, we utilize estimation theory (Maybeck, 1982) as our framework. In our case, the state of interest is the current pose of a rigid body (among other quantities), which we model as a vector random variable, $\mathbf{x}$, with a joint probability density function (PDF) $p(\mathbf{x})$. In recursive estimation, Bayesian inference is applied to update the state estimate as new measurements become available over time. At the heart of Bayesian inference is Bayes' rule, which produces an a posteriori estimate, or *belief*,

$p(\mathbf{x} \mid \mathbf{y})$, by incorporating measurements drawn from the distribution $\mathbf{y}$ with an associated PDF, $p(\mathbf{y})$:

$$p(\mathbf{x} \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}. \tag{2.34}$$

Note that we require a *measurement model*, $p(\mathbf{y} \mid \mathbf{x})$, that defines the likelihood of the measurement $\mathbf{y}$ given the state estimate $\mathbf{x}$. In the next section, we formally define our system and derive the *Bayes filter*, which uses Bayes' rule to recursively update the state belief when new information (i.e., sensor measurements) arrives. We then introduce the Kalman filter, which is a variant of the Bayes filter that relies on simplifying assumptions to produce a closed-form solution for the state posterior distribution.

2.2.1 The Bayes Filter

Formally, our goal is to determine the distribution (belief) over the state $\mathbf{x}_k$ at time t_k, using all previously available information (the initial a priori distribution $\check{\mathbf{x}}_0$, all known inputs to the system, $\mathbf{u}_{1:k}$, and all measurements, $\mathbf{y}_{0:k}$, from timesteps 0 to k):

$$p(\mathbf{x}_k \mid \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{y}_{0:k}). \tag{2.35}$$

As new sensor measurements are acquired, the Bayes filter recursively updates this posterior estimate through a two-step process. First, in the *prediction* step, the state belief is propagated forward in time via a *process model*, based on the most recent input to the system. Then, in the *correction* (or update) step, a measurement is received and the state is updated to produce a new posterior estimate. Starting from Equation (2.35), we derive this recursive form, which relies on a number of assumptions that are detailed next.

Under the assumption that all of the measurements are statistically independent, Equation (2.35) can be expressed as

$$p(\mathbf{x}_k \mid \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{y}_{0:k}) = \eta\, p(\mathbf{y}_k \mid \mathbf{x}_k)\, p(\mathbf{x}_k \mid \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{y}_{0:k-1}), \tag{2.36}$$

where η is a normalization factor that preserves the axiom of total probability, $\int p(\mathbf{x}_k) d\mathbf{x}_k = 1$. Then, by introducing a hidden state $\mathbf{x}_{k-1}$, the final term in Equation (2.36) can be defined as

$$p(\mathbf{x}_k \mid \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{y}_{0:k-1}) = \int p(\mathbf{x}_k, \mathbf{x}_{k-1} \mid \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{y}_{0:k-1}) d\mathbf{x}_{k-1}. \tag{2.37}$$

Through the chain rule of probability, and under the Markov assumption[3], the following relations hold:

$$\begin{aligned}
p(\mathbf{x}_k, \mathbf{x}_{k-1} \mid \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{y}_{0:k-1}) &= p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{y}_{0:k-1}) p(\mathbf{x}_{k-1} \mid \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{y}_{0:k-1}), \\
p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{y}_{0:k-1}) &= p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k,), \\
p(\mathbf{x}_{k-1} \mid \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{y}_{0:k-1}) &= p(\mathbf{x}_{k-1} \mid \check{\mathbf{x}}_0, \mathbf{u}_{1:k-1}, \mathbf{y}_{0:k-1}).
\end{aligned} \tag{2.38}$$

Finally, substituting the terms from Equation (2.38) into Equation (2.37), and in turn substituting Equation (2.37)

[3]The Markov assumption is that the *Markov property* holds for the system in question. The Markov property for a stochastic process dictates that future states are conditionally independent from past states, when they are conditioned on the current state, e.g., $p(x_3 \mid x_{0:2}) = p(x_3 \mid x_2)$.

into Equation (2.36), the Bayes filter is

$$p(\mathbf{x}_k \mid \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{y}_{0:k}) = \eta \underbrace{p(\mathbf{y}_k \mid \mathbf{x}_k)}_{\text{measurement model}} \int \underbrace{p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k)}_{\text{process model}} \underbrace{p(\mathbf{x}_{k-1} \mid \check{\mathbf{x}}_0, \mathbf{u}_{1:k-1}, \mathbf{y}_{0:k-1})}_{\text{prior belief}} d\mathbf{x}_{k-1}. \tag{2.39}$$

In general, propagating an arbitrary PDF through the filter is intractable due to the (typically) nonlinear process and measurement models. Therefore, simplifications (approximations) must be made; the (extended) Kalman filter (EKF) is one such simplified version of the Bayes filter.

2.2.2 The (Extended) Kalman Filter

The linear Kalman filter is a variant of the Bayes filter that assumes all distributions are Gaussian, that is, the state belief and all noise introduced to the system are assumed to be normally distributed. This simplification allows for a closed-form expression to be found in Equation (2.39). When dealing with a system that is linear and Gaussian, the Kalman filter produces an optimal estimate with a covariance that models the true uncertainty of the system (Barfoot, 2017). In practice, however, nonlinearities are present when modelling most real-world systems. The extended form of the Kalman filter handles these nonlinearities through linearization: a first-order approximation is used to propagate the Gaussian PDF through each nonlinear function. Below, we show the linearization procedure for the process and measurement models, and demonstrate how this leads to the canonical EKF equations. For further details, we refer the reader to Barfoot (2017).

In line with the Bayes' filter, the EKF estimates the PDF of a (multivariate) state vector $\mathbf{x}_k \in \mathbb{R}^n$ at timesteps $k \in 1, 2, \cdots, K$. The state belief is represented by a Gaussian PDF $p(\mathbf{x}_k) \sim \mathcal{N}\left(\hat{\mathbf{x}}_k, \hat{\mathbf{P}}_k\right)$, where $\hat{\mathbf{x}}_k$ represents the mean of the distribution and $\hat{\mathbf{P}}_k \in \mathbb{R}^{n \times n}$ is the state covariance matrix. The EKF propagates/updates the state through prediction and correction (or measurement update) steps. In the prediction step, a nonlinear process model is used to propagate the state, which incorporates process noise $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k)$:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{w}_k). \tag{2.40}$$

The first-order Taylor series approximation (about the current posterior mean $\hat{\mathbf{x}}_{k-1}$) is used for covariance propagation through the nonlinear function

$$f(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{w}_k) \approx \check{\mathbf{x}}_k + \mathbf{\Phi}_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}) + \mathbf{w}'_k, \tag{2.41}$$

where $\check{\mathbf{x}}_k = f(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, \mathbf{0})$, $\mathbf{\Phi}_{k-1} = \frac{\partial f(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{w}_k)}{\partial \mathbf{x}_{k-1}}\Big|_{\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, \mathbf{0}}$, and $\mathbf{w}'_k = \frac{\partial f(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{w}_k)}{\partial \mathbf{w}_k}\Big|_{\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, \mathbf{0}} \mathbf{w}_k$. The resulting PDF, when passed through f, is

$$p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \approx \mathcal{N}\left(\check{\mathbf{x}}_k + \mathbf{\Phi}_{k-1}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1}), \mathbf{Q}'_k\right), \tag{2.42}$$

where $\mathbf{Q}'_k = \mathbb{E}\left[\mathbf{w}'_k \mathbf{w}'^{\top}_k\right]$. The same linearization process can be applied to the nonlinear measurement model,

$$h(\mathbf{x}_k, \mathbf{n}_k) \approx \check{\mathbf{y}}_k + \mathbf{H}_k(\mathbf{x}_k - \check{\mathbf{x}}_k) + \mathbf{n}'_k, \tag{2.43}$$

which incorporates measurement noise $\mathbf{n}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$. Here, $\check{\mathbf{y}}_k = h(\check{\mathbf{x}}_k, \mathbf{0})$, $\mathbf{H}_k = \frac{\partial h(\mathbf{x}_k, \mathbf{n}_k)}{\partial \mathbf{x}_k}\Big|_{\check{\mathbf{x}}_k, \mathbf{0}}$, and $\mathbf{n}'_k = \frac{\partial h(\mathbf{x}_k, \mathbf{n}_k)}{\partial \mathbf{n}_k}\Big|_{\check{\mathbf{x}}_k, \mathbf{0}} \mathbf{n}_k$. Using the Equation (2.43) linearized measurement model, the measurement likelihood

is

$$p(\mathbf{y}_k \,|\, \mathbf{x}_k) \approx \mathcal{N}\left(\check{\mathbf{y}}_k + \mathbf{H}_k(\mathbf{x}_k - \check{\mathbf{x}}_k), \mathbf{R}'_k\right), \tag{2.44}$$

where $\mathbf{R}'_k = \mathbb{E}\left[\mathbf{n}'_k \mathbf{n}'^{\top}_k\right]$. Finally, the analytical solution for the new posterior, found by solving Equation (2.39) after substituting Equation (2.42) and Equation (2.44), is[4]

$$p(\mathbf{x}_k \,|\, \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{y}_{0:k}) = \mathcal{N}\left(\underbrace{\check{\mathbf{x}}_k + \mathbf{K}_k(\mathbf{y}_k - \check{\mathbf{y}}_k)}_{\hat{\mathbf{x}}_k}, \underbrace{(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)(\boldsymbol{\Phi}_{k-1}\hat{\mathbf{P}}_{k-1}\boldsymbol{\Phi}^{\top}_{k-1} + \mathbf{Q}'_k)}_{\hat{\mathbf{P}}_k}\right). \tag{2.45}$$

Here, $\mathbf{K}_k$ is known as the Kalman gain. The canonical EKF formulation for producing the Equation (2.45) posterior is

$$\begin{aligned}
\check{\mathbf{P}}_k &= \boldsymbol{\Phi}_{k-1}\hat{\mathbf{P}}_{k-1}\boldsymbol{\Phi}^{\top}_{k-1} + \mathbf{Q}'_k, \\
\check{\mathbf{x}}_k &= f(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, \mathbf{0}), \\
\mathbf{K}_k &= \check{\mathbf{P}}_k\mathbf{H}^{\top}_k\left(\mathbf{H}_k\check{\mathbf{P}}_k\mathbf{H}^{\top}_k + \mathbf{R}'_k\right)^{-1}, \\
\hat{\mathbf{P}}_k &= \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\check{\mathbf{P}}_k, \\
\hat{\mathbf{x}}_k &= \check{\mathbf{x}}_k + \mathbf{K}_k\left(\mathbf{y}_k - h(\check{\mathbf{x}}_k, \mathbf{0})\right).
\end{aligned} \tag{2.46}$$

One important source of error in the EKF manifests when the operating point used in the linearization of the measurement model (to produce $\mathbf{H}_k$ and $\mathbf{R}'_k$) is significantly different from the true state. Since the EKF uses the predicted state as an operating point for linearization, any error present within the previous posterior, or within the nonlinear motion model, can lead to linearization error. One way to help mitigate this issue is to repeat the linearization and recompute the final three equations of Equation (2.46), while using the *new* posterior estimate as the operating point. This process can be repeated until convergence (i.e., when further relinearization results in insignificant changes to the posterior). This extension to the EKF is called the *iterated* EKF (IEKF).

2.2.3 Representing Rotation Uncertainty

To represent uncertainty in our system, perturbations must be applied to each quantity within the state. Our state contains unconstrained vector quantities (such as translation and velocity) and constrained quantities (namely, rotation matrices or unit quaternions to represent orientation). Although it is straightforward to perturb an unconstrained quantity, it is more challenging to apply a perturbation to a constrained quantity such as a rotation matrix or unit quaternion while ensuring that all constraints are still satisfied. Perturbing unit quaternions or rotation matrices through addition, for example, yields quantities that are no longer valid elements of the S^3 and $\mathrm{SO}(3)$ groups, respectively. To address this problem for rotation matrices, we represent a random variable in $\mathrm{SO}(3)$ as

$$\mathbf{C} = \bar{\mathbf{C}}\exp\left(\delta\boldsymbol{\phi}^{\wedge}\right). \tag{2.47}$$

Here, $\mathbf{C}$ consists of a nominal (error free) component $\bar{\mathbf{C}}$, perturbed by a noise component $\delta\boldsymbol{\phi} \in \mathbb{R}^3$. If $\delta\boldsymbol{\phi}$ is a random variable sampled from a PDF (e.g., zero-mean, normally distributed through $\delta\boldsymbol{\phi} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$), it can be

[4] We forego the full derivation for the sake of brevity.

injected onto $\mathrm{SO}(3)$ through Equation (2.47) to produce the random variable $\mathbf{C}$ (i.e., Equation (2.47) induces a valid PDF $p(\mathbf{C})$ on $\mathrm{SO}(3)$ with a mean of $\bar{\mathbf{C}}$ and an associated covariance of $\mathbf{\Sigma}$).

Orientation uncertainty is similarly represented for unit quaternions. A perturbation vector with the same properties (i.e., $\delta\boldsymbol{\phi} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$) can be represented in S^3 using Equation (2.20) (i.e., the quaternion exponential map),

$$\delta\mathbf{q} = \exp\left(\frac{\delta\boldsymbol{\phi}}{2}\right) \approx \begin{bmatrix} 1 \\ \frac{\delta\boldsymbol{\phi}}{2} \end{bmatrix}, \tag{2.48}$$

and compounded with a nominal orientation, $\bar{\mathbf{q}}$, to produce a PDF over the quaternion manifold,

$$\mathbf{q} = \bar{\mathbf{q}} \otimes \delta\mathbf{q}. \tag{2.49}$$

The final expression for $\delta\mathbf{q}$ in Equation (2.48) is produced by applying the small angle approximations $\cos(\phi) \approx 1$ and $\sin(\phi) \approx \phi$ within the exponential map. Lastly, we note that these perturbations (for unit quaternions and rotation matrices) are both applied on the right side of the nominal component. Perturbations can alternatively be applied on the left or in the 'middle' (see Barfoot (2017) for more).

2.3 Mathematical Models of Visual and Inertial Sensors

In this section, we present our mathematical models for IMUs and camera sensors. These models describe how latent information (i.e., egomotion for the IMU or scene structure for the camera) is encoded within—and can be extracted from—the raw sensor measurements.

2.3.1 Inertial Measurements

An IMU produces angular velocity and specific force measurements, $\mathbf{u}_\tau = \begin{bmatrix} \boldsymbol{\omega}_{m,\tau}^\top & \mathbf{a}_{m,\tau}^\top \end{bmatrix}^\top$. The angular velocity measurement, $\boldsymbol{\omega}_{m,\tau} \in \mathbb{R}^3$, is modelled as

$$\boldsymbol{\omega}_{m,\tau} = \boldsymbol{\omega}_{v_\tau}^{v_\tau i} + \mathbf{b}_{\omega_\tau} + \mathbf{w}_{\omega,\tau}, \tag{2.50}$$

where the true angular velocity, $\boldsymbol{\omega}_{v_\tau}^{v_\tau i}$, is expressed in the vehicle frame $\underrightarrow{\mathcal{F}}_v$ at the discrete timestep τ. Additionally, there are two noise terms. First, there is a time-varying bias term, $\mathbf{b}_{\omega_\tau}$, the dynamics of which are modelled as a random walk process,

$$\dot{\mathbf{b}}_{\omega,\tau} = \mathbf{w}_{b_\omega,\tau}, \quad \mathbf{w}_{b_\omega,\tau} \sim \mathcal{N}(\mathbf{0}, \sigma_{b_\omega}\mathbf{I}_3). \tag{2.51}$$

Second, there is an additive white noise term, $\mathbf{w}_{\omega,\tau} \sim \mathcal{N}(\mathbf{0}, \sigma_\omega\mathbf{I}_3)$. The required quantity for dead reckoning is $\boldsymbol{\omega}_{v_\tau}^{v_\tau i}$, which is isolated from the other terms before being integrated to provide orientation updates through

$$\boldsymbol{\omega}_{v_\tau}^{v_\tau i} = \boldsymbol{\omega}_{m,\tau} - \mathbf{b}_{a_\tau} - \mathbf{w}_{\omega,\tau}. \tag{2.52}$$

Accelerometers measure specific force, which is the non-gravitational force per unit mass. The specific force, having units of $\frac{m}{s^2}$, is actually a measure of acceleration—specifically, the *proper* acceleration of the IMU, or the acceleration relative to free fall. When the IMU is stationary (with respect to Earth's surface), it will report a reading with a magnitude equal to that of the Earth's gravity, in the direction of the local vertical but away from

the centre of the Earth. The measurement is zero in free fall. The specific force measurement, $\mathbf{a}_{m,\tau} \in \mathbb{R}^3$, is modelled as

$$\mathbf{a}_{m,\tau} = \mathbf{a}_{v_\tau}^{v_\tau i} + \mathbf{C}_{v_\tau i}\, \mathbf{g}_i + \mathbf{b}_{a_\tau} + \mathbf{w}_{a,\tau}. \tag{2.53}$$

Note the presence of gravity $\mathbf{g}_i \approx \begin{bmatrix} 0 & 0 & 9.81 \end{bmatrix}^\top$, which we treat as a positive 'quantity' and whose direction is aligned with the vertical axis of $\underrightarrow{\mathcal{F}}_i$, but measured in the current vehicle frame, $\underrightarrow{\mathcal{F}}_{v_\tau}$, as $\mathbf{C}_{v_\tau i}\, \mathbf{g}_i$. Similar to the gyroscope measurement, there is a time-varying bias term, $\mathbf{b}_{a_\tau}$, and a white noise term, $\mathbf{w}_{a,\tau} \sim \mathcal{N}(\mathbf{0}, \sigma_a \mathbf{I}_3)$. The bias dynamics are modelled by a random walk process,

$$\dot{\mathbf{b}}_{a,\tau} = \mathbf{w}_{b_a,\tau}, \quad \mathbf{w}_{b_a,\tau} \sim \mathcal{N}(\mathbf{0}, \sigma_{b_a} \mathbf{I}_3)\,. \tag{2.54}$$

The required quantity for dead reckoning is the linear acceleration, $\mathbf{a}_i^{v_\tau i}$, which is integrated to provide velocity and position updates. This quantity must be isolated from the other terms (namely, the measured gravity and the sensor bias and noise terms):

$$\begin{aligned} \mathbf{a}_i^{v_\tau i} &= \mathbf{C}_{i v_\tau}\, \mathbf{a}_{v_\tau}^{v_\tau i}, \\ &= \mathbf{C}_{i v_\tau} (\mathbf{a}_{m,\tau} - \mathbf{C}_{v_\tau i} \mathbf{g}_i - \mathbf{b}_{a_\tau} - \mathbf{w}_{a,\tau}), \\ &= \mathbf{C}_{i v_\tau} (\mathbf{a}_{m,\tau} - \mathbf{b}_{a_\tau} - \mathbf{w}_{a,\tau}) - \mathbf{g}_i. \end{aligned} \tag{2.55}$$

Next, we introduce the discrete-time integration methods used to propagate orientation, velocity, and position estimates using IMU measurements. Solà (2015) provides additional details.

Angular Rate Integration

For inertial navigation, orientation updates are determined by integrating the angular rate outputs from the IMU. We derive a zeroth-order integration model, which is used to produce orientation updates from discrete-time angular velocity measurements. This derivation begins with the Taylor series expansion of a rotation matrix $\mathbf{C}_{\tau+1}$ at time $t_{\tau+1}$, evaluated by making use of the current orientation $\mathbf{C}_\tau$,[5]

$$\mathbf{C}_{\tau+1} = \mathbf{C}_\tau + \dot{\mathbf{C}}_\tau \delta t + \frac{1}{2!} \ddot{\mathbf{C}}_\tau \delta t^2 + \frac{1}{3!} \dddot{\mathbf{C}}_\tau \delta t^3 + \ldots, \tag{2.56}$$

where $\delta t = t_{\tau+1} - t_\tau$ is the time interval between discrete timesteps τ and $\tau + 1$. Using Poisson's kinematical equation to express the derivative terms on the right side of Equation (2.56) in terms of the current rotation and the instantaneous angular velocity, $\boldsymbol{\omega}_\tau$, we have

$$\dot{\mathbf{C}}_\tau = \mathbf{C}_\tau \boldsymbol{\omega}_\tau^\wedge, \tag{2.57}$$

$$\ddot{\mathbf{C}}_\tau = \mathbf{C}_\tau \dot{\boldsymbol{\omega}}_\tau^\wedge + \dot{\mathbf{C}}_\tau \boldsymbol{\omega}_\tau^\wedge \tag{2.58}$$

$$= \mathbf{0} + \mathbf{C}_\tau (\boldsymbol{\omega}_\tau^\wedge)^2, \tag{2.59}$$

$$\dddot{\mathbf{C}}_\tau = \mathbf{C}_\tau (\dot{\boldsymbol{\omega}}_\tau^\wedge)^2 + \dot{\mathbf{C}}_\tau (\boldsymbol{\omega}_\tau^\wedge)^2, \tag{2.60}$$

$$= \mathbf{C}_\tau (\boldsymbol{\omega}_\tau^\wedge)^3. \tag{2.61}$$

[5] Here, we simplify the rotation matrix representing the orientation of a body at time t_τ, relative to a fixed reference frame at time t_0, $\mathbf{C}_{t_\tau t_0}$, as $\mathbf{C}_\tau$.

Note that this zeroth-order approximation assumes that the angular velocity is constant throughout the interval between measurements (i.e., $\dot{\boldsymbol{\omega}} = \mathbf{0}$). Substituting these terms gives

$$\mathbf{C}_{\tau+1} = \mathbf{C}_\tau + \mathbf{C}_\tau \boldsymbol{\omega}_\tau^\wedge \delta t + \frac{1}{2!}\mathbf{C}_\tau(\boldsymbol{\omega}_\tau^\wedge)^2\delta t^2 + \frac{1}{3!}\mathbf{C}_\tau(\boldsymbol{\omega}_\tau^\wedge)^3\delta t^3 + \ldots, \tag{2.62}$$

$$= \mathbf{C}_\tau \left(\mathbf{I} + \boldsymbol{\omega}_\tau^\wedge \delta t + \frac{1}{2!}(\boldsymbol{\omega}_\tau^\wedge)^2\delta t^2 + \frac{1}{3!}(\boldsymbol{\omega}_\tau^\wedge)^3\delta t^3 + \ldots \right). \tag{2.63}$$

The expression within the brackets is equivalent to the exponential map defined in Equation (2.7), which we incorporate to yield the following expression for discrete-time orientation integration,

$$\mathbf{C}_{\tau+1} = \mathbf{C}_\tau \exp(\delta t \boldsymbol{\omega}_\tau^\wedge), \tag{2.64}$$

$$= \mathbf{C}_\tau \exp\left(\delta t(\boldsymbol{\omega}_{m,\tau} - \mathbf{b}_{a_\tau} - \mathbf{w}_{\omega,\tau})^\wedge\right). \tag{2.65}$$

The second line comes from substituting Equation (2.52) into the angular velocity term. A similar derivation for quaternions exists,

$$\mathbf{q}_{\tau+1} = \mathbf{q}_\tau \otimes \exp(\frac{\boldsymbol{\omega}_\tau \delta t}{2}), \tag{2.66}$$

$$= \mathbf{q}_\tau \otimes \begin{bmatrix} \cos(\frac{\|\boldsymbol{\omega}_\tau\|\delta t}{2}) \\ \frac{\boldsymbol{\omega}_\tau}{\|\boldsymbol{\omega}_t\|}\sin(\frac{\|\boldsymbol{\omega}_\tau\|\delta t}{2}) \end{bmatrix}, \tag{2.67}$$

which is the result of expanding $\mathbf{q}(t_\tau + \delta t)$ and applying Equation (2.23), the quaternion form of Poisson's kinematical equation.

Linear Acceleration Integration

The relationship between the continuous-time linear acceleration, $\mathbf{a}(t)$, and the rate of change in velocity, $\mathbf{v}$, and position, $\mathbf{r}$, is

$$\dot{\mathbf{r}}(t) = \mathbf{v}(t), \tag{2.68}$$

$$\dot{\mathbf{v}}(t) = \mathbf{a}(t). \tag{2.69}$$

These differential equations can be integrated over a discrete time interval to produce the discrete-time kinematics model used in dead reckoning. Beginning with the velocity term, continuous-time integration of the linear acceleration can determine the velocity at time $t + \delta t$, using the known velocity at time t, as

$$\mathbf{v}(t + \delta t) = \mathbf{v}(t) + \int_t^{t+\delta t} \mathbf{a}(s)ds. \tag{2.70}$$

In practice, this integral cannot be solved in closed form since the linear acceleration measurements are only available at discrete timesteps. We use the first-order Runge-Kutta method (also known as Euler integration) to approximate the solution to the integral by assuming the linear acceleration is constant across the interval δt:

$$\mathbf{v}(t + \delta t) = \mathbf{v}(t) + \mathbf{a}(t)\delta t. \tag{2.71}$$

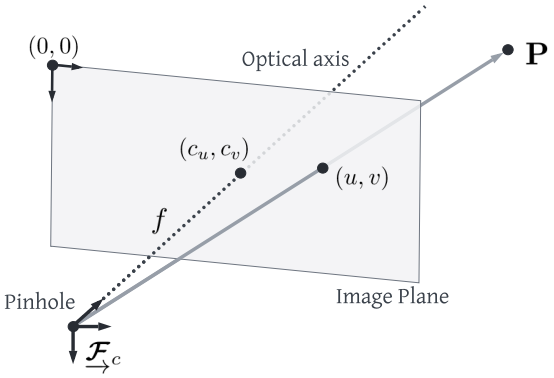


Figure 2.2: Frontal projection model for a pinhole camera with a focal length, f, and principal point (c_u, c_v).

Assuming that $t_\tau = \tau\delta t$, and that $\mathbf{v}_\tau = \mathbf{v}(t_\tau)$, with τ being a non-negative integer, we can rewrite Equation (2.71) as

$$\mathbf{v}_{\tau+1} = \mathbf{v}_\tau + \mathbf{a}_\tau \delta t. \tag{2.72}$$

Substituting Equation (2.55), the velocity update is expressed with respect to the IMU measurement, $\mathbf{a}_{m,\tau}$, as[6]

$$\mathbf{v}_{\tau+1} = \mathbf{v}_\tau + \left(\mathbf{C}_\tau \left(\mathbf{a}_{m,\tau} - \mathbf{b}_\tau - \mathbf{w}_{a,\tau}\right) - \mathbf{g}\right) \delta t. \tag{2.73}$$

We can subsequently derive the kinematic equation for position by integrating the velocity, where both the linear acceleration and the velocity are treated as constant across the interval. The resulting expression for position is

$$\mathbf{r}_{\tau+1} = \mathbf{r}_\tau + \mathbf{v}_\tau \delta t + \frac{1}{2}\left(\mathbf{C}_\tau \left(\mathbf{a}_{m,\tau} - \mathbf{b}_\tau - \mathbf{w}_{a,\tau}\right) - \mathbf{g}\right) \delta t^2. \tag{2.74}$$

2.3.2 Visual Measurements

Cameras capture a rich representation of the 3D environment in the form of 2D images. A *pinhole camera* is an idealized camera model used to relate the locations of 3D coordinates in the scene to their positions within the image. Light rays from these 3D points are projected onto the image sensor behind the pinhole, and the resulting planar representation of the scene is inverted in the process. It is easier (and equivalent) to place a virtual image plane in front of the pinhole, as shown in Figure 2.2, to avoid dealing with the inverted image. In this *frontal* projection model, the image plane is placed at a distance f from the pinhole, along the optical axis, which in this case is aligned with the z-axis of the camera reference frame, $\underrightarrow{\mathcal{F}}_c$. The optical axis intersects the image plane at the principal point (c_u, c_v) (relative to the image plane coordinate origin at the upper-left corner of the plane). A 3D point in space, $\mathbf{p} = \begin{bmatrix} x & y & z \end{bmatrix}^\top$, can be projected to the pixel coordinates $\mathbf{u} = \begin{bmatrix} u & v \end{bmatrix}^\top$

[6]Note that we remove the reference frame notation for simplicity. Here, $\mathbf{C}_k$ refers to the rotation matrix that transforms from the current IMU reference frame to the global reference frame that the velocity and gravity vector are expressed within.

through the projection model $\pi(\mathbf{p})$ defined by

$$\pi(\mathbf{p}) = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & 0 & c_u \\ 0 & f_v & c_u \\ 0 & 0 & 1 \end{bmatrix} \frac{1}{z}\mathbf{p}, \tag{2.75}$$

where $(f_u,\ f_v)$ are the camera focal lengths in the horizontal and vertical directions, respectively. Here, we observe that the projection results in a loss of depth information. This loss of information leads to the problem of *scale ambiguity* in monocular systems: a single camera can be used to estimate egomotion and scene structure, but these estimates will only be accurate up to some unknown scale factor. If a depth estimate $\tilde{z}$ (e.g., measured with any type of depth sensor) for $\mathbf{u}$ exists, the inverse projection model determines the 3D coordinates $\mathbf{p}$ from $\mathbf{u}$ as

$$\mathbf{p} = \pi^{-1}(\mathbf{u}, \tilde{z}) = \tilde{z} \begin{bmatrix} \frac{u-c_u}{f_u} & \frac{v-c_v}{f_v} & 1 \end{bmatrix}^\top. \tag{2.76}$$

Modelling Assumptions

The pinhole projection model does not account for the lens distortion effects that are present in all cameras in practice. Lens distortions introduce discrepancies between the observed coordinates of image plane points and the idealized projection that would result from applying the pinhole model. Various distortion models exist (Brown, 1966). Herein, we assume that images are undistorted prior to using them within our visual egomotion pipeline. Further, we assume the camera calibration parameters (i.e., the camera intrinsic parameters, f_u, f_v, c_u, c_v, and the static $\mathrm{SE}(3)$ transform between the camera, $\underrightarrow{\mathcal{F}}_c$, and the IMU, $\underrightarrow{\mathcal{F}}_v$, reference frames) are known. Finally, we assume the camera has a global shutter (and omit any discussion of the rolling shutter alternative).

2.4 An Overview of Deep Learning

Recent advancements in deep learning have given rise to a number of cutting-edge machine learning techniques for training data-driven parametric models (LeCun et al., 2015). We employ a number of these techniques to learn robust measurement models for aided navigation. In this section, we provide a brief introduction to deep neural networks and the methods used to train them, but we refer the reader to Goodfellow et al. (2016) for a more comprehensive introduction to deep learning.

At the heart of deep learning is the feedforward neural network, which is a parameterized model designed to approximate a target function $f^* : \mathbb{R}^M \to \mathbb{R}^N$ that maps input data to a desired output. A neural network $f_{\boldsymbol{\theta}}(\mathbf{x}_k)$ with a set of tunable parameters, $\boldsymbol{\theta}$, can be trained to approximate f^* through *supervised learning* to obtain a set of weights, $\boldsymbol{\theta}^*$, that minimizes the error between the true function f^* and the network output across a set of *training data*. Individual samples within the training set consist of an input vector $\mathbf{x}_k$, and the corresponding output from f^*, $\mathbf{y}_k$. Since the true output distribution of f^* is generally unknown (which is why we desire to approximate it with a neural network in the first place), the desired outputs, or *labels*, must be acquired through an 'expert' labelling scheme (hence the term *supervised learning*). With a dataset of labelled inputs, $\mathcal{D}_{train}$, the goal of supervised learning is to train the neural network to minimize an error, $\mathcal{L}$, between the target labels and the network output:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{\{\mathbf{x}_k, \mathbf{y}_k\} \in \mathcal{D}_{train}} \mathcal{L}\left(f_{\boldsymbol{\theta}}(\mathbf{x}_k), \mathbf{y}_k, \boldsymbol{\theta}\right). \tag{2.77}$$

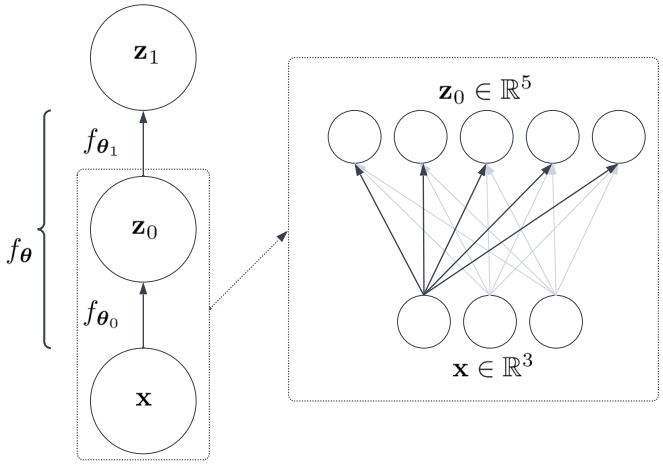


Figure 2.3: A two-layer fully connected network with input $\mathbf{x}$, intermediate (hidden) vector $\mathbf{z}_0$, and output $\mathbf{z}_1$. We illustrate the connections between a three-dimensional input and a five-dimensional intermediate output.

Network training is accomplished in practice through gradient-based optimization, which we describe in the next section.

The general form of the feedforward network is a series of functions that progressively transforms the input into the desired output. The term *feedforward* stems from this unidirectional flow of information through the network (without any form of feedback). The term *network* is used because the model consists of a number of functions that are composed together,

$$f_{\boldsymbol{\theta}}(\mathbf{x}_k) = (f_{\boldsymbol{\theta}_L} \circ f_{\boldsymbol{\theta}_{L-1}} \cdots f_{\boldsymbol{\theta}_1} \circ f_{\boldsymbol{\theta}_0})(\mathbf{x}_k), \tag{2.78}$$

where the composition operator, $\circ$, represents the passing of the output from one network layer into the next, $(f \circ g)(\mathbf{x}) = f(g(\mathbf{x}))$. The learnable parameters of each function are combined to form $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots \boldsymbol{\theta}_L, \}$. Finally, the term *neural* stems from the similarity between these parameterized models and neurons within the brain.[7] In deep networks, being *deep* refers to the use of a large number of sequentially composed layers. One of the more recent breakthroughs in machine learning was the discovery that increasing the number of network layers almost always leads to improved performance.

2.4.1 Fully Connected Networks

The standard form of feedforward network is the *fully connected* (FC) network illustrated in Figure 2.3. In a FC layer, a function $f_{\boldsymbol{\theta}_\ell}$ maps the input vector, $\mathbf{z}_{\ell-1}$ to an output vector, $\mathbf{z}_\ell$ through the following nonlinear function:

$$\mathbf{z}_\ell = f_{\boldsymbol{\theta}_\ell}(\mathbf{z}_{\ell-1}), \tag{2.79}$$

$$= \phi\left(\mathbf{W}_\ell \mathbf{z}_{\ell-1} + \mathbf{b}_\ell\right), \tag{2.80}$$

[7] Note that neural networks have been used to model/understand brain function but are not actually representative of the biological functionality of the brain.

where the tunable parameters, $\boldsymbol{\theta}_\ell = \{\mathbf{W}_\ell, \mathbf{B}_\ell\}$, consist of the linear weight matrix $\mathbf{W}_\ell \in \mathbb{R}^{L_\ell \times L_{\ell-1}}$ and a bias vector $\mathbf{b}_\ell$. Note that this type of layer is *fully connected* because the weight matrix generates output elements that are a function of *all* elements from the previous layer (see Figure 2.3 for an illustration). Following this linear transformation, a nonlinear *activation function*, ϕ, is applied (nonlinearity is essential). A common activation is the rectified linear unit (ReLU),

$$\phi(z) = \max(0, z), \tag{2.81}$$

applied element-wise to each unit, z. A number of variants exist, including the leaky ReLU (Maas et al., 2013), or the exponential linear unit (ELU) (Clevert et al., 2016),

$$\phi(z) = \begin{cases} z & z > 0, \\ \alpha(e^z - 1) & z \leq 0. \end{cases} \tag{2.82}$$

The choice of activation function is in general a network design decision and is an ongoing area of research. We refer the reader to Nwankpa et al. (2018) for additional details.

At the final network layer, the choice of activation (or lack thereof) is particularly important to allow the network to properly match the desired output. For an unconstrained regression task, no activation function is required. For a classification task, the *softmax* activation can be used to regress a normalized class probability for each output class z_i through

$$z_i = \frac{e^{z_i}}{\sum_{j=0}^{M} e^{z_j}}. \tag{2.83}$$

The element-wise sigmoid activation,

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{2.84}$$

restricts all elements of the output vector to be within the range $[0, 1]$.

2.4.2 Network Training

The usual way to train a network (i.e., fulfilling Equation (2.77)) is through gradient-based optimization. The training procedure, called *gradient descent*, performs a number of optimization steps that update the individual network weights θ_i in such a way that the overall loss, $\mathcal{L}$, is minimized. Each step updates the network parameters as

$$\theta_i \leftarrow \theta_i - \epsilon \frac{\partial \mathcal{L}}{\partial \theta_i}. \tag{2.85}$$

The step size, ϵ, known as the *learning rate*, dictates how much the weights should change with each gradient descent step. To carry out gradient descent, the derivative of the loss with respect to all learnable parameters must be computed. These gradients are determined using the chain rule of calculus to backpropagate the gradient from the loss, through the network, to the specific parameter. For example, with a two layer network $z =$

$(g_\phi \circ f_\theta)(x)$ having scalar weight parameters ϕ and θ, backpropagation with the chain rule is:

$$\frac{\partial \mathcal{L}}{\partial \phi} = \frac{\partial \mathcal{L}}{\partial g}\frac{\partial g}{\partial \phi}, \quad \frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial g}\frac{\partial g}{\partial f}\frac{\partial f}{\partial \theta}. \tag{2.86}$$

Note that, in practice, due to the vast size of modern training datasets, it is intractable to compute a gradient step with respect to the composite loss across all training samples. Rather, training is broken into *minibatches* of randomly sampled training samples. A forward pass is computed for all samples in the minibatch, and a gradient descent step is performed that reduces the loss value across this subset of data. A training epoch is complete once the full training set has been sampled; multiple training epochs are generally required before the network weights converge. This technique is called *stochastic gradient descent* (SGD). More advanced optimizers are available that build on SGD. The Adam optimizer (Kingma and Ba, 2014), for example, controls parameter-specific learning rates during training to improve convergence.

With modern deep learning frameworks such as PyTorch (Paszke et al., 2017), implementing a network-based system has become more straightforward over the last decade. This is in part due to *automatic differentiation* (or autograd), which is a built-in feature of PyTorch (and other software tools) that can automatically track the gradient flow through network layers. Autograd allows for rapid implementation of network architectures and loss functions without having to manually compute the complicated derivatives required for backpropagation. We use the PyTorch framework in Python to implement and train our deep networks.

Generalization and Overfitting

The goal of machine learning is to produce a *generalizable* model, that is, a model that achieves a high level of performance on data that exists outside of the training dataset. To properly generalize, models must be prevented from *overfitting* to the training data, which can happen when the weights are overtuned to 'memorize' the target outputs across the training dataset. Many *regularization* techniques have been designed to prevent overfitting. *Early stopping*, for example, prevents overfitting by stopping training when the accuracy on a held-out validation set begins to decrease. Other generalization techniques exist, such as increasing the amount of training data available, for example, by collecting more data (with labels) or *augmenting* the current data, can improve generalization. Augmentation involves applying transformations that change the input without impacting the label (or by applying a transformation that impacts the label in a known way). Other methods for regularization include, for example, weight decay (i.e., penalizing large weight values in an auxiliary loss), dropout (Srivastava et al., 2014), and batch normalization (Ioffe and Szegedy, 2015).

Loss Functions

Our work employs deep networks for *classification* and *regression*. Classification networks are constrained to output a class prediction from a set of possible discrete labels. For binary classification (i.e., choosing one of two labels), the *binary cross entropy* (BCE) loss is used,

$$\mathcal{L}_{BCE} = -\frac{1}{K}\sum_{k=1}^{K} y_k \log f_{\boldsymbol{\theta}}(\mathbf{x_k}) + (1 - y_k)\log\left(1 - f_{\boldsymbol{\theta}}(\mathbf{x_k})\right). \tag{2.87}$$

The BCE minimizes the error between the training labels $y_k = \{0, 1\}$ and the 1D network output, $f_{\boldsymbol{\theta}}(\mathbf{x_k})$. Note that the network's final layer is a sigmoid or softmax activation that constrains the output to be within $[0, 1]$. The output is considered to be the 'confidence' that the input corresponds to the positive class label (i.e., $p(y_k = 1)$).

The BCE loss is a specific instance of the *negative log likelihood* loss, which can be used for multi-class problems.

Regression tasks involve producing unconstrained network outputs that approximate the target values. Two common loss functions for regression tasks are the mean squared error (MSE) and the L_1 loss,

$$\mathcal{L}_{MSE} = \frac{1}{2K} \sum_{k=1}^{K} \| f_{\boldsymbol{\theta}}(\mathbf{x_k}) - \mathbf{y}_k \|_2^2 \,, \tag{2.88}$$

$$\mathcal{L}_{L_1} = \frac{1}{2K} \sum_{k=1}^{K} | f_{\boldsymbol{\theta}}(\mathbf{x_k}) - \mathbf{y}_k | \,. \tag{2.89}$$

Self-Supervised Learning

As an increasing amount of raw (unlabelled) data becomes available, there is potential to incorporate a vast amount of information to improve the overall effectiveness of deep networks. The primary limitation of supervised learning is that it requires these data to be accurately labelled in order to minimize the losses described above. Acquiring labels can be time consuming and expensive; it is often infeasible when millions of training samples may exist. An alternative and popular strategy is *self-supervised* learning. Self-supervised learning involves the automatic generation of training labels from data. Although this label generation process requires some form of domain knowledge, it is a more scalable process than supervised labelling techniques, since there is no upper limit on the amount of data that can be incorporated. Self-supervision is commonly achieved by leveraging the known structure of the data. For example, a network, given the lower half of an image, can be trained to predict (or generate) the upper half of the image. Similarly, given an image at one point in time, a network can be trained to predict temporally adjacent images. Since the targets in these cases are pixels that are already available, the labelling is done 'for free.' Alternatively, when multiple sensors are available, labels can be generated from one source of data in order to train a network that takes as its input an alternative source of data. For example, on a robotic platform, a lidar sensor can generate per-pixel depth labels, which can be used to train an image-based depth network that only requires a monocular camera to operate at test time.

2.4.3 Specialized Network Architectures

There are many alternatives to the standard fully connected network architecture that are specifically designed for certain data types. Next, we outline two important network structures: the convolutional neural network for images, and the recurrent neural network for time-series data.

Convolutional Neural Networks

Convolutional neural networks (CNNs) are designed to process 2D 'grids' of data; they are consequently well suited for image processing and computer vision tasks. Layers within the CNN apply convolutions to the input with a learned *kernel* that transforms the 2D input into a 'feature map'. By applying a series of convolutions, these feature maps are progressively transformed into lower-dimensional representations of the input. The representations can be used for a number of computer vision tasks such as image classification, object detection, or semantic segmentation. With the availability of visual sensors on robotic platforms, CNNs are being applied within the domain of robotics for numerous tasks including visual localization.

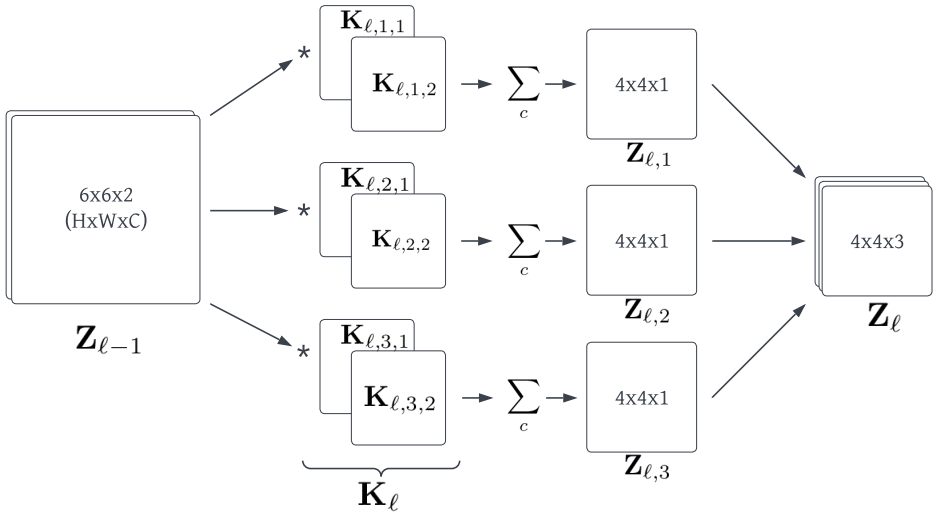


Figure 2.4: Illustrating discrete, 2D convolutions over a volume. Here, three convolutional kernels ($\mathbf{K}_{\ell,c,k} \in \mathbb{R}^{3\times3}$) are applied to the two-channel input $\mathbf{Z}_{\ell-1}$ to produce the three-channel output $\mathbf{Z}_\ell$. No padding is used, resulting in a reduction to the spatial resolution.

The convolution operator applied to a continuous function, $f(x)$, is

$$f(x) * K(x) = \int_{-\infty}^{\infty} f(s)K(x-s)ds, \tag{2.90}$$

$$= \int_{-\infty}^{\infty} f(x-s)K(s)ds, \tag{2.91}$$

where $K(x)$ is defined to be the *kernel function* operating on $f(x)$. In practice, CNN layers operate on finite and discrete 2D inputs, using the discrete-time version of the convolution operator. For a single-channel 2D input, $\mathbf{Z}_{\ell-1} \in \mathbb{R}^{H\times W\times1}$, the discrete convolution with the kernel $\mathbf{K}_\ell \in \mathbb{R}^{W\times W}$ produces[8]

$$\mathbf{Z}_\ell(u,v) = (\mathbf{Z}_{\ell-1} * \mathbf{K}_\ell)(u,v) = \sum_i \sum_j \mathbf{Z}_{\ell-1}(u+i, v+j)\mathbf{K}_\ell(i,j). \tag{2.92}$$

The weights in each kernel are the learnable parameters in the CNN layer. With only a small number of weights (kernels are often only 3×3 or 5×5 in size), convolutions significantly reduce the number of trainable parameters per layer compared with the fully connected variant as they utilize *weight sharing* by applying the same low-dimensional kernel to every element of the layer's input. A fully connected layer, in contrast, requires a specific weight between every element in the input and every element in the output. The use of CNN layers leads to a reduction in memory requirements and an improvement in efficiency during training compared with the fully connected layers when operating on 2D data.

Convolutions are typically applied to a *volume* of data, since multiple input channels can exist (a colour image, for example, has three channels of 2D data). At each layer, an arbitrary number of kernels can be applied to the input (in order to extract a richer set of features from the input), which progressively increases the number

[8]This expression is actually the *cross-correlation* function, which is the convolution operator without a flipped kernel. Cross-correlations do not possess the commutative property, i.e., $(\mathbf{Z} * \mathbf{K})(u,v) = (\mathbf{K} * \mathbf{Z})(u,v)$, that convolutions have, but this does not impact neural network design.

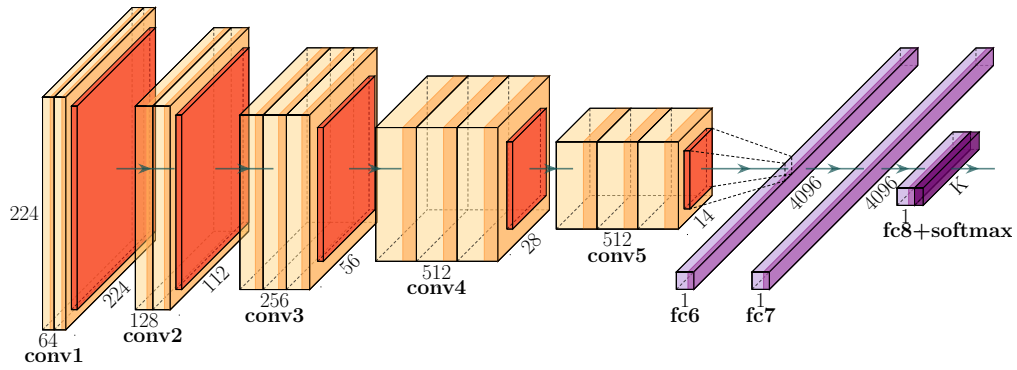


Figure 2.5: Diagram of VGG16 (Simonyan and Zisserman, 2014), one of the initial deep CNNs that pushed the state of the art for image classification. Five convolutional blocks (consisting of multiple convolution layers with ReLU activations) are applied, with max pooling (shown in red) applied after every block to reduce the spatial resolution by a factor of two. Following the convolution blocks, the output is reshaped, and then three fully connected layers (fc) are applied. Finally, a softmax activation is used for classification.

of output channels. The convolution to an input volume extends the 2D convolution from Equation (2.92) to

$$\mathbf{Z}_{\ell,k}(u,v) = \sum_{c}\sum_{i}\sum_{j}\mathbf{Z}_{\ell-1,c}(u+i,v+j)\,\mathbf{K}_{\ell,c,k}(i,j). \tag{2.93}$$

Here, in layer ℓ, kernel $\mathbf{K}_{\ell,k,c}$ is applied to the c^{th} channel of the input and combined with all of the other k^{th} kernels to produce the k^{th} output channel. This process is visualized in Figure 2.4. Here, we see that the convolution operators reduce the input dimensionality. To maintain the original spatial resolution, the border of the inputs can be padded (typically with zeros) prior to applying the convolution.

CNN Architecture Overview Figure 2.5 visualizes VGG16 (Simonyan and Zisserman, 2014), one of the original deep CNN architectures. A series of convolutions (followed by nonlinear activations and *pooling layers*) are chained together in the network. The role of the pooling layer is to reduce the spatial resolution of the output feature map by condensing the information within a window of features into a smaller output. Max pooling, for example, returns the maximum value within a square window of samples. Pooling over a 2×2 window of the feature map effectively halves the spatial resolution without removing a substantial amount of information from the feature map.[9] With each applied pooling operation, the spatial resolution of the feature map decreases, which causes the *receptive field* (the number of input units represented by each element of the current feature map) to increase. The increasing receptive field results in a hierarchical feature representation; as the network depth increases, the feature maps represent higher-level characteristics of the image. Importantly, these image-level feature representations can be produced using only *local* convolutions that operate on a small subset of the feature maps. At the end of the network, the low-dimensional feature representation can be utilized for regression or classification. In the case of VGG16, a fully connected layer is attached to the final convolutional layer output and a softmax activation is used for image classification. During training, the weights within all of the convolution kernels can be updated through gradient descent. Like the fully connected network, gradient descent uses the chain rule to backpropagate gradients from the loss to each individual kernel weight. We omit the full derivation of the backpropagation equations for convolutional layers.

[9] Alternatively, a strided convolution can be used to reduce the spatial resolution.

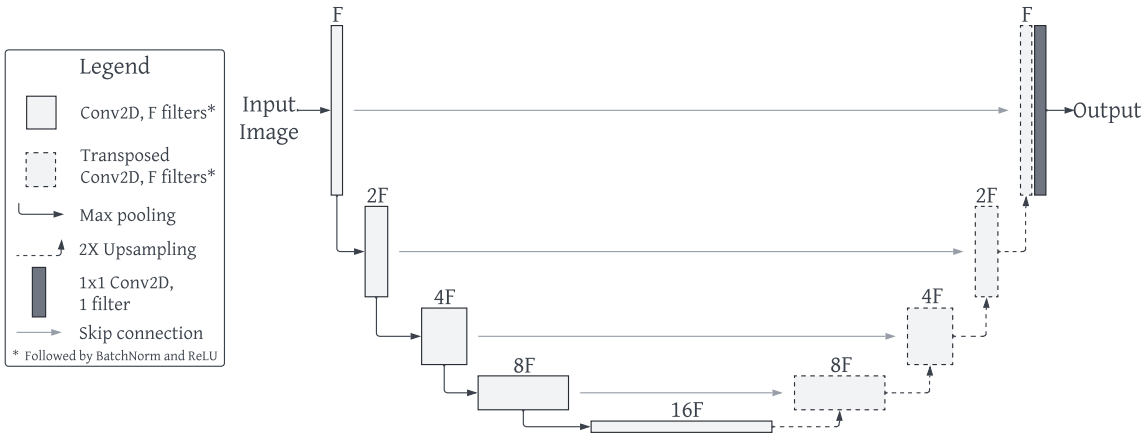


Figure 2.6: Network diagram for the U-Net. This type of network is composed of an encoder and a decoder. Feature maps from the encoder (brought forward through skip connections) are merged with feature maps in the decoder. The spatial resolution of the output is similar (or equal) to that of the input, making this type of network suitable for per-pixel classification or regression tasks. (Figure adapted from https://nchlis.github.io/).

U-Net A particularly important type of CNN architecture that we employ in our system is the U-Net (Ronneberger et al., 2015). This type of network, which takes as input a 2D image, is designed to output *dense*, or per-pixel, predictions that match the spatial resolution of the input. The architecture is particularly useful for computer vision tasks such as image segmentation or object detection. We use U-Net for visual depth estimation within our learned SfM system, as described in the latter half of the thesis. The U-Net, illustrated in Figure 2.6, is composed of an encoder and a decoder. The encoder is a conventional CNN, which produces feature maps that decrease in spatial resolution and increase in channel dimensionality towards the 'bottleneck'. Following the bottleneck is the decoder, which increases the resolution of the feature maps either through bilinear interpolation, or through *transposed convolutions* (Zeiler et al., 2010). The other unique components of the U-Net are the *skip connections* that directly connect feature maps from layers in the encoder to layers in the decoder.

Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of neural networks that leverage the temporal correlations that exist in sequential data. RNNs propagate a hidden 'memory' state through time, passing temporal information from prior states to aid the current prediction. The hidden state vector $\mathbf{h}_k$ is propagated by combining the previous hidden state $\mathbf{h}_{k-1}$ with the current input $\mathbf{x}_k$, and passing the weighted sum through a nonlinear activation function, $\phi(\cdot)$,

$$\mathbf{h}_k = \phi(\mathbf{W}^{hx}\mathbf{x}_k + \mathbf{W}^{hh}\mathbf{h}_{k-1}). \tag{2.94}$$

Here, $\mathbf{W}^{hx}$ and $\mathbf{W}^{hh}$ are matrices that contain the learnable weights. Note that the same weight matrices are applied at all timesteps in the sequence.

The backpropagation through time (BPTT) algorithm (Werbos, 1990) is used to train RNNs by passing gradient information from the current timestep to previous timesteps. BPTT enables the networks to learn how to exploit temporal relationships within the data. Although effective with shorter sequence lengths, BPTT with the basic RNN network structure is known to produce vanishing or exploding gradients when attempting to backpropagate through longer sequences. This problem limits the overall effectiveness of the vanilla RNN. To address the instability, alternative network structures have been proposed that are more effective at learning

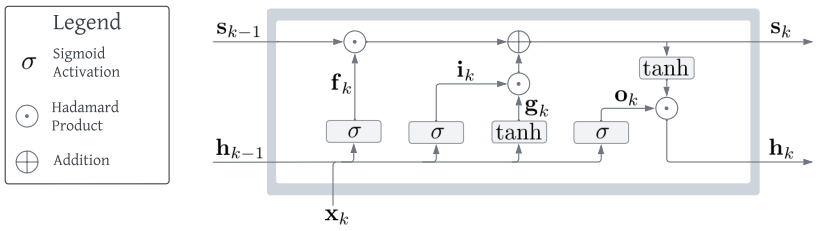


Figure 2.7: The long short-term memory (LSTM) recurrent neural network structure. The various gates with sigmoid activations control how the internal state is updated, by 'learning' and 'forgetting' information. (Graphic inspired by *Colah's Blog* https://colah.github.io/posts/2015-08-Understanding-LSTMs/).

long-term relationships. The long short-term memory (LSTM) network, in particular, is a popular alternative to the vanilla RNN. Here, we provide a brief description and refer the reader to Lipton et al. (2015) for a more detailed review of the theory of LSTM networks.

The LSTM, illustrated in Figure 2.7, avoids the exploding and vanishing gradient problems by incorporating a second internal state $\mathbf{s}^{(t)}$, which acts as a 'direct' connection between timesteps and allows for gradient values to be propagated through time without significantly changing in value. The LSTM equations for processing sequential data are

$$\begin{aligned}
\mathbf{g}_k &= \tanh(\mathbf{W}^{\mathrm{gx}}\mathbf{x}_k + \mathbf{W}^{\mathrm{gh}}\mathbf{h}_{k-1} + \mathbf{b}_{\mathrm{g}}),\\
\mathbf{i}_k &= \sigma(\mathbf{W}^{\mathrm{ix}}\mathbf{x}_k + \mathbf{W}^{\mathrm{ih}}\mathbf{h}_{k-1} + \mathbf{b}_{\mathrm{i}}),\\
\mathbf{f}_k &= \sigma(\mathbf{W}^{\mathrm{fx}}\mathbf{x}_k + \mathbf{W}^{\mathrm{fh}}\mathbf{h}_{k-1} + \mathbf{b}_{\mathrm{f}}),\\
\mathbf{o}_k &= \sigma(\mathbf{W}^{\mathrm{ox}}\mathbf{x}_k + \mathbf{W}^{\mathrm{oh}}\mathbf{h}_{k-1} + \mathbf{b}_{\mathrm{o}}),\\
\mathbf{s}_k &= \mathbf{g}_k \odot \mathbf{i}_k + \mathbf{s}_{k-1} \odot \mathbf{f}_k,\\
\mathbf{h}_k &= \tanh(\mathbf{s}_k) \odot \mathbf{o}_k.
\end{aligned} \tag{2.95}$$

The functions $\mathbf{i}_k$, $\mathbf{f}_k$, and $\mathbf{o}_k$ form the input, forget, and output gates respectively. These functions determine which elements from $\mathbf{h}_{k-1}$ and $\mathbf{x}_k$ compose the internal state $\mathbf{s}_k$ and the hidden output $\mathbf{h}_k$. A sigmoid activation function $\sigma(\cdot)$ restricts the gate outputs to be within 0 and 1, which causes data elements to be passed through the gate (if the gate value is close to 1) or rejected (if the gate value is close to 0).

Chapter 3

Learned Zero-Velocity Detection for Robust Inertial Navigation

Pedestrian navigation systems that use wearable sensors for pose estimation are a popular alternative to infrastructure-based methods such as GNSS (Hou and Bergmann, 2020). By relying solely on body-mounted sensors, these systems are standalone and can be rapidly deployed to provide localization estimates for humans in both indoor and outdoor environments. Notably, pedestrian navigation systems are important for coordinating personnel in first-response scenarios. For teams of first responders such as firefighters, the ability to track team members' locations in real time facilitates more effective response strategies and expedites personnel extraction in case of unexpected events or injuries (Fischer and Gellersen, 2010). These systems, moreover, have the potential to coordinate the movements of humans and mobile robots within a shared space.

Pedestrian navigation systems often utilize a body-mounted inertial sensor for dead reckoning, producing relative pose estimates with respect to a previous pose. Pedestrian dead reckoning (PDR) systems, by relying on a single IMU (and no other sensors) are ideal for pedestrian navigation, since IMUs are low cost, lightweight, low power, and straightforward to deploy without requiring the wearer to modify their behaviour. However, with relatively inexpensive inertial sensors that employ microelectromechanical systems (MEMS), PDR systems are only accurate for short durations, as sensor biases and noise quickly lead to estimates of progressively poorer quality. Position error, notably, grows cubically with time (Skog et al., 2010b) and some form of additional correction is necessary for accurate long-duration inertial odometry.

To combat inertial sensor drift, one attractive technique (that does not require an external aid or extant maps) is to mount the IMU on the foot of an individual and rely on zero-velocity updates. These updates are *pseudo-measurements* of the velocity state that occur during midstance, that is, the portion of the human gait during which the foot is flat on the ground and stationary relative to the navigation frame (see Figure 3.1 for an illustration of the gait cycle). By incorporating such pseudo-measurements into the INS through a Bayesian filter such as an extended Kalman filter (EKF), dead reckoning is limited to the intervals between footfalls, as opposed to over the entire tracked motion (Foxlin, 2005). As a result, PDR systems based on the zero-velocity-aided INS are capable of providing accurate position estimates and have been deployed in a number of settings (Foxlin, 2005; Godha and Lachapelle, 2008; Nilsson and Handel, 2014; Jiménez et al., 2010).

To successfully apply zero-velocity measurements, the stationary phases of the human gait must first be accurately identified. If the stationary phases are correctly identified, zero-velocity updates can significantly improve localization estimates. However, false-positive detections cause the length of the wearer's trajectory to

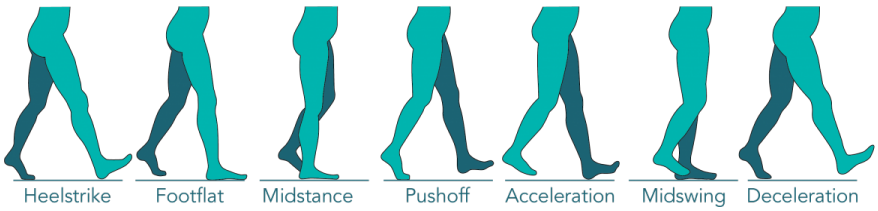


Figure 3.1: The phases of the human gait cycle. During the midstance phase, the foot is stationary with respect to the ground (and to the fixed navigation frame). A foot-mounted INS can utilize this information to apply zero-velocity measurements during the midstance phase. Image from tekscan.com/blog/medical/gait-cycle-phases-parameters-evaluate-technology

be underestimated, while false-negative detections (i.e., missed detections) lead to rapid and unbounded error growth. A range of zero-velocity detectors have been proposed to identify these stationary periods in order to appropriately apply zero-velocity pseudo-measurements. The detectors rely on statistical *likelihood-ratio tests* (LRTs) to estimate how similar the current IMU measurements are to those produced by a stationary IMU (Skog et al., 2010b). 'Classical' approaches based on the LRT, however, require careful tuning of a threshold parameter, the value of which is heavily based on the user's motion type. As the threshold parameter must be manually assigned, these classical detectors cannot be generalized to a broad range of human locomotion patterns. Therefore, achieving robust zero-velocity detection—that is, accurately detecting zero-velocity events across many 'styles' of locomotion—remains an open research problem.

In this chapter, we present our contributions to improve *zero-velocity-aided* inertial navigation. Specifically, we develop two robust, data-driven solutions to zero-velocity detection that are designed to operate consistently across different movement types. The first approach to zero-velocity detection introduces a learning-based motion classifier that identifies a wearer's motion type. The proposed zero-velocity detector uses this information to adjust its threshold to be optimal for the current motion. The second learned approach to zero-velocity detection replaces the classical zero-velocity detector with an LSTM neural network that directly classifies zero-velocity events. This approach effectively obviates the need for threshold tuning at test time. In this chapter, we introduce the zero-velocity-aided INS and discuss our contributions to the system. These contributions are summarized below:

1. we develop a motion-adaptive zero-velocity detector that uses a motion classifier to optimally select its threshold, given the current predicted motion type; to update the motion prediction, we use our own multiclass support vector machine (SVM) motion classifier that takes foot-mounted IMU data as input and distinguishes between common motion types (e.g., walking, running, stair-climbing);

2. we develop an LSTM-based zero-velocity detector trained to predict zero-velocity events in an end-to-end manner; this detector does not require a threshold parameter and operates independently of the wearer's motion type;

3. we incorporate these detectors within a zero-velocity-aided INS (based on an EKF) to provide robust zero-velocity pseudo-measurements;

4. we collect several kilometres of foot-mounted inertial data and ground-truth position information to evaluate our system and demonstrate that our data-driven approaches consistently outperform a number of common 'classical' zero-velocity detectors on walking, running, stair-climbing, and mixed-motion trajectories.

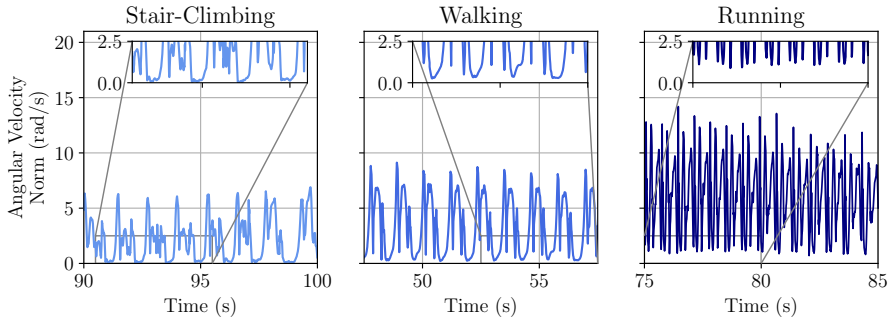


Figure 3.2: The norm of the angular velocity vector of a foot-mounted IMU during various motions. The angular velocity profile is highly dependent on the motion type.

3.1 Related Work

Zero-velocity detection can be formulated as a binary classification problem that, given a window of IMU measurements, $\mathbf{z}_\tau = \{\mathbf{u}_\tau, \dots \mathbf{u}_{\tau+W}\}$, involves choosing between one of two hypotheses:

$$\begin{aligned} \mathcal{H}_0 : &\quad \text{the IMU is moving,} \\ \mathcal{H}_1 : &\quad \text{the IMU is stationary.} \end{aligned} \tag{3.1}$$

A good detector should maximize the probability of a true detection while ensuring the risk of a false detection is reasonably low. Skog et al. (2010b) show that a likelihood-ratio test based on the Neyman-Pearson theorem can be used to classify zero-velocity intervals. The LRT, defined as

$$L(\mathbf{z}_n) = \frac{p(\mathbf{z}_\tau|\mathcal{H}_1)}{p(\mathbf{z}_\tau|\mathcal{H}_0)} > \gamma. \tag{3.2}$$

indicates a zero-velocity detection when the likelihood, $L(\mathbf{z}_n)$, exceeds a threshold, γ.

A range of statistical tests for zero-velocity detection have been proposed and studied (Skog et al., 2010a,b; Olivares et al., 2012), each based on specific assumptions or heuristics that make the terms in Equation (3.2) tractable. The stationary probability, $p(\mathbf{z}_n|\mathcal{H}_1)$, is typically based on one (or both) of two heuristic conditions that reasonably model the IMU measurements as the foot enters midstance: (1) that the measured specific force of the IMU is equal in magnitude and opposite in direction to the local gravity vector; and/or (2) that the norm of the measured angular velocity is zero. The PDFs satisfying these conditions can be modelled as normal distributions centred on the expected output for a stationary IMU (e.g., for a window of gyroscope measurements, the stationary probability is zero-mean and normally distributed, with a variance proportional to the noise associated with the gyroscope readings).[1] We introduce a range of LRT detectors that are based on these heuristics in Section 3.2.2.

A key requirement of all zero-velocity detectors, regardless of the heuristics on which they are based, is the selection of the zero-velocity threshold, γ. Choosing the threshold is not trivial; a threshold that is too low leads to a failure to report detections when the foot is in fact stationary (i.e., false negatives), while a threshold that is too high leads to false reports of zero-velocity events when the foot is still moving (i.e., false positives). Both cases result in irreversible error accumulation (Nilsson et al., 2012). To ensure localization accuracy, the

[1] Modelling the moving hypothesis, $p(\mathbf{z}_n|\mathcal{H}_0)$, on the other hand, is not as straightforward because there is a complex relationship between the wide range of typical human gait patterns and their resulting IMU readings. Since the stationary hypothesis is easier to define, the moving hypothesis can simply be modelled as a uniform distribution.

threshold parameter is typically tuned by selecting the value that leads to the minimum position error across a training set (Skog et al., 2010a) and then is subsequently fixed during future operation. Although this threshold selection strategy is reasonable for a limited range of motion (with the motion being consistent between the training and test set), the effectiveness of a fixed-threshold detector is limited during motions that are 'dynamic' in nature.[2] Dynamic motions result in varied acceleration and varied angular rate profiles when the foot makes contact with the ground (see Figure 3.2 for an illustration of angular rate variations as a function of motion type and Chan and Rudins (1994) for an analysis of impact forces during locomotion). By observing Figure 3.2, it is apparent that no threshold is sufficient to ensure accurate zero-velocity detection across all motion types; a low threshold is sufficient for walking and stair-climbing but would not detect midstance when running. Conversely, a higher threshold would be more suitable for running motions but would result in false-positive detections during walking or stair-climbing.

To improve upon fixed-threshold zero-velocity detectors, several methods in the literature have attempted to set a varying threshold that adapts to the motion of the wearer. For example, by assuming a Bayesian detection model, the threshold can be factored into a time-varying prior on the zero-velocity hypothesis and a time-varying loss for missed detections (Wahlstrom et al., 2019). Alternatively, the threshold can be modelled explicitly as a function of gait frequency (Tian et al., 2016), estimated linear velocity (Walder and Bernoulli, 2010; Ren et al., 2016), or estimated angular velocity (Ma et al., 2017). However, developing a handcrafted model that applies a proper threshold for *all* types of motion remains challenging. In our work, we attempt to remedy this by using data to learn the relationship between raw inertial data and latent variables such as the motion type of the wearer or the velocity state of the IMU. In line with our approach, others (Park et al., 2016; Rantanen et al., 2018) use data-driven motion classification to adaptively update the zero-velocity threshold. Our motivation for the use of learning stems in part from other work that successfully applies learning-based methods to process inertial data. For example, Hannink et al. (2018) present a method to train a deep CNN to predict human stride length from inertial measurements; Chen et al. (2018) use an LSTM network to directly estimate a trajectory from raw inertial data; and Cortés et al. (2018) train a CNN to regress velocities from IMU outputs.

3.2 Methodology

To create robust zero-velocity detectors, our approach utilizes learned classifiers, which have achieved state-of-the-art results in many domains. We present two classification-based methods for zero-velocity detection and describe straightforward ways to generate training labels for each that facilitate supervised learning. First, we train an SVM to classify the motion type of a wearer from a short sequence of IMU data; given the known motion type, we update the threshold of an existing zero-velocity detector to a predefined value optimized for that motion type. Second, we adopt a purely learning-based strategy by training a recurrent neural network to directly classify when the IMU is stationary. Given a sequence of inertial data, our network outputs a binary prediction of the velocity state of the IMU (i.e., whether the IMU is stationary or moving). The network is trained with a loss function that compares the network prediction to zero-velocity ground truth. Both approaches, illustrated in Figure 3.3, are data-driven, as they augment or replace the classical zero-velocity detector with models that are learned. The incorporation of these data-driven components within a filter-based INS results in a hybrid system that yields high-accuracy pose estimates.

First, in Section 3.2.1, we introduce the zero-velocity-aided INS, which consists of an error state Kalman

[2] Here, we define dynamic motions to be those that change in type (e.g., walking to running, to stair-climbing, to crawling, etc.) or intensity (e.g., slow walking versus fast walking, and jogging versus sprinting).

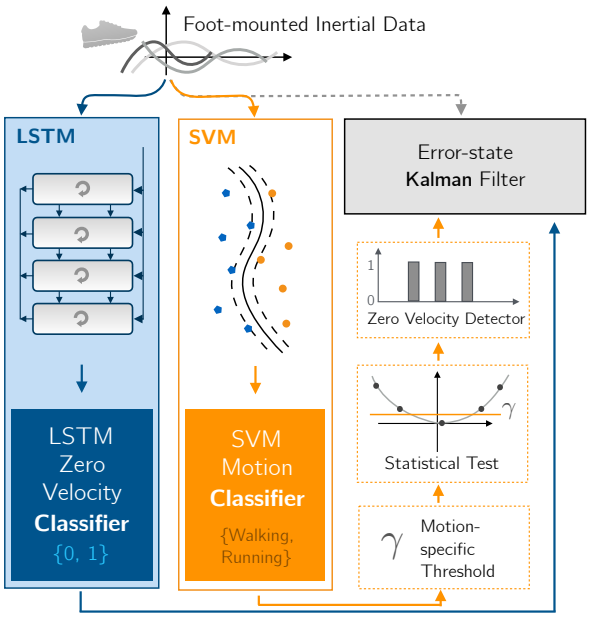


Figure 3.3: Our localization pipeline uses learning-based classifiers to improve zero-velocity detection: the motion classifier actively updates the threshold of an existing zero-velocity detector to be optimal for the current motion type, while the zero-velocity classifier replaces the zero-velocity detector with a LSTM network.

filter (ESKF) with a zero-velocity measurement model. Then, in Section 3.2.2, we describe a number of classical, threshold-based zero-velocity detectors from the literature. We follow this with descriptions of our data-driven zero-velocity detection approaches in Sections 3.2.3 and 3.2.4.

3.2.1 The Zero-Velocity Aided INS

Our system uses an ESKF to estimate the trajectory of a foot-mounted IMU. The filter state at the discrete timestep τ (and corresponding time t_τ) is

$$\mathbf{x}_\tau = \begin{bmatrix} \mathbf{r}_i^{v_\tau i} \\ \mathbf{v}_i^{v_\tau i} \\ \mathbf{q}_{iv_\tau} \end{bmatrix}, \tag{3.3}$$

and consists of the IMU position, $\mathbf{r}_i^{v_\tau i}$, velocity, $\mathbf{v}_i^{v_\tau i}$, and orientation (parameterized as a unit quaternion), $\mathbf{q}_{iv_\tau}$, all expressed within the world navigation frame $\underset{\rightarrow}{\mathcal{F}}_i$. The inputs to the system are the IMU measurements $\mathbf{u}_\tau = \begin{bmatrix} \mathbf{a}_{m,\tau}^\top & \boldsymbol{\omega}_{m,\tau}^\top \end{bmatrix}^\top$, which we assume contain additive, zero-mean noise,

$$\mathbf{w}_\tau = \begin{bmatrix} \mathbf{w}_{a,\tau} \\ \mathbf{w}_{\omega,\tau} \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}), \quad \mathbf{Q} = \begin{bmatrix} \sigma_a^2 \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \sigma_\omega^2 \mathbf{I}_3 \end{bmatrix}. \tag{3.4}$$

The state does not incorporate accelerometer or gyroscope biases (although in theory they are observable within a zero-velocity-aided system) because, as discussed in Nilsson et al. (2012), inaccuracies resulting from the zero-velocity assumption are the dominant error source (i.e., we assume that the IMU is *exactly* stationary during midstance, when in reality it always has a small, but non-zero, velocity).

The ESKF, in parallel, tracks an error-free (or *nominal*) state, $\bar{\mathbf{x}}_\tau$, alongside an error state, $\delta\mathbf{x}_\tau$. The error state propagates the errors and perturbations to the system:

$$\bar{\mathbf{x}}_\tau = \begin{bmatrix} \bar{\mathbf{r}}_i^{v_\tau i} \\ \bar{\mathbf{v}}_i^{v_\tau i} \\ \bar{\mathbf{q}}_{iv_\tau} \end{bmatrix}, \quad \delta\mathbf{x}_\tau = \begin{bmatrix} \delta\mathbf{r}_i^{v_\tau i} \\ \delta\mathbf{v}_i^{v_\tau i} \\ \delta\boldsymbol{\phi}_{iv_\tau} \end{bmatrix}. \tag{3.5}$$

As discussed in Section 2.2.3, the orientation error state, $\delta\boldsymbol{\phi}_{iv_\tau}$, is parameterized as a vector in the Lie algebra. The nominal state and error state is combined through composition to form the true state estimate,

$$\mathbf{x}_\tau = \bar{\mathbf{x}}_\tau \oplus \delta\mathbf{x}_\tau, \tag{3.6}$$

$$\begin{bmatrix} \mathbf{r}_i^{v_\tau i} \\ \mathbf{v}_i^{v_\tau i} \\ \mathbf{q}_{iv_\tau} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{r}}_i^{v_\tau i} + \delta\mathbf{r}_i^{v_\tau i} \\ \bar{\mathbf{v}}_i^{v_\tau i} + \delta\mathbf{v}_i^{v_\tau i} \\ \bar{\mathbf{q}}_{iv_\tau} \otimes \exp(\frac{\delta\boldsymbol{\phi}_{iv_\tau}}{2}) \end{bmatrix}. \tag{3.7}$$

In the prediction step of the ESKF, both the nominal and the error state are propagated independently using separate process models. The nominal state—which consists of the 'large signal' terms propagated through time using the standard dead reckoning motion model—is considered to be *error-free* and is not incorporated within the ESKF. Rather, only the 'small signal' error-state terms are filtered. Within the prediction step of the ESKF, the (Gaussian) error state and its associated covariance are propagated using a linear error-state process model. Then, with the incorporation of zero-velocity measurements, the error state is rendered observable; the measurement update step produces the corrected (a posteriori) error-state terms, which are injected back into the nominal state to refine the current estimate. After each measurement update, the error state is reset back to zero. In the next sections we discuss the prediction and measurement steps of the ESKF. We refer to Appendix C for the full derivation of the nominal/error state process models and covariance propagation model. We also refer to Solà (2015) for an in-depth explanation of the ESKF.

Prediction Step

Outside of the filter, the discrete-time motion model $f(\mathbf{x}_\tau, \mathbf{u}_\tau, \mathbf{0})$ is used to propagate the nominal state using the IMU inputs without taking into consideration the noise components:

$$\mathbf{x}_{\tau+1} = \begin{bmatrix} \mathbf{r}_i^{v_{\tau+1} i} \\ \mathbf{v}_i^{v_{\tau+1} i} \\ \mathbf{q}_{iv_{\tau+1}} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_i^{v_\tau i} + \mathbf{v}_i^{v_\tau i}\delta t + \frac{1}{2}\left(\mathbf{C}\{\mathbf{q}_{iv_\tau}\}\mathbf{a}_{m,\tau} - \mathbf{g}_i\right)\delta t^2 \\ \mathbf{v}_i^{v_\tau i} + \left(\mathbf{C}\{\mathbf{q}_{iv_\tau}\}\,\mathbf{a}_{m,\tau} - \mathbf{g}_i\right)\delta t \\ \mathbf{q}_{iv_\tau} \otimes \begin{bmatrix} \cos(\frac{\|\boldsymbol{\omega}_{m,\tau}\|\delta t}{2}) \\ \frac{\boldsymbol{\omega}_{m,\tau}}{\|\boldsymbol{\omega}_{m,\tau}\|}\sin(\frac{\|\boldsymbol{\omega}_{m,\tau}\|\delta t}{2}) \end{bmatrix} \end{bmatrix}. \tag{3.8}$$

See Section 2.3.1 for a derivation of this process model based on Euler integration. As IMU measurements are received, the non-linear process model is used to propagate the state posterior through time according to

$$\check{\mathbf{x}}_{\tau+1} = f(\hat{\mathbf{x}}_\tau, \mathbf{u}_\tau, \mathbf{0}). \tag{3.9}$$

The error state, on the other hand, is estimated within the ESKF. The discrete-time error state process model is linear with respect to the error state. Defined below, this model uses $\boldsymbol{\Phi}_{\tau+1,\tau}$, the linear error state transition

matrix, to propagate the error state during the prediction step

$$\delta\mathbf{x}_{\tau+1} = \mathbf{\Phi}_{\tau+1,\tau}\delta\mathbf{x}_\tau + \mathbf{w}'_\tau, \tag{3.10}$$

$$\mathbf{w}'_\tau \sim \mathcal{N}\left(\mathbf{0}, \mathbf{Q}'\right), \quad \mathbf{\Phi}_{\tau+1,\tau} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{I}_3\delta t & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_3 & -\mathbf{C}\{\hat{\mathbf{q}}_{iv_\tau}\}(\mathbf{a}_{m,\tau})^\wedge\delta t \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_3 - (\boldsymbol{\omega}_{m,\tau})^\wedge\delta t \end{bmatrix}. \tag{3.11}$$

For covariance propagation within the prediction step, the a posteriori state covariance $\hat{\mathbf{P}}_\tau$ is propagated through Equation (3.10) to produce $\check{\mathbf{P}}_{\tau+1}$,

$$\check{\mathbf{P}}_{\tau+1} = \mathbf{\Phi}_{\tau+1,\tau}\hat{\mathbf{P}}_\tau\mathbf{\Phi}^\top_{\tau+1,\tau} + \mathbf{Q}', \quad \mathbf{Q}' \approx \mathbf{G}\mathbf{Q}\mathbf{G}^\top\delta t, \quad \mathbf{G} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ -\mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & -\mathbf{I}_3 \end{bmatrix}. \tag{3.12}$$

We provide a more complete derivation of Equations (3.10) and (3.12) in Appendix C. Next, we discuss how zero-velocity measurements are incorporated within the update step of the filter to refine the error state estimates.

Measurement Update Step

The zero-velocity measurement model, $h(\mathbf{x}_\tau, \mathbf{n}_\tau)$, is

$$\mathbf{y}_\tau = h(\mathbf{x}_\tau, \mathbf{n}_\tau), \tag{3.13}$$

$$= \mathbf{v}_i^{v_\tau i} + \mathbf{n}_\tau. \tag{3.14}$$

Being a direct measurement of velocity, this model is effectively linear with respect to the state and with respect to the measurement noise, $\mathbf{n}_\tau \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$. We use a homoscedastic (constant) measurement covariance, $\mathbf{R} = \sigma_v\mathbf{I}_3$. To perform the measurement update within the EKF, the measurement Jacobian $\mathbf{H} = \frac{\partial h(\mathbf{x}_\tau, \mathbf{n}_\tau)}{\partial\delta\mathbf{x}_\tau}$ is required,

$$\mathbf{H} = \begin{bmatrix} \mathbf{0} & \mathbf{I}_3 & \mathbf{0} \end{bmatrix}, \quad \mathbf{H} \in \mathbb{R}^{3\times 9}. \tag{3.15}$$

Upon the arrival of a zero-velocity measurement, the predicted error state and its covariance are updated as

$$\mathbf{K}_\tau = \check{\mathbf{P}}_\tau\mathbf{H}^\top\left(\mathbf{H}\check{\mathbf{P}}_\tau\mathbf{H}^\top + \mathbf{R}\right)^{-1}, \tag{3.16}$$

$$\hat{\mathbf{P}}_\tau = \left(\mathbf{I}_9 - \mathbf{K}_\tau\mathbf{H}\right)\check{\mathbf{P}}_\tau, \tag{3.17}$$

$$\delta\hat{\mathbf{x}}_\tau = \mathbf{K}_\tau\left(\mathbf{y}_\tau - h(\check{\mathbf{x}}_\tau, \mathbf{0})\right). \tag{3.18}$$

The corrected error state is then compounded with the nominal state,

$$\hat{\mathbf{x}}_\tau = \check{\mathbf{x}}_\tau \oplus \delta\hat{\mathbf{x}}_\tau, \tag{3.19}$$

to produce the updated state posterior. Finally, after the composition step, the error state is reset to zero.

As previously discussed, the zero-velocity measurements should only be applied when the foot-mounted IMU is stationary (i.e., during the midstance phase of the gait cycle). In our system, we detect these stationary intervals using a zero-velocity detector. Next, we introduce a number of popular *classical* zero-velocity detectors

and subsequently introduce our proposed data-driven zero-velocity detection approaches.

3.2.2 Threshold-Based Zero-Velocity Detection

Below, we provide brief descriptions of five threshold-based detectors. The first four are existing LRT-based detectors from the literature. The final one is a custom zero-velocity detector that we implement using a Vicon motion capture system. These detectors all play a role in our data-driven zero-velocity detection framework: our SVM-based motion classifier (Section 3.2.3) can be paired with any of these detectors to produce a motion-adaptive detector; and our data-labelling procedure (for training the learned zero-velocity detector in Section 3.2.4) uses the output of these detectors to produce zero-velocity labels.

Stance Hypothesis Optimal Estimation (SHOE) Detector. The SHOE detector (Skog et al., 2010b) uses a small temporal window of IMU readings to either accept or reject the hypothesis that the sensor is moving relative to the navigation frame. The hypothesis is rejected (and the alternate hypothesis—that the IMU is stationary—is accepted) when the average of the L_2-norm of the gravity-subtracted acceleration, combined with an angular velocity term, falls below a threshold, γ. The binary SHOE detector output is given by

$$y_\tau = \begin{cases} 1, \text{ if } \frac{1}{W} \sum_{n=\tau}^{\tau+W-1} \left(\frac{1}{\sigma_a^2} \left\| \mathbf{a}_{m,n} - g \frac{\bar{\mathbf{a}}_m}{\|\bar{\mathbf{a}}_m\|} \right\|^2 + \frac{1}{\sigma_\omega^2} \|\boldsymbol{\omega}_{m,n}\|^2 \right) < \gamma, \\ 0, \text{ otherwise.} \end{cases} \tag{3.20}$$

The two terms in the SHOE detector LRT are weighted by the variances of the linear acceleration and angular velocity measurements, σ_a^2 and σ_ω^2, respectively. The term $\bar{\mathbf{a}}_m$ is the per-channel mean of the linear acceleration over all of the samples within the current window of W samples and the term g is the norm of the gravity vector, $g = \|\mathbf{g}_i\|$.

Angular Rate Energy Detector (ARED). The ARED (Skog et al., 2010b) shares the same angular velocity detection component as the SHOE detector, but omits the linear acceleration term:

$$y_\tau = \begin{cases} 1, \text{ if } \frac{1}{W} \sum_{n=\tau}^{\tau+W-1} \|\boldsymbol{\omega}_{m,n}\|^2 < \gamma_\omega, \\ 0, \text{ otherwise.} \end{cases} \tag{3.21}$$

Note that the trigger threshold, γ_ω, is different than that of the SHOE detector. This detector is known to produce false positives when the foot moves without rotating.

Acceleration-Moving Variance Detector (AMVD). The AMVD (Skog et al., 2010b) identifies zero-velocity events by computing the variance of the linear acceleration measurements over W timesteps. A zero-velocity event is detected when the acceleration variance falls below a predefined threshold, γ_v,

$$y_\tau = \begin{cases} 1, \text{ if } \frac{1}{W} \sum_{n=\tau}^{\tau+W-1} \|\mathbf{a}_{m,n} - \bar{\mathbf{a}}_m\|^2 \leq \gamma_v, \\ 0, \text{ otherwise.} \end{cases} \tag{3.22}$$

Memory-Based Graph Theoretic Detector (MBGTD). The MBGTD (Olivares et al., 2012) is similar to the AMVD in its use of linear acceleration values collected over a window. Rather than computing the variance of

the accelerometer signal, it computes the average Euclidean distance between samples within the window:

$$y_\tau = \begin{cases} 1, \text{ if } \left(\max_{\tau \leq i < j \leq \tau + W} \frac{\sum_{l=i}^{j-1} \sum_{n=j}^{\tau+W} \|\mathbf{a}_{m,l} - \mathbf{a}_{m,n}\|}{(j-i)(N-j+1)} \right) \leq \gamma_m, \\ 0, \text{ otherwise.} \end{cases} \tag{3.23}$$

We refer the reader to Olivares et al. (2012) for a more detailed description.

Vicon-Based Zero-Velocity Detector. Vicon-based zero-velocity detection is an approach we developed to acquire accurate zero-velocity labels for training our system. This zero-velocity detection approach uses a motion-tracking system to determine the position of the IMU accurately. The position ground truth is then used to estimate the IMU velocity through numerical differentiation. The velocity norm (i.e., the foot speed) is determined and a threshold, γ_v, is applied to detect when the foot velocity is nearly zero,

$$y_\tau = \begin{cases} 1, & \text{if } \left\| \frac{\mathbf{r}_\tau - \mathbf{r}_{\tau-1}}{t_\tau - t_{\tau-1}} \right\| < \gamma_v, \\ 0, & \text{otherwise.} \end{cases} \tag{3.24}$$

Our Vicon-based detector plays an important role in identifying midstance when the wearer's foot is moving directly upwards or downwards (e.g., when stair-climbing). During such a motion, the angular velocity and linear acceleration of the foot are both small, which can cause classical detectors to report false positives. Since the Vicon-based detector measures foot speed only, it is less likely to erroneously detect midstance during upwards or downwards motions. With a nearly perfect velocity estimate, one would expect the Vicon-based detector to be more accurate than other detectors that rely on the IMU measurements only. However, the significant impact forces associated with locomotion can cause the Vicon markers mounted to the IMU to vibrate, which can result in a noisy Vicon-based velocity estimate during walking and running.

3.2.3 Motion Classification for Adaptive Zero-Velocity Threshold Selection

Our approach for motion-adaptive zero-velocity detection incorporates a motion classifier into the zero-velocity detection pipeline, as illustrated in Figure 3.3. Since the optimal zero-velocity threshold is highly dependent on motion type, we determine a motion-specific threshold for each motion type and use the motion classifier to adaptively update the threshold of a classical zero-velocity detector. In our experiments, we train a three-motion support-vector machine (SVM) classifier—capable of classifying walking, running, and stair-climbing—for adaptive thresholding. These motions are chosen to encompass the general range of movement present during locomotion within multiple-storey buildings.

The SVM is a linear classifier that can be trained to identify the hyperplane that maximally separates groups of differently labelled data (Cervantes et al., 2020). Contrary to other linear classifiers, SVMs can classify datasets that are not linearly separable by using what is known as the kernel trick, transforming the inputs to a higher dimensional feature space where they become linearly separable. We use a popular kernel function called the radial basis function (RBF). SVMs are capable of learning a generalizable model using only a small amount of data, and are therefore a popular choice of classifier for small-scale learning applications. To train the SVM we use a dataset whose individual samples consist of a window of IMU measurements with a consistent (and known) motion type. We use the 'one-against-one' approach for multiclass classification. Additional training details are discussed in Section 3.3.2.

At test time, the SVM-based motion classifier can adaptively update the threshold of a classical zero-velocity

detector. We use the SHOE detector in our experiments but note that this method is amenable to any form of threshold-based detection. To identify the optimal threshold for each motion type, we propose a threshold optimization strategy that maximizes the F_β score of the detector across a training dataset,

$$F_\beta = \left(1 + \beta^2\right) \frac{PR}{\beta^2 P + R}. \tag{3.25}$$

In this F-measure, P refers to the detector precision and R to the detector recall, with respect to the 'ground truth' zero-velocity signal measured by the Vicon-based zero-velocity detector. The F score balances precision and recall, with the β parameter controlling the importance of precision relative to recall. For $\beta < 1$, precision is favoured over recall. Empirically, we find that precision is more important than recall, since false-positive detections are detrimental to the system when the true IMU velocity is larger than zero.

3.2.4 LSTM-Based Zero-Velocity Classification

Our second learning-based approach to zero-velocity detection removes most modelling assumptions (e.g., the assumption that distinct motion classes exist) by entirely replacing the zero-velocity detector with an RNN. We use a supervised learning approach to train a binary classifier for zero-velocity detection from a dataset containing inertial measurements with zero-velocity labels. Inertial measurements are sequential and low-dimensional, which makes them well suited for RNN sequence learning. In contrast to classical zero-velocity detection methods, which utilize a short sequence of data, an RNN is able to propagate its memory state across long input sequences. The network can therefore make use of temporal context during classification and is able to exploit the periodic nature of the human gait. We use the LSTM variant of RNNs for learned zero-velocity detection.

We employ a standard cross-entropy loss function to compare the predicted output, p_i, with the target, y_i,

$$\mathcal{L}_{CE} = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i). \tag{3.26}$$

We generate our own binary zero-velocity labels (moving versus stationary) for every sequence within our training dataset. While existing work has performed dataset labelling through pressure sensing (Skog et al., 2010a) or manual annotation (Olivares et al., 2012), we leverage the five classical detectors from the previous section that have been optimized to produce outputs that minimize the position error over a short movement sequence of one motion type. For each motion trial, the threshold of each detector is varied (using a linear search) to identify the optimal detector-specific value. The zero-velocity output from the detector that produces the lowest position error with our INS is used as the approximate zero-velocity ground truth for the trial. This zero-velocity labelling technique is based on the assumption that existing handcrafted zero-velocity detectors can produce optimal outputs if two conditions are met: (1) the wearer's motion is of a fixed type and intensity and (2) the zero-velocity threshold is optimized for the current motion. We ensure that the motion trials within the training dataset satisfy these requirements, such that a fixed threshold detector can produce an (approximately) optimal zero-velocity output over the full sequence.

3.3 Experiments

In this section, we describe the data collection and training procedures for the two proposed data-driven zero-velocity detectors. Then, we benchmark the accuracy of our data-driven approaches by comparing them to

Table 3.1: An overview of the foot-mounted inertial datasets that we collected. The Vicon dataset is used to train our learning-based classifiers. The remaining datasets are used for evaluation.

Dataset (Relevant Section)	**IMU** (Frequency)	**Ground Truth**	
		Method	Type
Vicon (Sections 3.3.1 to 3.3.2)	`MicroStrain 3DM-GX3-25` (200 Hz)	Motion Tracking (200 Hz)	3D Position
Hallway 1 (Sections 3.3.1 and 3.3.3)	`VectorNav VN-100` (200 Hz)	Surveyed (8 points)	3D Position
Hallway 2 (Section 3.3.5)	`MIMU22BTP` (125 Hz)	Surveyed (6 points)	3D Position
Stair-Climbing (Section 3.3.3)	`VectorNav VN-100` (200 Hz)	Surveyed Floor Height	1D (Vertical Only)

Figure 3.4: The Vicon motion capture area, used to collect foot-mounted inertial data with position ground truth.

existing classical zero-velocity detectors on several challenging indoor motion datasets.

3.3.1 Foot-Mounted Inertial Datasets

We collected training and evaluation datasets within three different environments at the University of Toronto. The training data were collected in a room with a Vicon infrared motion tracking system to produce the position labels for each inertial measurement. Following training, two additional evaluation datasets were collected: a hallway dataset where the IMU wearer was performing walking and running motions, and a stair-climbing dataset. Table 3.1 summarizes these datasets.

Vicon Dataset

The Vicon dataset consists of 60 motion trials (amounting to a distance of approximately 1 km in total). All trials were performed by a single person who had an IMU mounted to the top of their right foot, secured by shoe laces. For the initial data collection, we used a LORD MicroStrain 3DM-GX3-25 IMU operating at 200 Hz. The motion trials primarily consisted of walking and running at a range of speeds, but also included five stair-climbing trials and five crawling trials. Approximately ten trials involved irregular motions such as shuffling, walking backwards, or raising a foot vertically. Each motion trial was performed in our 5 m $\times$ 5 m Vicon motion capture area (see Figure 3.4), where ground truth position updates at 200 Hz were available.

We ensured that the motion within each trial consisted of a fixed type and gait frequency for two reasons. First, it simplified the labelling procedure for our motion classifier training, as all data within the trial could be assigned the same motion class. Second, it allowed us to produce high-quality zero-velocity labels using classical zero-velocity detectors with an optimized zero-velocity threshold (using the procedure described in

Section 3.2.4). In summary, we optimized each detector threshold by minimizing the position error relative to ground truth, and selected the zero-velocity output from the best-performing detector. A linear search was used to identify the threshold that minimized the average position error. These labels were then used to train the LSTM zero-velocity classifier.

Table 3.2 details the number of times each detector (with its per-trial optimized threshold) resulted in the lowest error for one of the 60 trials. The SHOE detector produced the lowest average root-mean-square error (ARMSE) in general. The Vicon-based detector and the ARED were comparable to the SHOE detector in terms of accuracy, while the MBGTD and the AMVD performed substantially worse. Table 3.2 also shows the threshold range that resulted in the minimum error over the motion trials. The wide range of the per-trial optimal threshold indicates how dependent the threshold parameter is on the motion type and is further evidence that a single threshold is not suitable for dynamic motions.

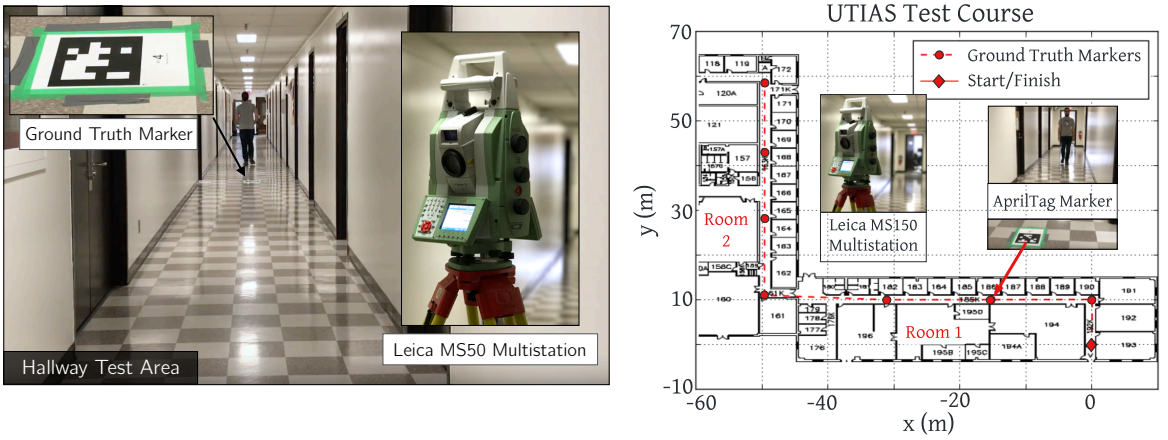
UTIAS Hallway Dataset

We evaluate our proposed data-driven zero-velocity detectors over longer trajectories by carrying out a series of motion trials in the hallways (i.e., the Hallway 1 Dataset of Table 3.1) of our building at the University of Toronto Institute for Aerospace Studies (UTIAS). Ground truth position information was obtained by surveying the locations of a series of flat markers on the floor using a Leica Nova MS50 MultiStation. We illustrate the marker locations in Figure 3.5. Each marker consisted of an AprilTag (Olson, 2011) with a side length of 28 cm affixed to the floor. Although AprilTags are a visual fiducial system, we used only their outline to define the orthogonal coordinate axes of each marker frame, with the bottom left corner of the AprilTag defining the marker itself. Because all of the markers were not visible from a single surveying location, we mapped pairs of consecutive coordinate frames to define frame-to-frame $\mathrm{SE}(3)$ transforms, and compounded them to compute the coordinates of each marker in a single navigation frame.

For data collection in these experiments, we used a VectorNav VN-100 IMU operating at 200 Hz. Test subjects were given a handheld trigger that they pressed every time their foot was directly on top of a floor marker—this allowed the INS position estimate at the current timestep to be compared with the known marker location. For each hallway trial, we report the ARMSE between the estimated position and the ground truth position across all markers along the trajectory. Data were collected from five different test subjects. All subjects started and ended at the same position on the test course. During each trial, a subject travelled approximately 110 m through three different hallways, turned around, and returned to the origin along the same path. We carried out three different types of trials: walking, running, and combined motion. For the walking and combined motion trials, the test subjects recorded their position at every marker; for the running trials, only the corner markers were used so that the test subjects did not need to slow down unnecessarily on straight sections of the course. For the combined motion trials, test subjects alternated between walking and running along the course. The path was

Table 3.2: Results for zero-velocity labelling of data from the 60 motion trials carried out within our Vicon motion capture arena. Num. Trials refers to the number of trials in which the particular detector produced the lowest ARMSE after threshold optimization.

Detector	Avg. Error (m)	Min. Thresh.	Max. Thresh.	Num. Trials
VICON	0.074	2.25×10^{-2}	8.25×10^{-1}	15
SHOE	**0.068**	4.75×10^{5}	6.50×10^{8}	**30**
AMVD	0.336	1.00×10^{-3}	1.95×10^{0}	0
ARED	0.075	1.25×10^{-2}	2.70×10^{0}	13
MBGTD	0.329	5.75×10^{-3}	9.75×10^{-1}	2



(a) The hallway testing area at UTIAS.

(b) Top-down view of the UTIAS (hallway) test course.

Figure 3.5: The hallway dataset used for evaluation of our data-driven zero-velocity detectors.

also extended to allow subjects to enter two rooms (identified as Room 1 and Room 2 in Figure 3.5). In Room 1, there was an open space where the test subjects completed three circular laps while alternating between walking and running. Subjects ascended six steps to enter Room 2 and then descended along a ramp, at which point they re-entered the last hallway. The five subjects repeated each trial three times, which resulted in a total of 45 motion trials that covered approximately 6.6 km.

Stair Climbing Dataset

We also collected a stair-climbing dataset, acquired from a single test subject who ascended and descended eight flights of stairs within a single stairwell. We note that this dataset is unique compared with other methods in the literature, which typically are only evaluated on planar surfaces. For each trial, the test subject started and finished at the same 3D position, allowing the 3D loop-closure error to be used as a performance metric. Additionally, the test subject pressed a handheld trigger to indicate when they reached a new flight of steps, which enabled us to compute the vertical error on a per-flight basis. The highest floor was approximately 16.5 m above the ground floor. Every flight consisted of 12 steps, each 17.1 cm in height, resulting in a spacing of 2.05 m between floors.

The test subject performed eight stair-climbing trials in total. The first four trials consisted of ascending from the 1^{st} to the 3^{rd}, 5^{th}, 7^{th}, and 9^{th} floors, respectively, followed by a return to the ground floor. The last four trials reversed this order, with the user starting on the 9^{th} floor and descending to the 7^{th}, 5^{th}, 3^{rd}, and 1^{st} floors, respectively, followed by an ascent back to the 9^{th} floor. In total, 80 flights of stairs were climbed, amounting to 960 steps and 164.6 m of vertical displacement.

3.3.2 Implementation Details

In this section, we discuss the training and implementation details for our data-driven classifiers. First, we discuss the training of the SVM motion classifier. Then, we discuss the training of the LSTM-based zero-velocity classifier.

SVM Details

We train two separate SVMs for motion classification. The first is a six-motion classifier, which we use to demonstrate the possibility of multiple-motion classification. The second is a three-class SVM, which we use for motion-adaptive zero-velocity classification. In both cases, we use data collected in our Vicon arena to train the SVMs. We select motion-specific trials and generate training and validation samples from them; data from the first half of the trial are used for training and data from the second half are used for validation. A total of 2,000 samples are generated per trial, with each sample consisting of one second of data. The six IMU channels are concatenated together across the window. Each sample is normalized by scaling the acceleration and angular velocity channels to be of unit norm. Every generated sample is labelled with the known motion class of the trial it originates from.

For the six-motion classifier, we collected foot-mounted inertial data from five people, who each recorded six separate motion trials that consisted of either walking, jogging, running, sprinting, crouch-walking, or ladder-climbing within the Vicon area.[3] In each trial, the users moved along a circular trajectory (with a radius of approximately three metres) for ten laps. For the motion-adaptive zero-velocity classifier, we use a subset of the Vicon dataset from Section 3.3.1 (four walking trials and five running trials), along with six additional trials from the stair-climbing dataset. We apply data augmentation by randomly rotating the specific force and angular velocity measurements within each sample to simulate additional arbitrary IMU orientations. Within each sample, the same random rotation is applied to all measurements.

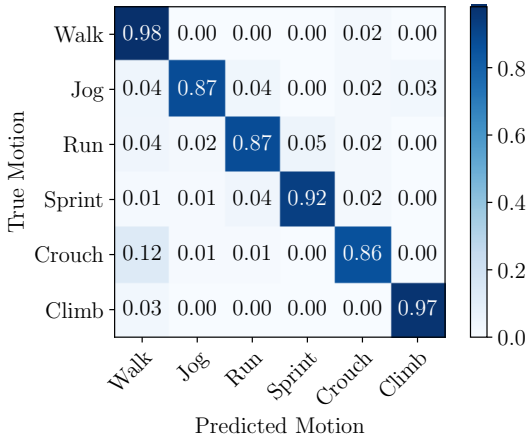
The SVMs are trained using the Python `scikit-learn` library (Pedregosa et al., 2011). We choose a radial basis function (RBF) kernel with a kernel coefficient of 0.001. Figure 3.6 presents a confusion matrix indicating the accuracies of the six-class and three-class SVM on the validation sets. For the six-class SVM, we achieve high accuracies across all classes, which demonstrates that generalization of motion classification across multiple persons and multiple motions is possible. Our three-motion classifier achieves accuracies of over 70% for all three motions, but often mis-classifies stair-climbing as walking and vice versa. We attribute this to a number of factors: the similarity of walking and stair-climbing, the smaller amount of training data compared with the six-motion SVM, and the randomization of the IMU orientation (which prevents the SVM from inferring the motion type by observing the direction of the gravity vector). We note that this reduction in accuracy is not detrimental to the zero-velocity-aided INS since the optimal thresholds for walking and stair-climbing are similar.

Following training, we select optimal zero-velocity thresholds for each motion type by maximizing the F_β score, using β^2 values of 0.16 and 0.4 for walking and running respectively. Thresholds were sampled within the range $[10^2, 10^8]$ using a linear search. Since we do not have ground-truth zero-velocity labels for the stair-climbing data, we manually select the threshold to be the same as the walking threshold. In the future, optimizing the stair-climbing threshold could lead to additional improvements.

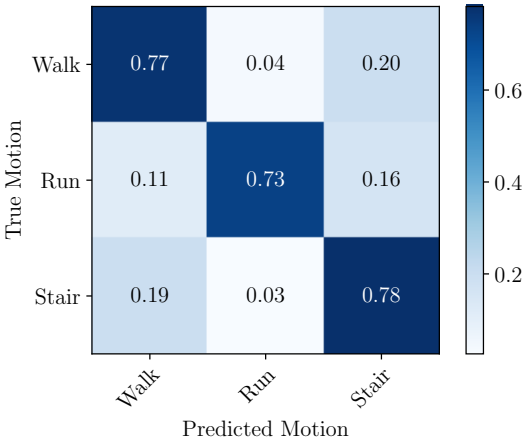
LSTM Details

Our LSTM network is composed of six layers, with 80 units per layer. The final layer is fully connected and reduces the number of outputs from the previous layer (80) to two. A softmax function bounds the two outputs to be within zero and one; the final network output is the argmax of the softmax result, which indicates whether

[3] Note that this multiple-user dataset used to train our six-class SVM was supplementary to the datasets that were introduced in Section 3.3.1.



(a) Confusion matrix illustrating the accuracy of our six-motion SVM classifier.



(b) Confusion matrix illustrating the accuracy of our three-motion SVM classifier.

Figure 3.6: Our SVM-based motion classifier results. We train a six-motion model as a proof of concept, and then use the three-class (walking, running, and stair-climbing) within our motion-adaptive zero-velocity detector.

a zero-velocity or an in-motion prediction is more likely.[4]

We train the LSTM-based zero-velocity classifier with the Vicon dataset described in Section 3.3.1. The motion trials are split into training and validation sets (51 and 9 trials, respectively). We process the raw, labelled inertial motion data into a form that is appropriate for training: 7,000 samples are extracted from each trial, with each sample spanning a window of 100 timesteps. An individual sample, $\mathbf{x}_i \in \mathbb{R}^{100\times 6}$, is paired with the zero-velocity label $y_i \in \{0, 1\}$ that indicates if the IMU is stationary at the final timestep in the window. To ensure that the network is able to generalize to new inertial data, we apply three data augmentation techniques: (1) we rotate each training sample by a random rotation; (2) we randomly scale each sample by a random factor $s \in [0.92, 1.02]$; and (3) we add zero-mean Gaussian noise ($\sigma = 0.075$). These augmentation methods simulate angular velocity and linear acceleration readings from different IMU orientations and motions to improve generalization.

We implement our LSTM network in PyTorch and train it for 300 epochs on an NVIDIA Titan X GPU, using the Adam optimizer to minimize the loss (Equation (3.26)). During training, we use gradient clipping (limiting our gradient magnitude to be less than one) to avoid issues with exploding gradients. Our training parameters include a learning rate of 5×10^{-3} (with its size reduced by half every 30 epochs), weight decay of 1×10^{-5}, and minibatch sizes of 800. At test time, we achieve real-time performance on a CPU.

3.3.3 Experimental Results

In this section, we demonstrate the effectiveness of our data-driven approaches by evaluating their performance on the hallway and stair-climbing trials. We follow this with a short discussion of the benefits and drawbacks of our system. Finally, we present the results of a domain adaptation experiment that extends our method to generalize on different (low-cost) IMUs without having to recollect new training data.

[4]The number of false-positive zero-velocity classifications is reduced at test-time by removing all positive events whose confidence level falls below 0.85.

Table 3.3: Position error results (3D ARMSE) from the hallway trials, comparing a fixed-threshold SHOE detector and ARED against our motion-adaptive zero-velocity detector and LSTM-based zero-velocity classifier. The RMSE is computed at several locations in the hallway where surveyed floor markers were placed.

Motion		**ARMSE (m)**							
Type	Subject	ARED			SHOE			Motion-Adaptive[1]	LSTM
		γ_{walk} (0.3)	γ_{mid} (0.55)	γ_{run} (0.8)	γ_{walk} (1×10^7)	γ_{mid} (8.5×10^7)	γ_{run} (3.5×10^8)	$\gamma_{walk} \mid \gamma_{run} \mid \gamma_{stair}$	–
Walking	0	1.11	2.93	4.25	0.54	1.26	6.16	0.56	**0.52**
	1	1.10	1.35	1.57	0.84	1.15	2.09	0.85	**0.81**
	2	2.51	2.73	2.95	2.27	2.66	3.86	2.43	**2.04**
	3	**1.07**	1.39	1.69	1.09	1.20	3.31	1.08	1.10
	4	0.65	0.73	0.82	**0.62**	0.73	1.24	0.75	0.64
Running	0	1.17	1.15	2.66	3.31	0.87	2.44	0.89	**0.87**
	1	3.70	1.86	1.93	16.45	1.64	0.88	0.76	**0.67**
	2	1.47	1.31	1.43	1.64	1.32	2.43	1.36	**0.86**
	3	**0.62**	1.33	1.54	1.04	1.29	1.31	0.67	0.62
	4	1.25	15.01	15.91	1.29	1.17	2.04	1.96	**0.93**
Combined	0	1.15	1.72	2.42	1.05	1.25	3.59	1.00	**0.90**
	1	2.29	2.02	2.20	2.00	1.73	2.31	1.35	**1.20**
	2	2.47	2.97	3.30	2.09	2.75	3.91	2.35	**1.65**
	3	1.11	1.28	1.37	1.10	1.22	2.37	1.10	**1.07**
	4	2.57	2.66	2.64	2.51	2.63	1.60	**1.00**	1.24
Mean		1.52	2.48	3.03	2.38	1.44	2.81	1.14	**0.97**

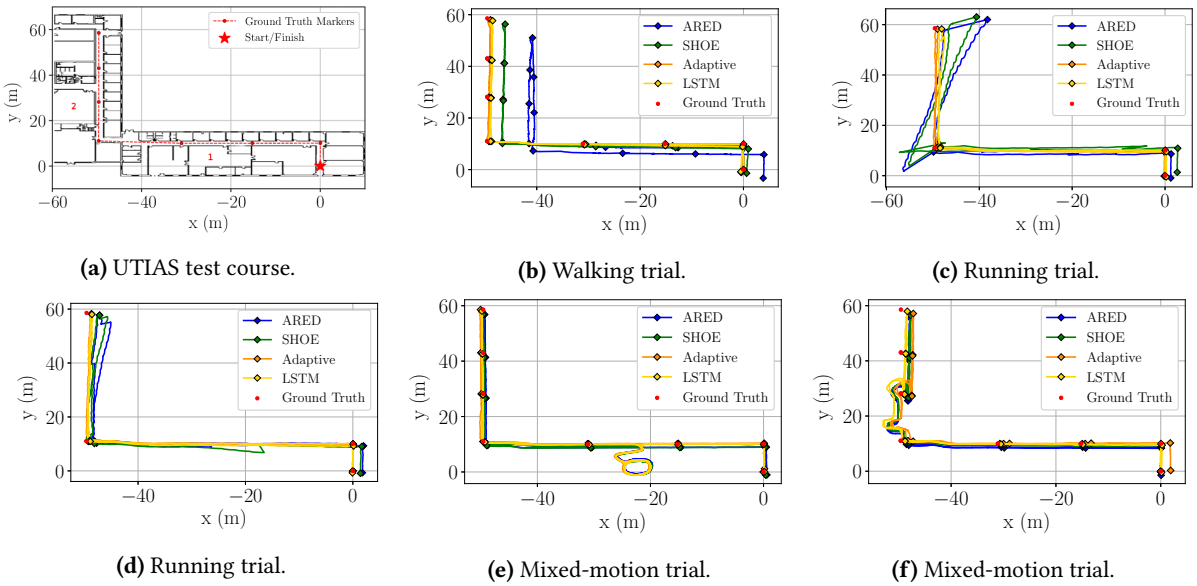
[1] For the motion-adaptive thresholds, the appropriate SHOE detector thresholds are used: $\gamma_{walk} = 1 \times 10^7$, $\gamma_{run} = 3.5 \times 10^8$, and $\gamma_{stair} = 1 \times 10^7$.

Hallway Trial Results

For each trial, we evaluate our baseline INS with the SHOE detector, the ARED, our motion-adaptive detector, and our LSTM-based zero-velocity classifier. Table 3.3 shows the performance of these detectors for the hallway trials. Results for the SHOE detector and the ARED are calculated using three fixed thresholds that are optimized for walking (γ_{walk}), running (γ_{run}), and for achieving consistent performance across the mixed-motion Vicon test dataset (γ_{mid}). On average, our LSTM detector achieves a 32.6% lower ARMSE than the most accurate fixed-threshold detector (SHOE with γ_{mid}). Figures 3.7 and 3.8 illustrate the increase in accuracy that our LSTM-based classifier achieves for the hallway trials. Our motion-adaptive detector produces a similar accuracy as the LSTM detector, but notably is not as accurate at height estimation (see Figure 3.8).

Stair-Climbing Results

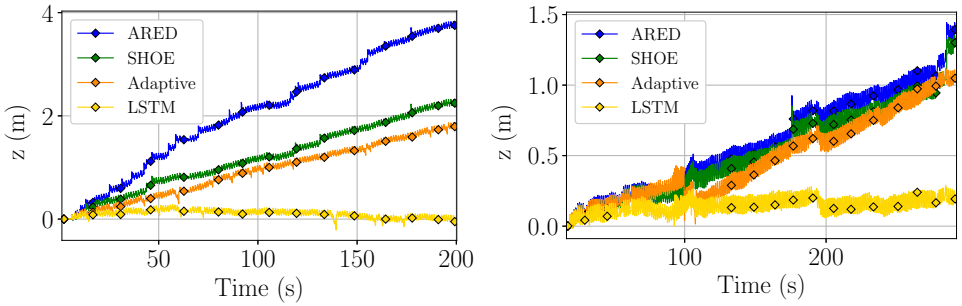
Again, we compare the performance of the motion-adaptive detector and the LSTM-based zero-velocity classifier to the SHOE detector and ARED. Both the SHOE detector and the ARED use fixed thresholds that, in this case, are optimized to minimize the error on the Vicon test set. While a threshold optimized for stair-climbing would produce more accurate results for the SHOE detector and for the ARED, we omit this from our comparison because optimizing for stair-climbing would substantially degrade detector performance for more common motion types. Table 3.4 lists the results for the 3D and vertical loop-closure error and the vertical *farthest-point error*, which we define as the difference between the estimated height of the IMU and its known height at the farthest point of the trajectory. In all cases, our data-driven detectors outperform the SHOE detector and the ARED by a large margin. Figure 3.9 depicts the results from our stair-climbing experiments. Figures 3.9a and 3.9b illustrate failure modes of the SHOE detector and the ARED: these hand-engineered detectors both overestimate *and* underestimate vertical displacement to such a degree that even floor-level accuracy cannot be attained. In



(a) UTIAS test course. **(b)** Walking trial. **(c)** Running trial.

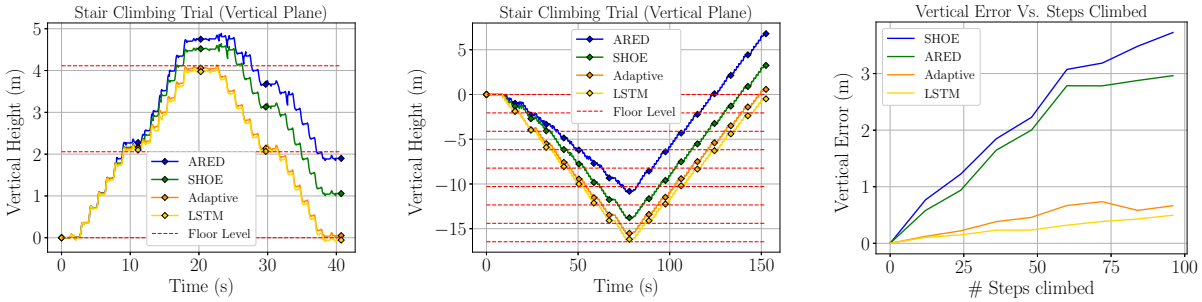
(d) Running trial. **(e)** Mixed-motion trial. **(f)** Mixed-motion trial.

Figure 3.7: INS trajectories (top-down view) for walking, running, and mixed-motion trials. The proposed data-driven detectors (the adaptive and LSTM-based methods) consistently outperform classical zero-velocity detectors that rely on fixed thresholds.



(a) Walking trial (vertical view). **(b)** Mixed-motion trial (vertical view).

Figure 3.8: Vertical trajectory estimates for motion on a planar surface (a flat floor). The vertical offsets are due entirely to INS drift.



(a) Height over time (two floors up and down).

(b) Height over time (eight floors down and up).

(c) Cumulative error for the stair-climbing trials.

Figure 3.9: Results from the stair-climbing trials. The LSTM-based zero-velocity detector outperforms existing detectors in terms of the vertical position error.

Table 3.4: Results from the stair-climbing trials. Loop-closure error is computed at the end of the trajectory (when the user returned to the origin).

# Flights	Detector	Position Errors (m)		
		Loop-Closure (3D Error)	Loop-Closure (Vertical Error)	Farthest-Point (Vertical Error)
2	ARED	1.895	1.878	1.265
	SHOE	1.014	1.001	0.703
	Adaptive	**0.091**	0.055	**0.109**
	LSTM	0.098	**0.052**	0.115
4	ARED	3.155	2.865	2.015
	SHOE	1.841	1.18	1.526
	Adaptive	**0.321**	**0.199**	**0.146**
	LSTM	0.343	0.251	0.172
6	ARED	2.443	2.109	3.294
	SHOE	1.833	1.499	2.945
	Adaptive	1.124	0.513	0.931
	LSTM	**0.362**	**0.279**	**0.396**
8	ARED	4.83	4.811	3.725
	SHOE	2.312	2.244	2.963
	Adaptive	0.853	0.648	0.665
	LSTM	**0.73**	**0.495**	**0.496**
Mean	ARED	3.081	2.916	2.575
	SHOE	1.75	1.481	2.034
	Adaptive	0.597	0.354	0.463
	LSTM	**0.384**	**0.269**	**0.295**

contrast, our proposed detectors on average produce a vertical estimate that exceeds the accuracy required for floor level identification. Figure 3.9c plots the vertical displacement error against the number of steps climbed for all trials. The proposed detectors maintain an accuracy of better than one metre over a range of 100 steps, while the SHOE detector and the ARED exceed this error bound within 25 steps.

3.3.4 Discussion

Our proposed zero-velocity detectors operate consistently with five different test subjects, while being trained with data from a single subject only. Furthermore, the detectors operate largely independently of the orientation of the IMU on the foot and are invariant to the location of the IMU on the shoe and to the shoe type in general (we

Table 3.5: Operating frequencies for several zero-velocity detectors. Our proposed detectors run in real time on an Intel i7-6700HQ CPU.

	ARED	SHOE	Adaptive	LSTM
Operating Frequency (kHz)	215	32.2	1.97	3.35

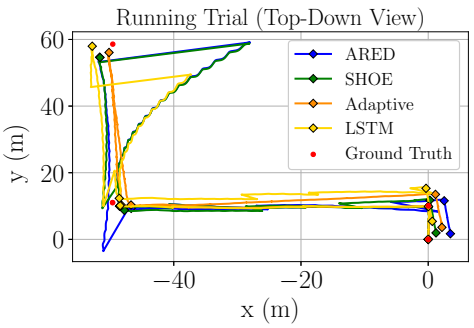


Figure 3.10: A failure case for our LSTM, where the test subject's velocity is outside of the training distribution.

did not specify where the IMU should be mounted or what type of shoe should be worn for data collection). The learning-based detectors are able to run on a CPU in real time; we report the operating frequencies in Table 3.5; both of the proposed detectors, although slower than the classical detectors, are able to function at a frequency above approximately 2 kHz. In general, this operating frequency is much greater than the operating frequency of an IMU.

Despite the increase in accuracy (and the other benefits described in the previous paragraph) provided by the learning-based detectors, there are areas where further improvements could be made. First, the motion-specific threshold of the zero-velocity detector could be extended to additional motions. Figure 3.6a illustrates how the SVM classifier accurately classifies a number of additional motion types; by extending our motion-adaptive zero-velocity detector to include these motion types, our system could become more robust to varying motion. Second, the LSTM-based zero-velocity classifier could make use of training data collected in areas other than our Vicon room and at greater velocities (as seen in Figure 3.10, the LSTM model is occasionally unable to detect zero-velocity events when a wearer's speed is outside of the training distribution).

Lastly, we note that the increase in positioning accuracy derived from the use of our learning-based detectors is primarily a result of an improved velocity estimate. Although the zero-velocity updates do impact the IMU roll and pitch estimates, the yaw (heading) remains unobservable (Nilsson and Handel, 2014). Classical zero-velocity detectors such as the SHOE detector permit roll and pitch to be recovered accurately (because the magnitude of the gravity vector, which appears in the error state computation, is large in comparison with other measured accelerations); we find that the use of learning-based detectors does not substantially change the accuracy of the attitude estimate.

3.3.5 Generalizing to New IMUs

Although we have shown that our LSTM-based zero-velocity classifier generalizes to new IMU placements, new users, and varying motion types, we do not expect it to function properly with an IMU that has significantly different *hardware* characteristics. To improve this, we propose a data manipulation technique that transforms the data within the training dataset to be representative of the outputs of a different IMU. By retraining the zero-velocity classifier with the transformed data, the LSTM network is able to generalize to inertial data from

a different, lower-quality sensor. Importantly, this approach obviates the need to collect any new training data with the alternate IMU, which would be tedious, time-consuming, and potentially infeasible.

Our proposed transformation technique adds zero-mean Gaussian noise to each IMU channel and then down-samples the original data to a lower frequency. These steps are meant to account for varying measurement quality and for varying IMU sample rates. The newly transformed data are then used to retrain the LSTM network to be compatible with an alternate IMU. We note that our data manipulation approach can only be used to simulate an IMU with higher levels of noise and a lower update frequency than the IMU used to collect the training data. Despite this limitation, given a training dataset collected with a high-quality IMU (e.g., our Vicon dataset), there is a wide array of lower-cost IMUs to which the data manipulation technique is applicable.

IMU Generalization Experiments

Our initial LSTM-based zero-velocity detector is trained with data from the Vicon dataset, collected with the MicroStrain 3DM-GX3-25 IMU operating at 200 Hz. While we demonstrate that the learned model generalizes to the (comparable) VectorNav VN-100 IMU at test time, both IMUs have similar noise characteristics and operate at the same frequency. Herein, as a proof-of-concept study, we show that our model can be retrained to operate with the low-cost Osmium MIMU22BTP IMU, running at 125 Hz, which has significantly noisier accelerometers and gyroscopes. For brevity, we refer to this low-cost IMU as the Osmium IMU.

We transform the Vicon training dataset to be representative of the data generated by the Osmium by down-sampling from 200 Hz to 125 Hz[5] and adding Gaussian noise to each sample ($\sigma_a = 1\times10^{-2}$ and $\sigma_\omega = 1.74\times10^{-3}$ for the accelerometer and gyroscope channels, respectively). We then retrain the LSTM model with the modified VICON dataset using the same LSTM training procedure described in Section 3.2.4.

We evaluate the LSTM classifier (i.e., both the original and the retrained models) on the Hallway 2 dataset (see Table 3.1) that includes data from five test subjects walking and running with the Osmium mounted on their right foot. We evaluate the end-point error at the farthest point along the trajectory and we report the average end-point error for each motion type. Table 3.6 shows that the retrained LSTM-based zero-velocity classifier results in the most accurate position estimates—its use led to an 18.4% reduction in mean error with respect to the original LSTM network trained with the MicroStrain data. These results indicate that the proposed IMU generalization method can facilitate the use of the LSTM-based classifier with new IMUs without the need to recollect any training data.

3.4 Summary and Future Work

Ensuring correct zero-velocity detection is a crucial step towards producing a robust pedestrian navigation system. In this chapter, we have presented two new techniques for zero-velocity detection that are robust to varying motion type. Our SVM-based motion-adaptive detector operates as a motion classifier to actively update the threshold parameter of a classical zero-velocity detector. In contrast, our LSTM-based zero-velocity classifier directly outputs zero-velocity pseudo-measurements without requiring any motion-specific parameter tuning. We showed that both of our proposed techniques outperform existing threshold-based detectors on several large datasets involving walking, running, stair-climbing, and mixed-motion trials. The LSTM-based zero-velocity classifier produced the lowest error on average, followed by our SVM-based motion classifier. Lastly, we introduced a generalization method that permits the use of our LSTM-based classifier with lower-cost IMUs without the need to collect additional training data.

[5] We apply a first-order low-pass (Butterworth) filter with a cutoff frequency of 40 Hz to prevent any aliasing effects.

Table 3.6: IMU generalization results. When evaluating the LSTM on the Hallway 2 dataset (which is collected with the low-cost Osmium IMU), the retrained LSTM network outperforms the original LSTM trained with the MicroStrain IMU data.

Motion		**End-Point ARMSE (m)**					
Type	Subject	SHOE			Adaptive	Original LSTM	Retrained LSTM
		γ_{walk}	γ_{mid}	γ_{run}	—	—	—
Walking	1	1.17	1.86	2.23	1.24	**0.98**	1.30
	2	0.76	1.10	1.27	0.74	0.77	**0.72**
	3	**0.31**	0.65	0.85	0.34	0.45	0.32
	4	0.60	1.47	3.07	0.62	**0.48**	0.56
	5	1.02	1.73	2.17	1.01	1.26	**0.92**
Running	1	12.36	2.39	2.39	**2.22**	2.30	2.40
	2	99.11	3.74	1.82	1.82	4.02	**1.72**
	3	62.27	**1.73**	1.82	1.82	1.79	1.80
	4	154.88	3.37	4.90	4.89	3.31	**3.28**
	5	128.31	**1.12**	1.48	1.48	4.09	1.33
Combined	1	5.45	2.61	2.91	2.70	2.69	**2.60**
	2	2.80	2.33	2.40	2.30	**1.93**	2.00
	3	2.50	0.76	0.86	**0.70**	0.71	0.72
	4	11.66	2.26	2.72	2.48	2.13	**2.12**
	5	0.87	1.29	1.64	1.09	**0.77**	0.83
Mean		32.27	1.89	2.17	1.70	1.85	**1.51**

This chapter empirically demonstrates how data-driven models can be used to augment a filter-based estimator. Building on our current work in the domain of zero-velocity-aided navigation, future work may investigate the possibility of removing the zero-velocity detector component and directly learning a zero-velocity measurement model using a differentiable filter (Haarnoja et al., 2016). This modification would allow the system to be trained end-to-end by minimizing a pose loss directly and would allow for a heteroscedastic measurement noise covariance model to be learned. By producing uncertainty-aware measurements, the system could properly inflate the measurement uncertainty when measurements are unreliable (e.g., during high-intensity motions, when the IMU may not be completely stationary during midstance). A similar approach could be taken to more accurately model sensor noise. For example, inflating the process noise covariance during the high acceleration period that occurs when the heel strikes the ground can mitigate error growth (Ju et al., 2018).

Future work may also investigate how other pseudo-measurements can be learned and incorporated into our estimator. The zero-angular-rate update (ZARU) detector, for example, provides pseudo-measurements of the gyroscope bias during midstance when the angular velocity of the IMU is approximately zero (Ashkar et al., 2013). Detecting these periods, however, is challenging, and could be accomplished with a data-driven model instead. Finally, our system could incorporate magnetometer measurements, which can be used to render the yaw (heading) direction observable. In systems with magnetometer-based heading error reduction, magnetic disturbances must be accounted for properly and are usually removed/ignored by applying handcrafted heuristics. Like the classical zero-velocity detector, these heuristics are based on simplifying assumptions and are brittle as a result. Therefore, there is motivation to replace them with data-driven models.

Chapter 4

Visual Egomotion Estimation

Next, we shift our focus from inertial navigation to visual egomotion estimation, or *visual odometry* (VO).[1] In this chapter, we introduce the general framework for visual egomotion estimation and contrast the existing 'classical' approaches with alternative data-driven systems. Since the focus of this thesis is to investigate how learning-based methods can be used for robust self-localization, after identifying some of the primary failure modes of classical methods, we propose a data-driven approach as a robust alternative. In particular, we introduce a framework for data-driven visual egomotion estimation that utilizes domain knowledge from the field of structure from motion (SfM). After introducing the learned SfM system within this chapter, we present our contributions to improve the system in Chapters 5 and 6.

4.1 Classical Approaches to VO

Visual odometry, a term first coined by Nistér et al. (2004) in their seminal paper, is the estimation of egomotion using a stream of visual measurements from one or more cameras. The first VO system was proposed in the early 1980s by Moravec (1980). In the following two decades, VO systems were developed extensively for NASA's Mars exploration program (Scaramuzza and Fraundorfer, 2011). The pioneering work of Nistér et al. (2004), who proposed the five-point algorithm for estimating camera motion by computing the essential matrix in real time, led to a number of works that further advanced the field of 'classical' VO. These advances were also made possible in part by the development of simultaneous localization and mappping (SLAM) (Durrant-Whyte and Bailey, 2006), which is closely tied with VO; the *front-end* of SLAM systems often incorporate VO algorithms to produce local motion estimates, which are then refined within the *back-end*. Further, VO methods have been influenced by *structure from motion* (SfM) (Longuet-Higgins, 1981) techniques first used in the field of computer vision. At its core, SfM involves the joint estimation of camera motion and scene structure through an optimization procedure known as bundle adjustment (Triggs et al., 1999).

Modern VO systems are distinguished by the amount of image information they utilize for motion estimation (i.e., sparse versus dense) and whether a photometric or geometric loss is minimized (i.e., direct versus indirect). In *sparse* approaches, a relatively small number of image features are extracted and tracked between frames; their 3D locations are then estimated. Conversely, *dense* approaches, which minimize a direct photometric loss based on image alignment, maintain depth estimates for a substantially larger number of pixels in the image.

[1] In this chapter, we use the terms 'VO' and 'visual egomotion estimation' interchangeably. VO is the more popular term in the field of robotics, while egomotion estimation is more popular in the field of computer vision.

4.1.1 Feature-Based (Indirect) Methods

The three primary steps for sparse VO are: (1) extracting a relatively small number of points, or *features*, within the image; (2) establishing correspondences of these features between frames; and (3) using one of several motion estimation algorithms based on triangulation to determine camera motion.[2]

Feature Detection and Tracking. Feature detection usually involves, but is not limited to, the extraction of 'corner' pixels that have large gradients in the horizontal and vertical directions in the image (Li, 2017). After these features are identified, their correspondences in nearby frames are established. Establishing feature correspondence involves matching features with a similarity-based metric, or tracking how features move between frames (e.g., using optical flow).

Motion Estimation. Once correspondences are established, motion estimation is performed using one of several possible methods. The '2D-to-2D' method uses the epipolar constraint to compute the essential matrix, from which the camera motion can be extracted. The simplest form of 2D-to-2D estimation is the five-point algorithm (Nistér et al., 2004), which requires five points with known correspondence between two frames. This method is repeated within a random sample consensus (RANSAC) (Fischler and Bolles, 1981) procedure for determining the essential matrix that maximizes the number of inlier point correspondences. The '3D-to-2D' method explicitly estimates 3D feature locations and determines the camera motion through the minimization of a reprojection loss. Here, the estimated 3D coordinates for each feature are projected to the image plane and directly compared with their observed (measured) locations.

Local Optimization. Following interframe motion estimation, the camera poses and feature locations can be optimized across a window of frames using a filtering approach, or sliding-window bundle adjustment. Note that this joint optimization becomes more computationally difficult as the number of tracked features increases.

4.1.2 Dense (Direct) Methods

In contrast to indirect methods, which minimize a reprojection error, direct methods minimize a photometric loss based on image alignment in order to estimate the camera motion between frames. The loss is minimized when pixels from the current frame are projected to their corresponding (correct) locations in the previous image. This image alignment procedure is based on the photometric consistency assumption, that is, that there is constant scene illumination and that points within the scene have Lambertian reflectance. For dense methods, the photometric loss is minimized for all (or most) of the pixels within the image, causing the overall computational burden to increase relative to feature-based methods. As a result, joint estimation of scene structure and camera motion cannot be performed in the same way that it can for sparse optimization methods. Instead, existing methods (Engel et al., 2013) use alternate optimization steps for estimating camera motion and updating the 3D scene representation. First, given the current (fixed) 3D scene representation, camera motion between the previous and current frame is estimated through dense image alignment. Then, given the (fixed) motion estimate, the 3D scene representation is transformed to the next frame and updated. Most methods, rather than tracking *all* pixels within the image, adopt a semidense approach that only takes into consideration pixels that have strong local gradients. Since image regions without texture do not impact the photometric loss, the removal of low-gradient pixels has very little impact on performance but can substantially reduce the computational load.

[2]We note that some methods deviate from this categorization. Forster et al. (2014) (SVO) and Engel et al. (2017) (DSO), for example, demonstrate sparse, direct methods based on minimization of a photometric loss across a subset of high-gradient pixels in the image.

4.1.3 Failure Modes

Classical VO approaches exhibit a high degree of accuracy within 'nominal' operation conditions, where there is constant illumination, sufficient scene texture, and few dynamic objects within the scene as well as sufficient overlap between consecutive frames. Outside of these conditions, the modelling assumptions built into the various VO frameworks may no longer hold, resulting in degradation of accuracy, or even failure to produce a pose estimate. These failure modes may differ between different types of estimators.

In sparse feature-based systems, accurate (and robust) localization is predicated on the continued existence of a sufficient number of outlier-free correspondences between frames. Adverse operating conditions can result in either a lack of detected features or in the failure to track or match existing features between frames. The detection of new features is limited, for example when operating in environments with low light or insufficient texture, or as a result of motion blur from rapid camera motion. Failed tracking or matching between frames happens when: (1) there is an insufficient number of matched correspondences or (2) features are incorrectly matched to produce outlier correspondences. The former can occur within degraded visual environments or as a result of large camera motion that diminishes the scene overlap between frames; the latter case is often a result of the presence of dynamic objects or repeated texture within the scene.

Dense methods forego the feature tracking/matching step, since correspondences are not required to evaluate the direct (photometric) loss based on image alignment. These methods can therefore maintain accuracy over feature-based methods within some types of visually degraded environments (e.g., scenes with little texture). Instead, the primary failure mode is the divergence of the alignment optimization procedure. Divergence may occur as a result of poor initialization or as a result of violations of the brightness constancy assumption. Direct methods are therefore best suited to conditions with small changes in perspective between frames and constant scene illumination.

4.2 Learned Approaches to VO

New data-driven paradigms for VO replace portions (or all) of the classical localization pipeline with learned models. We limit our discussion here to 'end-to-end' approaches, leaving the discussion of 'hybrid' approaches for Chapter 7. End-to-end approaches parameterize the estimator as a neural network (typically a CNN) that takes as input a pair of images and directly regresses the interframe pose change between frames. These end-to-end networks can be trained with either supervised or self-supervised losses. In the next sections, we introduce both types of systems in detail.

4.2.1 Supervised Learning Approaches to VO

If ground truth pose labels are available, egomotion labels can be generated for the training data and a pose supervision loss can be formulated. The pose supervision loss for an image pair with predicted translation $\mathbf{r}_j$ and rotation $\boldsymbol{\phi}_j$ between frames is generally the mean squared error (MSE) between the ground truth egomotion (consisting of translation $\mathbf{r}'_j$ and rotation $\boldsymbol{\phi}'_j$ terms) and the network predictions (Wang et al., 2017):

$$\mathcal{L}_{sup} = \left\|\mathbf{r}'_j - \mathbf{r}_j\right\|_2^2 + \kappa \left\|\boldsymbol{\phi}'_j - \boldsymbol{\phi}_j\right\|_2^2 . \tag{4.1}$$

Here, $\boldsymbol{\phi}$ is a parameterization of the rotation between the two image frames and is usually a constraint-free representation such as an axis-angle vector or Euler angles. There are, however, other rotation representations that have been shown to be more suitable for learning, including continuous representations in 5D and 6D

(Zhou et al., 2019c). The constant κ in the loss function is added to balance the magnitude of the translation and rotation errors. Using a large value for κ is particularly important for driving datasets, since the translation is often several orders of magnitude larger than the rotation.

Networks trained to minimize a pose supervision loss are primarily distinguished by the inputs and the network structures that are employed. The standard network input is a stacked image pair. Alternatively, dense optical flow estimates (Costante et al., 2015; Costante and Ciarfuglia, 2018) can be used as inputs. From these inputs, the baseline egomotion network consists of a CNN that returns the 6-DOF egomotion prediction (Konda and Memisevic, 2015). Given the sequential nature of visual data, a natural extension is to incorporate RNN layers into the baseline network structure (Xue et al., 2019; Wang et al., 2017).

Supervised approaches have a number of positive and negative aspects. One positive aspect of the supervised loss is that it provides the network with scale information (through the translation labels). Minimization of the loss therefore results in a *scale-aware* egomotion network. The primary drawback of supervised learning, however, is that it requires high-quality pose labels that can be arduous to obtain (and may not always be accurate). The scalability of systems trained through supervised learning alone, therefore, is limited compared to that of self-supervised alternatives.

4.2.2 Self-Supervised Learning Approaches to VO

Self-supervised learning has become popular in recent years, motivated by the potential to train learned systems without the need to secure a large amount of training labels. Further, self-supervised losses facilitate lifelong learning by enabling network retraining with new data acquired at test time. For end-to-end VO systems, a self-supervised training procedure based on the SfM framework has recently gained attention (Zhou et al., 2017); here neural networks parameterize a direct mapping from pixel space to scene depth and camera motion. To train these networks, a photometric reconstruction loss, which encodes the difference between a given 'target' image and a virtual image, is minimized. This virtual image is a reconstruction of the target image synthesized by warping a nearby source view. The source view is warped through a view synthesis procedure using the estimated scene depth and the relative camera pose (egomotion).

The general framework for self-supervised learning of SfM is illustrated in Figure 4.1. There are three primary components: (1) the networks, DepthNet and EgoNet, which respectively produce a depth and egomotion prediction for a source-target image pair; (2) a view synthesis module that uses the predictions to reconstruct the 'target' image from pixels within the nearby 'source' images; and (3) the photometric reconstruction loss, which is minimized during training. The self-supervised SfM framework that originated with *SfMLearner* (Zhou et al., 2017) has subsequently been developed by many others, including Godard et al. (2019) who proposed a number of improvements in *MonoDepth2*. Below, we provide a general overview of a self-supervised system that is representative of the majority of existing approaches in the literature. We build upon this system in the remaining chapters of the thesis.

Network Structures

For a single image pair consisting of a source image and a target image, $\mathbf{x} = \{\boldsymbol{\mathcal{I}}_s, \boldsymbol{\mathcal{I}}_t\}$, the latent variables of interest (i.e., those required to reconstruct the target image from source image pixels) are the target image depth $\boldsymbol{\mathcal{D}}_t \in \mathbb{R}^{H \times W}$ and the interframe pose change (egomotion) $\mathbf{T}_{st} \in \mathrm{SE}(3)$. Here, H and W are the original height and width of the target image. We produce the depth and egomotion estimates from two separate networks: DepthNet, $\boldsymbol{\mathcal{D}}_t = f_{\boldsymbol{\theta}_D}(\boldsymbol{\mathcal{I}}_t)$, and EgoNet, $\mathbf{T}_{st} = f_{\boldsymbol{\theta}_E}(\boldsymbol{\mathcal{I}}_s, \boldsymbol{\mathcal{I}}_t)$. We provide a high-level overview of these networks

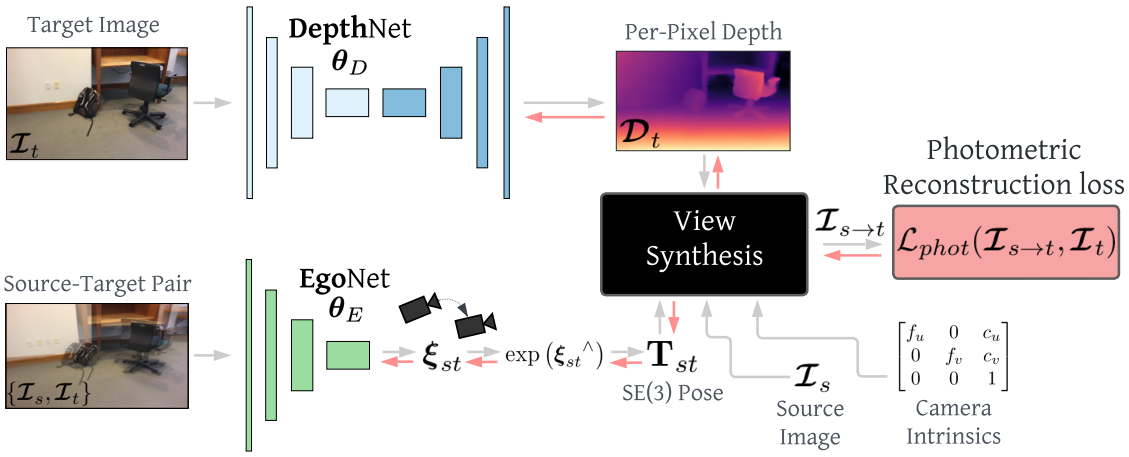


Figure 4.1: Overview of the baseline system that employs a depth and egomotion network for learned SfM. The predictions from these networks are used to reconstruct a target view from the pixels of a nearby source view through view synthesis. The photometric reconstruction loss minimizes the error between the target and reconstructed view, jointly training the depth and egomotion networks. Gray arrows represent the forward pass, while red arrows represent the backwards pass during backpropagation.

in this section and refer to Appendix B for more detailed descriptions.

DepthNet. The depth network takes as input a single image and returns the per-pixel prediction of the scene depth. A U-Net autoencoder network (Ronneberger et al., 2015), depicted in Figure 4.1, is used to ensure the input and output dimensions are equal. The encoder—generally a ResNet-style network (He et al., 2016)—extracts features from the image. Within the encoder, the dimensionality of the feature maps is reduced as stride-2 convolutions are applied. Following the encoder, a series of decoder blocks increase the dimensionality back to the original image size; these blocks generally consist of 2D transposed convolution layers (Dumoulin and Visin, 2016) or interpolation layers (e.g., nearest neighbour interpolation). Other work (Guizilini et al., 2020a) has proposed different upsampling schemes as well. The final network layer, in practice, outputs the *inverse* depth prediction, $\mathcal{D}_t^{-1}$, produced through[3]

$$\mathcal{D}_t^{-1} = \frac{1}{d_{min}} + \left(\frac{1}{d_{max}} - \frac{1}{d_{min}}\right)\mathbf{O}_t. \tag{4.2}$$

Here, the network output $\mathbf{O}_t$ is a single-channel $H \times W$ map, the per-pixel values of which are within $[0, 1]$. The final depth prediction $\mathcal{D}_t$ is found by computing the *Hadamard inverse* of $\mathcal{D}_t^{-1}$ (i.e., by computing the element-wise inverse). The resulting per-pixel depth values are all within $[d_{min}, d_{max}]$. The minimum and maximum depths, d_{min} and d_{max}, are tunable hyperparameters.

EgoNet. The egomotion network produces an estimate of the interframe pose change,

$$\mathbf{T}_{st} \in \mathrm{SE}(3) = \begin{bmatrix} \mathbf{C}_{st} & \mathbf{r}_s^{ts} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}, \quad \mathbf{C}_{st} \in \mathrm{SO}(3), \tag{4.3}$$

[3]The inverse depth parameterization is analogous to the similar technique used for classical depth estimation (Montiel et al., 2006). This parameterization prevents the network from having to directly regress (often large) depth values that span several orders of magnitude.

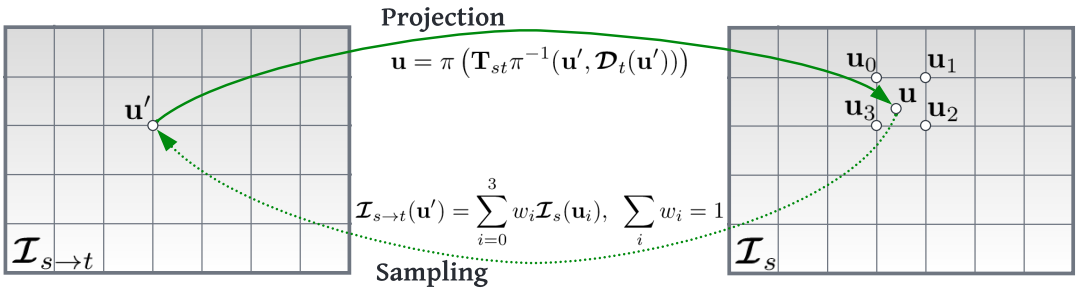


Figure 4.2: Illustration of the view synthesis procedure for a single pixel, consisting of projection and sampling steps. Bilinear interpolation (using pixel intensities from the nearest locations) is used to populate $\mathbf{u}'$ in the synthesized image $\mathcal{I}_{s\to t}$.

between $\underset{\to}{\mathcal{F}}_c$ at frames t and s (i.e., the target and source frames). The network is a standard CNN consisting of a series of convolutional blocks with nonlinear activations. Multi-stride convolutions are applied to downsample the feature maps instead of using pooling layers, since pooling layers remove spatial information that is crucial for encoding camera motion. The final output of EgoNet is the egomotion prediction, $\boldsymbol{\xi}_{st} \in \mathbb{R}^6$, which is parameterized as a vector in $\mathfrak{se}(3)$. The SE(3) pose is computed according to $\mathbf{T}_{st} = \exp\left(\boldsymbol{\xi}_{st}{}^{\wedge}\right)$. The view synthesis module, which we discuss next, uses the predicted depth and egomotion quantities to reconstruct the target image from nearby source views.

Image Reconstruction Through View Synthesis

During training, a view synthesis module makes use of the predicted quantities (and the known camera intrinsics) to reconstruct the target image view from the pixel values of a nearby source image. Figure 4.2 shows how each pixel in the reconstructed image, $\mathcal{I}_{s\to t}$, is populated. First, in the projection step, the coordinates of a pixel $\mathbf{u}'$ from $\mathcal{I}_{s\to t}$ are found in $\mathcal{I}_s$. The coordinates in $\mathcal{I}_s$ are denoted as $\mathbf{u}$. This correspondence is established by

$$\mathbf{u} = \pi\left(\mathbf{T}_{st}\pi^{-1}\left(\mathbf{u}', \mathcal{D}_t(\mathbf{u}')\right)\right). \tag{4.4}$$

Here, $\pi(\cdot)$, defined in Section 2.3.2, is the pinhole camera projection model that maps a 3D point to its corresponding pixel coordinates. In the sampling step, the pixel value $\mathcal{I}_{s\to t}(\mathbf{u}')$ is populated with the value at $\mathcal{I}_s(\mathbf{u})$,

$$\mathcal{I}_{s\to t}(\mathbf{u}') = \mathcal{I}_s(\mathbf{u}). \tag{4.5}$$

Since the pixel coordinates $\mathbf{u}$ are unlikely to be exact integers, bilinear interpolation is applied to determine the pixel value by blending the four surrounding pixel values according to their relative distances from $\mathbf{u}$ (see Figure 4.2 for an illustration of this process). In practice, the reconstructed image $\mathcal{I}_{s\to t}$ is produced using a spatial transformer (Jaderberg et al., 2015), which uses differentiable grid sampling to efficiently populate all pixel locations. Target image pixels that are outside of the bounds of $\mathcal{I}_s$ are populated with zeros and are excluded from the reconstruction loss.

Finally, we discuss the key modelling assumptions used within the view synthesis module. First, we assume that there is a constant illumination source and that points within the scene have Lambertian reflectance. Second, we assume that all points in the scene are stationary (relative to a fixed, world reference frame) in order for the projection step (which considers camera motion only) to be valid. Third, we assume there are no occlusions

of points in the scene due to the camera motion. In reality, these assumptions do not hold for all pixels, and minimizing the reconstruction loss for these outlier pixels can negatively impact training.

Although the modelling assumptions above are similar to the ones made in classical VO systems, there is a key difference between how the assumptions impact the robustness of classical and learned systems. With the data-driven framework, there is a range of methods that can be employed during training to ignore or downweight outlier pixels that violate the modelling assumptions. By incorporating these techniques during training, the networks are able to learn to reject unreliable image regions and to instead rely on more stable image regions. This enables the learned SfM system to operate relatively robustly, even when some modelling assumptions are violated.

Self-Supervised Losses

After view synthesis, the main supervisory signal for the learned SfM system is a loss that compares the reconstructed image with the observed target image. This photometric reconstruction loss, in its simplest form, is the L_1 norm of the difference between these two images,

$$\mathcal{L}_{L_1}(\boldsymbol{\mathcal{I}}_{s\to t}, \boldsymbol{\mathcal{I}}_t) = |\boldsymbol{\mathcal{I}}_{s\to t} - \boldsymbol{\mathcal{I}}_t| \,. \tag{4.6}$$

Most methods additionally incorporate the *structural similarity index measure* (SSIM) (Wang et al., 2004) into the reconstruction loss. The SSIM is an alternative measure of similarity that compares the structure contained within two image patches, rather than directly comparing pixel intensities, and is consequently more robust to illumination changes between frames. Operating on two image subregions with the same size, $\mathbf{x}$ and $\mathbf{y}$, the SSIM index between them is

$$SSIM(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}, \tag{4.7}$$

where μ_i, σ_i^2 are the mean and variance of the values in $i \in \{\mathbf{x}, \mathbf{y}\}$, σ_{xy} is the covariance between the two windows, and c_1, c_2 are small values added for numerical stability. The SSIM returns a scalar quantity within $[0, 1]$, with larger values indicating a greater similarity. Therefore, the SSIM loss employs the *dissimilarity* measure,

$$\frac{1 - SSIM(x, y)}{2}. \tag{4.8}$$

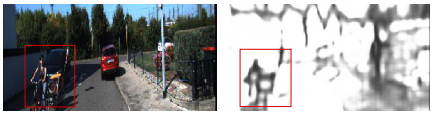
The SSIM loss, $\mathcal{L}_{\text{SSIM}}(\boldsymbol{\mathcal{I}}_1, \boldsymbol{\mathcal{I}}_2)$, is an $H \times W$ 'dissimilarity' measure between all sub-regions of size 3×3 within $\boldsymbol{\mathcal{I}}_1$ and $\boldsymbol{\mathcal{I}}_2$.[4] The combined photometric loss,

$$\mathcal{L}_{phot}(\boldsymbol{\mathcal{I}}_{s\to t}, \boldsymbol{\mathcal{I}}_t) = (1 - \alpha)\mathcal{L}_{L_1} + \alpha\mathcal{L}_{\text{SSIM}}(\boldsymbol{\mathcal{I}}_{s\to t}, \boldsymbol{\mathcal{I}}_t), \tag{4.9}$$

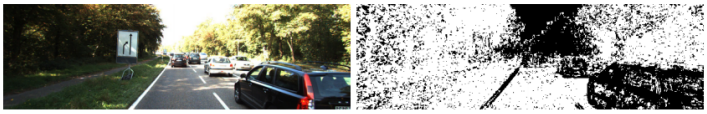
is balanced with $\alpha \in [0, 1]$, whose value determines the relative importance of the L_1 and SSIM loss terms.

Although the photometric reconstruction loss is the primary loss function used to train the learned SfM system, minimization of this loss on its own is not a well-posed problem as there are many different combinations of depth and egomotion that can (locally) minimize the error. For the system to converge in such a way that the network predictions are representative of the true scene structure and camera motion, additional con-

[4] Similar to a 2D convolution with a stride of one, computing the SSIM (or the dissimilarity measure) for all 3×3 sub-regions will produce $H - 1 \times W - 1$ terms. Reflection padding is then used to return the dimensionality to $H \times W$ in order match the dimensionality of the L_1 loss.



(a) Explainability mask examples (Zhou et al., 2017).



(b) Automasking removes the 'stationary' pixels of the vehicle moving alongside the camera (From Godard et al. (2019)).

Figure 4.3: Illustrating the various techniques for removing the effect of outlier pixels on the reconstruction loss.

straints must be imposed during training. Next, we discuss how a number of auxiliary loss terms and several modifications to the training procedure can be applied to further constrain the learning problem.

Masking Out Unreliable Pixels

Owing to the presence of dynamic objects, illumination changes between frames, and occlusions of portions of the scene (as a result of the change in camera perspective between frames), there will always be pixels that violate the modelling assumptions of view synthesis. In these circumstances, view synthesis will not be able to adequately reconstruct a subset of the target image pixels from a single (nearby) source image view, even when the true camera motion and scene depth are used. Consequently, photometric error for these 'outlier' pixel regions will be large and minimization of these errors will negatively impact training.

To reduce the negative impact of these unstable pixel regions, pixel-wise masks can be applied to the reconstruction loss to remove or downweight the outlier error values. In SfMLearner (Zhou et al., 2017), the egomotion network additionally predicts an image-wise *explainability mask*, $\mathbf{W}_t \in \mathbb{R}^{H \times W}$, the per-pixel values of which are within $[0, 1]$. The explainability mask is applied to the reconstruction loss (through element-wise multiplication) to downweight the errors for unstable pixels. The resulting loss is

$$\mathcal{L} = \mathbf{W}_t \odot \mathcal{L}_{phot}(\boldsymbol{\mathcal{I}}_{s \to t}, \boldsymbol{\mathcal{I}}_t) - \log \mathbf{W}_t. \tag{4.10}$$

The first term in this expression is the reconstruction loss (downweighted by $\mathbf{W}_t$ through element-wise multiplication) and the second term is a regularizer that encourages the explainability mask to produce values close to one—thus preventing the trivial solution of minimizing $\mathcal{L}_{phot}$ by producing a mask of zeros only. An example of the learned explainability mask output is shown in Figure 4.3a. A second masking technique called 'automasking' is introduced by Godard et al. (2019) (see Figure 4.3b). Automasking removes 'stationary' pixels (where the values do not change between frames) that are associated with a static camera or an object moving with the same velocity as the camera. In either case, minimizing the reconstruction loss for these pixels causes the depth network to create maximum depth 'holes' in the predictions. These artifacts can be prevented using automasking. The binary automask is generated based on the criterion that $\mathcal{L}_{phot}(\boldsymbol{\mathcal{I}}_s, \boldsymbol{\mathcal{I}}_t) < \mathcal{L}_{phot}(\boldsymbol{\mathcal{I}}_{s \to t}, \boldsymbol{\mathcal{I}}_t)$.

Godard et al. (2019) also introduce a *per-pixel minimum reprojection loss* that evaluates $\mathcal{L}_{phot}$ by projecting multiple source images to a given target image, and for each pixel, populating the loss with minimum value:

$$\mathcal{L}_{min}(u, v) = \underset{s \in \{t-1, t+1\}}{\operatorname{argmin}} \mathcal{L}_{phot}(\boldsymbol{\mathcal{I}}_{s \to t}, \boldsymbol{\mathcal{I}}_t)(u, v). \tag{4.11}$$

By providing a secondary view for reconstruction, outlier pixels (from one source view) can be removed from the image reconstruction and replaced with the projected pixels from the secondary source view. This technique leads to a more accurate reconstruction and is particularly effective at handling occlusion due to camera motion, since there is a high likelihood that a target pixel in frame t, if occluded in one source image, will not be occluded

in another.

Auxiliary Loss Functions

To further constrain the learning problem, a number of auxiliary self-supervised loss terms have been proposed that can be added during training alongside the reconstruction loss. These losses incorporate additional domain knowledge into the training procedure to improve the depth and egomotion estimates. The inverse depth smoothness loss (Godard et al., 2017),

$$\mathcal{L}_{\mathrm{S}} = \left| \frac{\partial}{\partial_x} \left(\boldsymbol{\mathcal{D}}_t^{-1} \right) \right| e^{-\left\| \frac{\partial}{\partial_x} \boldsymbol{\mathcal{I}}_t \right\|} + \left| \frac{\partial}{\partial_y} \left(\boldsymbol{\mathcal{D}}_t^{-1} \right) \right| e^{-\left\| \frac{\partial}{\partial_y} \boldsymbol{\mathcal{I}}_t \right\|}, \tag{4.12}$$

leverages the strong correlation between depth discontinuities and edge regions within the image by encouraging a similarity between the gradients of the inverse depth and those of the target image. The depth consistency loss, from Bian et al. (2019),

$$\mathcal{L}_{DC} = \frac{|\boldsymbol{\mathcal{D}}_{s \to t} - \boldsymbol{\mathcal{D}}_t|}{\boldsymbol{\mathcal{D}}_{s \to t} + \boldsymbol{\mathcal{D}}_t}, \tag{4.13}$$

assumes that nearby source image depth predictions, when projected into the target frame (using the predicted egomotion), should be consistent with the predicted target image depth. This loss, which conveniently produces values between 0 and 1, can also be used to assemble a *self-discovered* mask, $\mathbf{W}_{DC} = \mathbf{1} - \mathcal{L}_{DC}$. This mask identifies (and masks) pixels belonging to dynamic objects, since their motion between frames results in interframe depth inconsistency.

Scale Ambiguity

Finally, we note that the minimization of the photometric reconstruction loss does not lead to the recovery of the true (metric) scale of the depth and egomotion predictions. Instead, the depth and egomotion predictions must adopt an approximately constant scale factor that is shared between the networks across the training dataset. Typically, the networks adopt an *approximately* consistent notion of scale, but this scale factor is prone to drift (especially at test time). As discussed in other works (Bian et al., 2019; Wang et al., 2021), scale inconsistency can hinder convergence during training. To improve scale consistency, Bian et al. (2019) use the Equation (4.13) depth consistency loss to implicitly enforce a more consistent scale factor. Stereo cameras have also been used to resolve metric scale through a left-right consistency loss (Godard et al., 2017), which enforces consistency between the depth predictions from the left and right stereo images. In this case, the known stereo camera baseline provides the metric information required to recover the true, metric scale factor. The stereo consistency loss, however, is not available at test time for monocular systems and cannot benefit from online retraining. Building on these works, we develop novel approaches to resolve metric scale and to improve scale consistency.

Network Training

The depth and egomotion networks are jointly trained by minimizing a combination of the self-supervised losses that we introduced in this chapter. This loss is minimized by updating the network weights through gradient descent with a standard optimizer such as SGD or Adam (Kingma and Ba, 2014). For each minibatch of training samples, a forward pass through the network is performed, and the predictions are used for view synthesis. Then, the loss is evaluated, and the gradients are backpropagated through the (differentiable) view synthesis

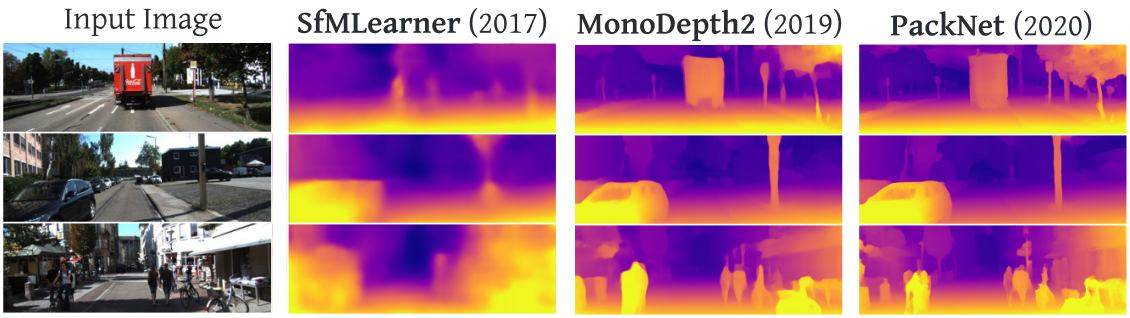


Figure 4.4: Illustration of the progression of depth network accuracy from SfMLearner (Zhou et al., 2017) to modern systems such as MonoDepth2 (Godard et al., 2019) and PackNet (Guizilini et al., 2020a).

module into the depth and egomotion networks. The network weights can either be initialized with a standard weight initialization scheme, or can be pretrained; generally, the ResNet depth encoder is initialized with publicly available weights from training on ImageNet. In most cases, individual training samples consist of a window of three temporally adjacent images to facilitate the Equation (4.11) minimum reconstruction loss calculation (where two source images are used to reconstruct the (middle) target image).

To maximize data efficiency, augmentation is used to generate additional data. Usually, augmentation involves applying random perturbations to the image brightness, contrast, and saturation. Additionally, images are flipped about their vertical axis to simulate different camera motions. Data-efficiency is further improved by minimizing the reconstruction loss in *both* directions; here, we define the forward direction as the projection of both source images to the target frame. The inverse direction projects the target image into the two source frames in order to compute the inverse reconstruction loss $\mathcal{L}_{inv} = \mathcal{L}_{phot}(\boldsymbol{\mathcal{I}}_{t\to s}, \boldsymbol{\mathcal{I}}_s)$. Finally, we note that it is popular to adopt a *multi-scale* approach (Godard et al., 2019), which involves outputting depth predictions at multiple image resolutions, and computing the reconstruction loss at each scale.[5] We find, however, in line with Bian et al. (2019), that the multiscale approach is not worth the increase in computational burden and is often detrimental to egomotion prediction accuracy.

Training/Evaluation Datasets. The learned SfM system requires a significant amount of data for training. In subsequent chapters, we train and evaluate our system on a number of visual datasets collected with autonomous driving platforms (i.e., the KITTI (Geiger et al., 2013) and Oxford RobotCar (Maddern et al., 2017) datasets), with a hand-held camera (i.e., the ScanNet (Dai et al., 2017a) dataset), and with a UAV platform (i.e., the EuRoC (Burri et al., 2016) dataset). We refer the reader to Appendix A for more information on each of these datasets, including the evaluation metrics used for benchmarking.

4.3 Motivation for the Learned SfM Framework

Considerable advances have been made to self-supervised depth and egomotion frameworks through modification of the network architectures, loss functions, and training procedures. These improvements are best demonstrated by the results in Figure 4.4 that illustrates the progression of depth-prediction quality over time. Despite these advances, we note that there has been less emphasis placed on the improvement of egomotion network accuracy. However, being data-driven and fully self-supervised, the egomotion network is a strong candidate for

[5] This is analogous to the techniques in computer vision that operate at different *pyramid levels*.

robust self-localization within modern autonomy systems. In this section, we discuss our primary motivation for investigating how the accuracy of the self-supervised egomotion network can be improved—ideally to the point where it is comparable to that of classical egomotion estimators.

Being data-driven, the learned SfM system can leverage the capacity and flexibility of modern neural networks to model the complex relationship between visual measurements and camera motion within challenging environments. Further, the self-supervised nature of the networks can be leveraged to mitigate the current limitations (i.e., generalization and accuracy, discussed in Table 1.1) of data-driven systems. The issue of generalization can be addressed by the potential for lifelong learning; these systems can continually improve over time by adapting to environmental changes. The issue of accuracy (with respect to classical systems, under nominal operating conditions) can be remedied in part by unique modifications that are possible as a result of using self-supervised losses (see Chapter 6) and in part by incorporating the learned pipeline within a larger hybrid system (see Chapter 7).

When contrasting the network-based approach to classical visual egomotion estimation approaches, there are key differences that favour the use of the learned SfM framework for self-localization. Namely, at test time, EgoNet is a *structure-free* VO estimator that is able to estimate camera motion without requiring an explicit representation of scene structure.[6] This structureless approach is beneficial when scene structure cannot be accurately recovered within visually degraded environments. While network-based approaches rely on data-driven priors to maintain accuracy in these degraded environments, classical approaches are more prone to failure as they require accurate estimates of 3D locations in every frame to estimate camera motion. Further, we advocate for the learned SfM system because, being a dense estimator, it utilizes the full range of information within the visual measurements. Unlike other dense, optimization-based methods, the learned SfM system does not require explicit initialization in order to produce its predictions.

In the coming chapters, we investigate several ways to improve egomotion network accuracy. Chapters 5 and 6 discuss our contributions for improving the baseline SfM system, while Chapter 7 describes how this system can be incorporated into a larger hybrid framework for robust visual-inertial egomotion estimation. Below, we introduce our initial proof-of-concept system for self-supervised SfM, the performance of which we improve upon in the remainder of the thesis.

4.3.1 The Self-Supervised Deep Pose Correction Network

Our initial proof-of-concept approach for self-supervised SfM is a hybrid system that builds upon the *deep pose correction* (DPC) network of Peretroukhin and Kelly (2018). In DPC-Net, a data-driven model is trained to produce corrections to a classical VO estimator in order to account for the systematic bias that results in reduced accuracy. Specifically, the classical VO estimator produces a 'large' prior and a deep neural network learns to produce smaller 'corrections' that take into account how the classical estimator degrades in adverse situations (e.g., when modelling assumptions no longer hold). DPC-Net is trained using a supervised loss function that requires ground truth pose information; for a given VO estimator, ground truth pose corrections are generated by taking the difference between the known pose change and the predicted pose change, across all image pairs within the training dataset. Minimization of the supervised loss enforces the network output to be similar to these corrections. In their experiments, the authors of DPC-Net paired the network with the classical (stereo) Libviso2 estimator (Geiger et al., 2011) and evaluated the system on the KITTI dataset. Although the authors

[6] Since the depth network is only required for view synthesis, it can be removed at test time without impacting the egomotion predictions from EgoNet. For this reason, we consider EgoNet to be a 'structure-free' VO estimator: it does not require an explicit 3D representation of scene structure to operate.

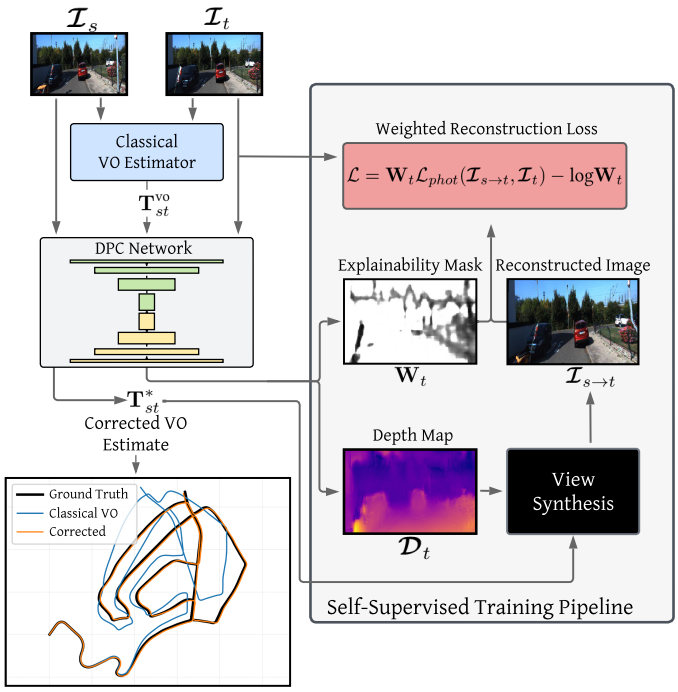


Figure 4.5: Our self-supervised deep pose correction (DPC) network regresses a pose correction to a classical VO estimator.

demonstrate a significant improvement in accuracy, the overall generalization ability (beyond the initial training distribution) of DPC-Net is limited by its reliance on supervised pose labels.

Our system, illustrated in Figure 4.5, builds upon DPC-Net by incorporating the learned SfM framework for self-supervised training of the pose corrector network. As with other self-supervised methods, our network outputs a predicted interframe pose change and a depth map—however, our predicted egomotion is initialized with an egomotion *prior* from a classical VO estimator. We compound this prior with a *correction* that is produced by our pose network. Concretely, our pose network regresses an $\mathrm{SE}(3)$ *correction*, $\mathbf{T}_{st}^{\text{corr}}$, that refines the classical VO estimate, $\mathbf{T}_{st}^{\text{vo}}$. To parameterize this correction, our network outputs an unconstrained vector from the $\mathfrak{se}(3)$ Lie algebra, $\boldsymbol{\xi}_{st}^{\text{corr}} \in \mathbb{R}^6$; applying the exponential map yields an on-manifold $\mathrm{SE}(3)$ correction,

$$\mathbf{T}_{st}^{*} = \exp\left(\boldsymbol{\xi}_{st}^{\text{corr}\wedge}\right) \mathbf{T}_{st}^{\text{vo}}. \tag{4.14}$$

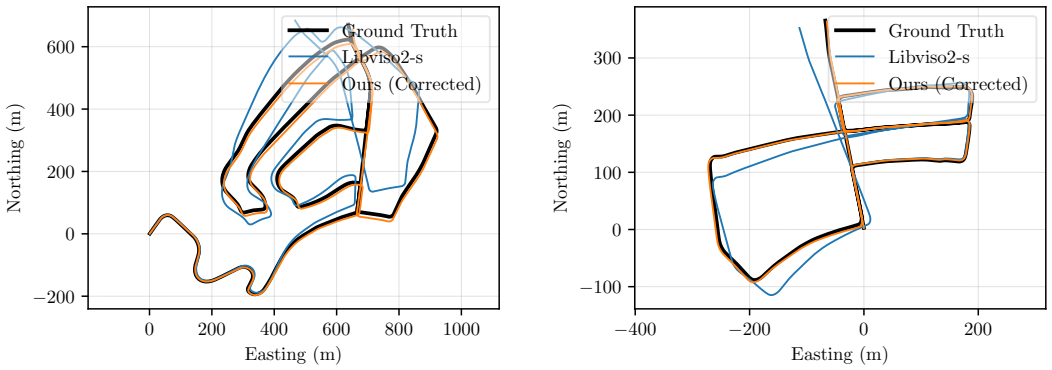
We follow the training procedure of SfMLearner (Zhou et al., 2017), discussed in Section 4.2.2, to jointly train our modified DPC-Net by minimizing the weighted photometric reconstruction loss defined in Equation (4.10). This loss requires an explainability mask to be learned in addition to the depth and egomotion predictions. Next, we discuss the primary results, but refer to Wagstaff et al. (2020) for further details about the network structure, training procedure, and experimental details.

Similar to the original DPC-Net, we pair our networks with `libviso2-s`, the stereo VO approach of Geiger et al. (2011), and train/evaluate our system on the KITTI dataset. Table 4.1 reports the errors for our corrected trajectories, while Figure 4.6 visually depicts these results and compares them to the `libviso2-s` estimates. We benchmark against the original supervised DPC-Net and also a direct, keyframe-based VO implementation based on DSO (Engel et al., 2017). There are three primary takeaways from this work:

1. **The self-supervised system outperforms the supervised variant of DPC-Net**: This is an interest-

Table 4.1: Results of correcting `libviso2-s` with our self-supervised DPC network. ATE is the average trajectory error. and the mean segment error is reported for all segments within $\{100, 200, \cdots, 800\}$ metres.

Test Sequence (Length)	Estimator	ATE		Mean Seg. Err.	
		Trans. (m)	Rot. (°)	Trans. (%)	Rot. (°/100m)
00 (3.7 km)	`libviso2-s`	53.77	13.30	2.79	1.292
	DPC-Net	15.68	3.07	1.62	0.559
	Direct VO	**12.41**	**2.45**	1.28	0.542
	Ours	14.65	3.32	1.03	**0.444**
02 (5.1 km)	`libviso2-s`	68.60	12.55	2.42	0.923
	DPC-Net	17.69	2.86	1.16	0.436
	Direct VO	16.33	3.19	1.21	0.467
	Ours	21.31	**1.91**	**0.83**	**0.373**
05 (2.2 km)	`libviso2-s`	19.68	6.30	2.31	1.135
	DPC-Net	9.82	3.57	1.34	0.562
	Direct VO	5.83	2.05	**0.69**	0.320
	Ours	**4.03**	**1.18**	0.83	**0.304**



(a) KITTI sequence 02. **(b)** KITTI sequence 05.

Figure 4.6: Corrected `libviso2-s` trajectories. We show the original `libviso2-s` estimate for comparison.

ing result, given that the supervised variant uses additional information—ground truth pose labels—for training. Potential explanations for the boost in accuracy are that: (1) the dense (per-pixel) nature of the reconstruction loss is 'richer' than the supervised pose loss; (2) incorporating domain knowledge (through the SfM framework) by formulating SfM as a multi-task learning problem is known to improve subtask accuracy (Zhang and Yang, 2021); and (3) pose labels that, to some degree, contain errors, are removed. Future work could potentially investigate how the photometric reconstruction loss, assembled from thousands of per-pixel loss terms, provides a richer gradient flow to the networks compared to a six-dimensional pose supervision loss. Further, the impact of adding noise to the pose labels could be studied to determine how imperfect ground truth impacts performance. For our purposes, we use this finding as motivation to further explore self-supervised learning techniques that employ the photometric reconstruction loss.

2. **The scale ambiguity present in the (monocular) reconstruction loss negatively impacts the translation corrections**: We find that the improvements from incorporating DPC-Net are solely due to corrections being applied to the rotation estimates. The scale ambiguity of the reconstruction loss prevents the network from learning how to improve the (already accurate) translation estimate of the stereo VO estimator. Further, our system fails to converge when paired with `libviso2-m`, the monocular variant of Geiger et al. (2011). We posit that this failure is due to scale drift present within the classical VO estimate, which our DPC-Net cannot account for. Our findings are in line with recent results (Bian et al., 2019; Wang et al., 2021) that demonstrate how removal of scale ambiguity is important for improving accuracy within learned SfM. Our contributions in Chapters 5 and 6 are motivated in part by this finding.

3. **The hybrid system outperforms both classical and end-to-end methods**: DPC-Net is an interesting case where a hybrid system, combining a classical estimator with a learned component, leads to improved accuracy compared with standalone classical/learned systems. This is a noteworthy (and perhaps unexpected) outcome, since it suggests that modelling pose corrections is easier than modelling the full relationship between sensor measurements and camera motion, even though these two tasks are somewhat equivalent (the DPC network must learn some notion of what the 'target' egomotion is in order to provide an accurate correction to the current estimate). This discovery suggests there is an interesting synergy between classical and learned models that should be further explored. Our work in Chapter 7, which investigates another hybrid system for visual-inertial egomotion estimation, is motivated in part by this finding.

In the next two chapters, we shift from the DPC framework to learning full camera-egomotion predictions. This decision is motivated by the discussion in (2) above. Although DPC-Net is advantageous in certain settings (e.g., when using stereo cameras), its performance fundamentally depends on the classical egomotion estimator it is paired with. Since monocular VO estimators are prone to scale drift and can completely break down when operating under adverse conditions, it can be challenging to produce accurate corrections with DPC-Net. Instead, we develop a standalone learned SfM system to produce accurate and scale-consistent monocular egomotion predictions. As future work, DPC-Net could be paired with this learned SfM system to produce corrections that account for systematic error present in the learned egomotion predictions.

Chapter 5

Self-Supervised Scale Recovery for the Learned SfM System

Scale ambiguity is a well-known limitation of monocular VO and SfM systems: the true (metric) scale of the depth and egomotion cannot be resolved, and the unknown scale factor is prone to drift over time. While classical techniques aim to determine the scale factor by incorporating scene information that is a priori known (e.g., known object sizes or known camera height above the ground plane) into the VO pipeline, this type of hand-engineering is challenging to tune for performance (Frost et al., 2018).

Dealing with scale ambiguity is also a prominent challenge in monocular, data-driven systems. As we noted in the previous chapter, the self-supervised loss formulation employed to train the learned SfM system is unable to resolve metric scale. A side effect of this scale ambiguity is that the depth and egomotion predictions are *inconsistent*, since the (independent) networks producing these predictions are unable to adopt a consistent notion of scale. Notably, this scale ambiguity is a source of error during training and can hinder convergence (Bian et al., 2019). In this chapter, we build on the baseline learned SfM system from Chapter 4 by introducing a *scale recovery loss* that both enforces metric scaling and improves overall scale consistency across a given dataset.

Our proposed scale recovery loss enables the learning of metric scale by ensuring that the estimated camera height (over the ground plane) is the same as the known camera height. To the best of our knowledge, this self-supervised learning-based system is the first to produce scaled (metric) depth and egomotion estimates while only requiring monocular (as opposed to stereo) images during training. To enable the use of the novel scale recovery loss, we extract the ground plane from each training image and determine the plane normal and offset (i.e., the camera height) through a least-squares technique. The scale recovery loss then forces the estimated camera height to be consistent with the known camera height. By doing so, we inject metric information during the training process, which in turn causes the depth and egomotion networks to produce metrically scaled predictions that remain properly scaled at test time. Importantly, no ground plane segmentation is required at test time, unlike existing scale recovery methods (Xue et al., 2020). In short, the main contributions of this chapter are as follows:

1. we address the monocular scale ambiguity problem in learned SfM systems by incorporating a novel loss function that enforces metrically scaled depth and egomotion estimates without requiring ground truth labels or stereo images during training;

2. as part of our scale recovery formulation, we develop a self-supervised framework for training a ground

plane segmentation network that outputs the likelihood of each pixel belonging to the ground plane;

3. we conduct a number of experiments that comprehensively demonstrate the ability of our loss functions to resolve metrically scaled depth and egomotion predictions correctly, and furthermore show that these self-supervised losses can facilitate network retraining with a small amount of data collected online.

5.1 Related Work

For monocular egomotion estimation and SfM, accurate scale recovery—that is, the process of making relative depth and egomotion predictions consistent with metric, ground truth measurements—remains an active area of research. In this section, we describe existing scale recovery methods in the literature for both classical and learning-based egomotion estimators.

5.1.1 Scale Recovery in Classical Systems

In monocular systems, scale is often initialized at the first frame (e.g., by setting the first camera translation to have a displacement of one unit), and its relative change (drift) is tracked and accounted for over time by computing the distances between observed 3D points in multiple adjacent image pairs (Scaramuzza and Fraundorfer, 2011). Monocular systems are still subject to scale drift, however, despite these attempts to maintain a constant scale factor. SLAM frameworks such as ORB-SLAM2 (Mur-Artal and Tardós, 2017) can reduce scale drift through windowed bundle adjustment or loop closure detection, but extreme scale drift may cause loop closure to fail and may lead to irreversible errors during the map-building process (Frost et al., 2016).

To recover the scale factor in a monocular VO system, a source of metric information must be provided. Commonly, this is achieved either through sensor fusion (e.g., by adding a second camera for stereo VO, an IMU, wheel odometry, or radar/lidar sensors) or through the incorporation of scene information (e.g., known object sizes or a known camera height). If the camera height is known, the scale factor can be resolved by estimating the height of the camera (relative to a ground plane), and then comparing this estimate to the a priori known camera height. Several classical methods (Song et al., 2015; Fanani et al., 2017; Zhou et al., 2019b; Wang et al., 2018b; Kitt et al., 2011) utilize this approach for scale recovery; we draw inspiration from these methods but note that they have some key limitations. Many of these algorithms (Song et al., 2015; Zhou et al., 2019b; Kitt et al., 2011) assume that the ground plane appears within a predefined image region, which is problematic when the ground plane is not visible (e.g., when the ground plane is occluded by a vehicle on the road). An alternative is to classify ground plane pixels using colour information (Lee et al., 2015), but because the hue and intensity of the ground plane pixels may change significantly depending on scene illumination and camera settings, this form of road plane detection is unreliable. Wang et al. (2018b) address this shortcoming by detecting the ground plane by fitting a model to 3D feature points. Although this technique is more robust to ground plane pixel hue and intensity changes, the ground plane (being smooth and textureless) often lacks readily identifiable features. To mitigate these difficulties, our ground plane segmentation network is trained using a geometry-based loss, which is independent of pixel intensity and external illumination. Since we use a dense set of pixels to determine the ground plane region, we expect to outperform feature-based plane detection in areas that lack identifiable features. Finally, we note that all of these scale recovery methods require the presence of a visible ground plane in *every* image at test time. Our method only requires the presence of the ground plane in the training images.

5.1.2 Scale Recovery in Data-Driven Systems

The most straightforward means of enforcing metrically scaled depth and egomotion predictions is through supervised learning, where metric information is available from ground truth depth or pose labels. However, collecting ground truth data can be time consuming, expensive, and may not always be reliable (e.g., due to GNSS errors within urban canyons). Additionally, relying on ground truth limits the ability of learning-based systems to be retrained online when ground truth is not available. Online retraining is important when deploying robots that must operate in environments which are different than the original training environment. This desire for flexibility motivates the use of self-supervised training methods.

A limitation of the standard self-supervised photometric reconstruction loss is that it can only be used to train depth and egomotion networks that produce *unscaled* predictions. Furthermore, like classical systems, the predictions are *scale inconsistent*: different inputs produce depth and egomotion predictions with a varying scale factor, since there is nothing in the loss formulation that encourages independent predictions to have the same scale. To address scale inconsistency, recently developed algorithms (Bian et al., 2019; Zhao et al., 2020) have attempted to enforce a global scale factor using a depth consistency loss. Despite producing scale-consistent estimates, this loss cannot be used to resolve metric scale.

To resolve scale in a self-supervised system, Godard et al. (2017) introduce a left-right consistency loss that uses stereo image pairs with a fixed (and known) baseline distance. However, despite being self-supervised in nature, the stereo-based loss cannot be used for retraining when only a single camera is available. We formulate a loss function that enforces metric scaling by making use of the known camera height relative to the ground plane. Although the camera height may be considered as ground truth information, this quantity often remains available at test time, which facilitates self-supervised learning and online retraining.

The work most similar to our own is DNet (Xue et al., 2020), which uses an online technique to estimate the scale factor of its learning-based depth and egomotion networks by detecting the ground plane. DNet requires the presence of a visible ground plane at test time, while we embed information about metric scale during the training procedure and thus do not require a visible ground plane at test time. This change simplifies scale recovery and makes our approach less prone to failure at test time when the ground plane is not seen or is incorrectly detected. Further, we show that incorporating scale information leads to an improvement in inter-frame scale consistency over existing techniques (Bian et al., 2019).

5.2 Methodology

We build on the baseline depth and egomotion network structures that we introduced in Chapter 4 by incorporating a ground plane segmentation network. This network is used to estimate the per-image camera height, which can be compared with the known camera height to determine scene scale. In Section 5.2.1, we introduce the depth and egomotion networks, along with the standard self-supervised loss formulation used to jointly train them. Then, in Section 5.2.2, the design of the plane segmentation network and the self-supervised loss used for training are described. Finally, Section 5.2.3 outlines our proposed scale recovery loss, which relies on the trained plane segmentation network.

5.2.1 Self-Supervised Depth and Egomotion Networks

We employ the standard self-supervised system that we introduced in Chapter 4 consisting of DepthNet, $\boldsymbol{\mathcal{D}}_t = f_{\boldsymbol{\theta}_D}(\boldsymbol{\mathcal{I}}_t)$, and EgoNet, $\boldsymbol{\xi}_{st} = f_{\boldsymbol{\theta}_E}(\boldsymbol{\mathcal{I}}_s, \boldsymbol{\mathcal{I}}_t)$, which are jointly trained through minimization of a photometric

Figure 5.1: Examples of the plane segmentation masks (top row) and scene depth predictions (bottom row) produced by our plane segmentation and depth networks, respectively. The images are from KITTI sequence 05.

reconstruction loss.[1] The complete baseline loss used to train these networks consists of the full photometric reconstruction loss $\mathcal{L}_{phot}$ (with the L_1 and SSIM losses), in addition to the inverse depth smoothness loss $\mathcal{L}_S$ and the depth consistency loss $\mathcal{L}_{DC}$. We also include a *pose consistency* loss term, $\mathcal{L}_{PC} = |\boldsymbol{\xi}_{st} + \boldsymbol{\xi}_{ts}|$ that enforces the forward and backward pose predictions to be consistent. Overall, the *per-sample* loss that we use to train the system is

$$\mathcal{L}_{base} = \frac{1}{HW} \sum_{u,v} (\lambda_{phot} \mathcal{L}_{phot} + \lambda_S \mathcal{L}_S + \lambda_{DC} \mathcal{L}_{DC}) + \frac{1}{6} \sum \lambda_{PC} \mathcal{L}_{PC}, \tag{5.1}$$

and is averaged across all training samples. Note that within $\mathcal{L}_{phot}$, we use the automasking and minimum reprojection techniques, along with the self-discovered mask, to remove and/or downweight the unstable image regions. Section 4.2.2 contains additional information about these components. Our baseline system is trained with this loss to produce *unscaled* depth and egomotion estimates. Next, we discuss how we augment this system by incorporating scale recovery into the training procedure.

5.2.2 Self-Supervised Ground Plane Segmentation

In our scale recovery approach, we compute the per-image scale factor of the depth predictions by observing the difference between the measured camera height and the known camera height. A scale factor of unity (corresponding to the heights being equal) is enforced during training by incorporating our scale recovery loss. To estimate the scale factor, we first compute the camera height over the local ground plane and compare it to the known camera height. This requires the ground plane itself to be extracted from the image. To extract the ground plane, we use our own plane segmentation network. Alternatively, the driveable road region could be detected using an existing supervised framework (Teichmann et al., 2018), but we choose to implement our own self-supervised technique in order to facilitate retraining alongside the depth and egomotion networks. *SegNet*, our plane segmentation network, takes as input an RGB image and outputs a corresponding plane segmentation mask $\mathbf{M}_t$, whose per-pixel values $\mathbf{M}_t(u, v) \in [0, 1]$ indicate the likelihood that each pixel is a ground plane 'inlier' (i.e., belongs to the ground plane).

We train SegNet on its own by minimizing a plane consistency loss, $\mathcal{L}_{plane}$. To do so, we assume that for a given image, the lower, centre region contains the ground plane only. Although this is a limiting assumption in general, it only applies to the training data, where we can be reasonably confident that the region consistently represents the ground plane. After computing the normal vector $\tilde{\mathbf{n}}_t$ and offset (i.e., the per-image camera height),

[1] As a reminder, EgoNet regresses the Lie algebra vector of the inter-frame pose change, and the exponential map is used to produce the SE(3) transform, through $\mathbf{T}_{st} = \exp(\boldsymbol{\xi}_{st}{}^{\wedge})$.

h_t, of this specified patch of ground, we train our plane segmentation network to minimize $\mathcal{L}_{plane}$,

$$\mathcal{L}_{plane}(u,v) = \lambda_{plane}\left(\mathbf{M}_t(u,v)\left|h_t - \mathbf{p}_t(u,v)^\top \tilde{\mathbf{n}}_t\right|\right) - \lambda_{reg}\log \mathbf{M}_t(u,v), \tag{5.2}$$

over all pixels and images in the training dataset. To minimize the first term, SegNet learns to produce small values (i.e., a low confidence prediction) for pixels whose 3D coordinates $\mathbf{p}_t(\mathbf{u})$ do not lie on the ground plane. Since a trivial solution exists for this first term (outputting zero for all pixels in the image), the second term—a cross entropy regularization loss—enforces the SegNet predictions to be close to unity. The overall loss is minimized by training the network to accurately predict 'inlier' plane pixels with high confidence while downweighting all other pixels. The two loss terms are balanced by the scalar weights λ_{plane} and λ_{reg}.

In Equation (5.2), the per-image camera height h_t is computed from the predefined ground plane region using a plane-fitting procedure. For the N ground plane pixels (in the target image), the 3D coordinates $\mathbf{p}_t(\mathbf{u})$ (expressed within $\underrightarrow{\mathcal{F}}_{c_t}$) are computed as

$$\mathbf{p}_t(\mathbf{u}) = \pi^{-1}\left(\mathbf{u}, \boldsymbol{\mathcal{D}}_t(\mathbf{u})\right) = \boldsymbol{\mathcal{D}}_t(\mathbf{u})\begin{bmatrix}\frac{u-c_u}{f_u} & \frac{v-c_v}{f_v} & 1\end{bmatrix}^\top, \tag{5.3}$$

which is the inverse pinhole projection model (Equation (2.76)) that employs a pre-trained depth network.[2] The N 3D coordinates are stacked in $\mathbf{P}_t = \begin{bmatrix}\mathbf{p}_t(\mathbf{u}_0) & \mathbf{p}_t(\mathbf{u}_1) & \cdots & \mathbf{p}_t(\mathbf{u}_N)\end{bmatrix} \in \mathbb{R}^{3\times N}$, and the ground plane normal vector $\tilde{\mathbf{n}}_t$ is found by solving $\mathbf{P}_t^\top \mathbf{n}_t = \mathbf{1}_{N\times 1}$ for $\mathbf{n}_t$ using the Moore-Penrose pseudo-inverse. The unit normal to the plane is

$$\tilde{\mathbf{n}}_t = \frac{\mathbf{n}_t}{\|\mathbf{n}_t\|}. \tag{5.4}$$

The estimated camera offsets (i.e., the camera height relative to the plane) are then $\mathbf{h}_t = \mathbf{P}_t^\top \tilde{\mathbf{n}}_t$; we take the mean value h_t as our camera height estimate.

We provide further details about the training procedure for SegNet in Section 5.3.2. Figure 5.1 shows several examples of the trained plane segmentation network outputs. We note that our approach, by assuming local planarity, will only learn to identify the portion of the ground that is approximately planar. We expect the network to downweight road plane pixels in images where this assumption does not hold.

5.2.3 Scale Recovery Loss Formulation

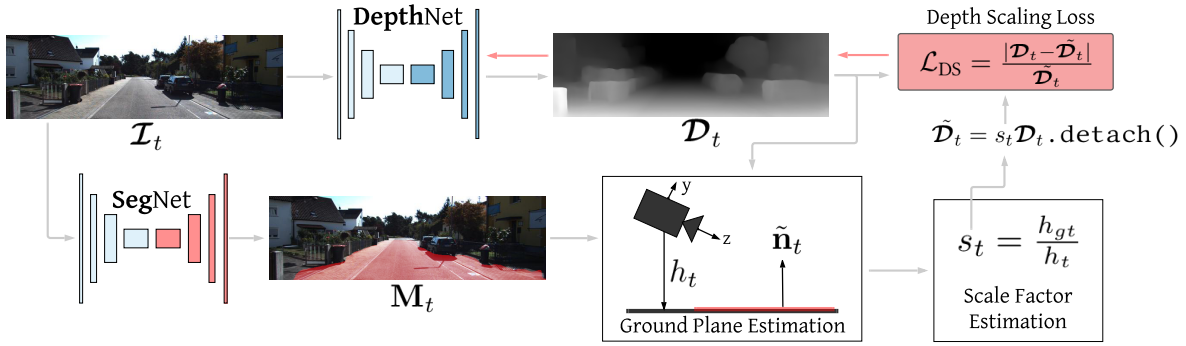
With SegNet trained to produce a per-pixel ground plane segmentation mask, the extracted ground plane can be used to estimate the per-image camera height h_t. A scale recovery loss (employed alongside our other self-supervised losses) can then enforce this camera height estimate to be similar to the known camera height, h_{gt}:

$$\mathcal{L}_{cam} = |h_t - h_{gt}|. \tag{5.5}$$

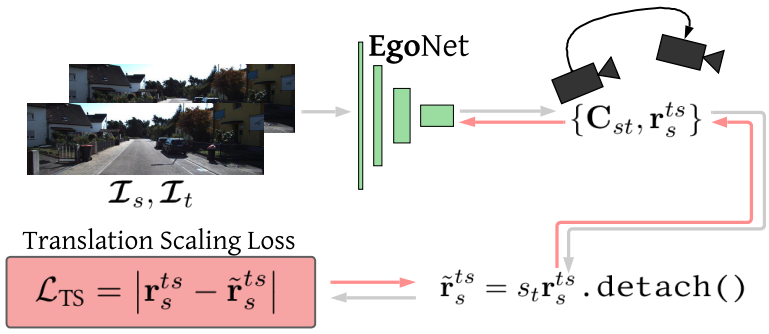
Given the plane segmentation mask $\mathbf{M}_t$, the estimated camera height can be determined through weighted least squares in a two-stage process. In the first stage, the (non-unit norm) ground plane normal direction $\mathbf{n}_t$ is found by minimizing the least squares loss,

$$L_t = \frac{1}{2}(\mathbf{P}_t^\top \mathbf{n}_t - \mathbf{1}_{HW\times 1})^\top \mathbf{M}_t^{-1}(\mathbf{P}_t^\top \mathbf{n}_t - \mathbf{1}_{HW\times 1}), \tag{5.6}$$

[2] The pre-trained depth network that we use is unscaled: we use the networks and losses from Section 5.2.1 and follow the same training procedure outlined in Section 5.3.2.



(a) Diagram explaining the depth scaling loss formulation. SegNet outputs a dense (per-pixel) mask that is used to compute the scale factor of the current depth prediction.



(b) Diagram explaining the translation scaling loss formulation.

Figure 5.2: Illustration of our novel scale recovery loss formulation. For each training image, the scale factor is computed by comparing the estimated camera height to the known camera height. By enforcing the scale factor to converge to unity during training (through our proposed scale recovery loss), the network predictions become metrically scaled.

where $\mathbf{P}_t \in \mathbb{R}^{3\times HW}$ are the stacked 3D coordinates for every pixel in the image, and $\mathbf{M}_t^{-1} \in \mathbb{R}^{HW\times HW}$ is a diagonalized matrix of plane segmentation network outputs ($\mathbf{M}_t$). The least squares solution is

$$\mathbf{n}_t = (\mathbf{P}_t\mathbf{M}_t^{-1}\mathbf{P}_t^\top)^{-1}(\mathbf{P}_t\mathbf{M}_t^{-1}\mathbf{1}_{HW\times 1}). \tag{5.7}$$

The result is normalized to produce $\tilde{\mathbf{n}}_t$. In the second stage, the estimated camera height is determined by taking the weighted average of the offset (relative to the plane) of all 3D coordinates:

$$h_t = \frac{1}{\sum_{u,v}\mathbf{M}_t(u,v)}\sum_{u,v}\mathbf{M}_t(u,v)\,\mathbf{p}_t(u,v)^\top\tilde{\mathbf{n}}_t. \tag{5.8}$$

Importantly, since a weighted least squares approach is used, Equation (5.5) becomes differentiable with respect to the depth predictions of *every* pixel in the image (the current depth prediction is used to construct the 3D coordinates, $\mathbf{p}_t$, for each pixel). This link allows for our depth network weights to be updated through gradient descent by minimizing Equation (5.5), which enforces a metric scaling of the depth predictions. We found, however, that there are issues that make Equation (5.5) unsuitable in practice; namely, because the loss is primarily a function of the ground plane pixels (since predictions for off-plane pixels are generally close to zero), the ability of the depth network to properly resolve scale over the whole scene is limited. Instead of scaling all of the depth predictions, the ground plane depth would erroneously 'sink' due to scaling, while other components of the scene remained unchanged.

To avoid the problem above, we propose an alternative loss that enforces metric depth (i.e., a scale factor of unity) by affecting all image pixels equally. Rather than directly comparing h_t to h_{gt}, as in Equation (5.5), we can compute an image-specific scale factor $s_t = \frac{h_{gt}}{h_t}$, and generate per-pixel 'depth scaling' targets,

$$\tilde{\boldsymbol{\mathcal{D}}}_t(u,v) = s_t\boldsymbol{\mathcal{D}}_t(u,v), \tag{5.9}$$

which can be directly applied in a depth scaling loss,

$$\mathcal{L}_{\text{DS}}(u,v) = \frac{|\boldsymbol{\mathcal{D}}_t(u,v) - \tilde{\boldsymbol{\mathcal{D}}}_t(u,v)|}{\tilde{\boldsymbol{\mathcal{D}}}_t(u,v)}. \tag{5.10}$$

To enforce proper depth rescaling, all gradients associated with the target depth $\tilde{\boldsymbol{\mathcal{D}}}_t$ are removed (e.g., through $\tilde{\boldsymbol{\mathcal{D}}}_t$`.detach()` in PyTorch); this forces the network to update *all* pixel depths, instead of only updating the ground plane pixels. The denominator in Equation (5.10) normalizes the per-pixel depth values to prevent large depths from dominating the loss function. Figure 5.2a illustrates how our depth scaling loss is applied.

We use the same technique (see Figure 5.2b) to define a 'translation scaling' loss,

$$\mathcal{L}_{\text{TS}} = \left|\mathbf{r}_s^{ts} - \tilde{\mathbf{r}}_s^{ts}\right|, \tag{5.11}$$

where $\tilde{\mathbf{r}}_s^{ts} = (s_t\mathbf{r}_s^{ts})$`.detach()`. We find that applying both scaling loss terms improves stability during training and causes the learned scale to converge to unity more quickly. By combining our scale recovery loss with the baseline loss, our (per-sample) overall loss becomes

$$\mathcal{L} = \mathcal{L}_{\text{base}} + \frac{\lambda_{\text{DS}}}{HW}\sum_{u,v}\mathcal{L}_{\text{DS}}(u,v) + \frac{\lambda_{\text{TS}}}{3}\sum\mathcal{L}_{\text{TS}}. \tag{5.12}$$

By incorporating these scale recovery loss terms (balanced by λ_{DS} and λ_{TS}), the scale factor will converge towards unity while the original loss terms are minimized.

5.3 Experiments

To elucidate the value of our approach, we provide details of our experiments below. First, we discuss the training and validation datasets and the relevant evaluation metrics. Then, we describe our training procedure. We follow with extensive experimental results on the KITTI and Oxford RobotCar datasets. Specifically, we focus on experiments that showcase the use of our proposed losses for scale recovery.

5.3.1 Datasets and Evaluation Metrics

KITTI Dataset. For the KITTI dataset, we downsize the images to 192×640. For depth evaluation, we follow the Eigen train/test split and report the standard metrics discussed in Appendix A. For odometry evaluation, we use sequences `00`, `02`, `06-08`, `11`, `13-16`, `19` and leave out `05` for validation, and `09-10` for testing. We report the average translational and rotational errors ($t_{err}(\%)$, $r_{err}(^o/100\text{ m})$) over possible sub-sequences of length (100, 200, ..., 800) metres.

Oxford RobotCar Dataset. We use the Oxford dataset to pretrain DepthNet and EgoNet before the KITTI experiments. We downsize the images to 192×640 and use the training/validation split from Appendix A.

5.3.2 Implementation Details

We implement our networks in PyTorch (Paszke et al., 2017), using the network structures presented in Appendix B. The networks are trained for 25 epochs on the KITTI odometry dataset using the Adam optimizer with a minibatch size of six and a learning rate of 1×10^{-4} that is reduced by half after every four epochs. Our network inputs are whitened using the ImageNet (Deng et al., 2009) statistics. During training all images are augmented through random horizontal flipping, and random transformations to the brightness, contrast, saturation, and hue within the ranges ± 0.2, ± 0.2, ± 0.2, ± 0.1 respectively (the same modification is made to all the target and source images within a training sample).

SegNet Training. Prior to implementing our scale recovery losses, we train SegNet to accurately identify the ground plane regions by minimizing the Equation (5.2) loss. The SegNet training procedure requires accurate depth predictions to extract 3D coordinates for each pixel in the image. Therefore, we initially train an *unscaled* depth and egomotion network (with the Equation (5.1) baseline, unscaled loss function), and use the unscaled depth predictions from this network to train SegNet. The hyperparameters used during SegNet training are $\lambda_{plane} = 25$ and $\lambda_{reg} = 0.05$.

Scale Recovery Training. Next, we employ the trained SegNet (with frozen weights) within the scale recovery losses to train scale-aware variants of DepthNet and EgoNet. Starting with (unscaled) networks pretrained on the Oxford Robotcar dataset, we minimize the Equation (5.12) loss on the KITTI training sequences. We initialize our networks by training for one epoch with the unscaled (baseline) loss, since our scale recovery loss requires reasonable depth estimates to estimate the scale factor. After one epoch, we incorporate the scale recovery loss terms to begin resolving metric scale. We set $\alpha = 0.85, \lambda_{phot} = 1, \lambda_S = 0.05, \lambda_{DC} = 0.14, \lambda_{PC} =$

$5, \lambda_{DS} = 0.02$, $\lambda_{TS} = 6$, $d_{min} = 1.8$, $d_{max} = 60$, with the ground truth camera height $h_{gt} = 1.70$ m.[3] Figure 5.3 illustrates the convergence of the scale factor during training; the scale factor converges within 500 minibatch iterations. Over time, as illustrated by the final epoch in the training plot, the scale factor becomes more consistent, and is generally very close to unity.

Loss Type	Mean Scale Factor (Std. Dev.)		
	Seq. 05 (val.)	Seq. 09 (test)	Seq. 10 (test)
Supervised	0.98 (0.04)	1.00 (0.04)	1.01 (0.05)
Stereo Consist.	1.01 (0.03)	1.01 (0.05)	1.01 (0.05)
$\mathcal{L}_{TS} + \mathcal{L}_{DS}$	1.00 (0.04)	1.00 (0.03)	1.01 (0.04)

Table 5.1: The average scale factor across the held-out sequences for three separate recovery methods. A more accurate scale factor is closer to unity.

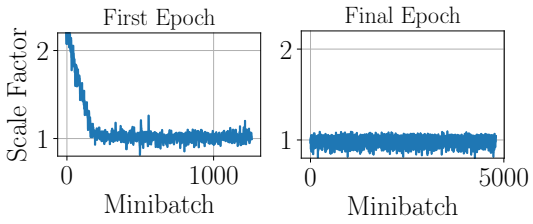


Figure 5.3: The scale convergence during the first and final training epochs while using our proposed scale recovery losses.

5.3.3 Experimental Results

We verify that our scale recovery method produces depth and pose estimates that are metrically scaled through four experiments. First, we demonstrate that the accuracy of our scale recovery method is comparable with alternative techniques that require stereo images or ground truth. Then, we show that DepthNet, being scale aware, does not require scale alignment with ground truth to produce accurate depth predictions. In our next experiment, we show that our scale recovery loss, by promoting scale consistency during training, is able to improve the overall egomotion estimation accuracy compared with online scale recovery methods such as DNet (Xue et al., 2020). Finally, we demonstrate how our loss formulation is well suited for online retraining to improve accuracy in new environments.

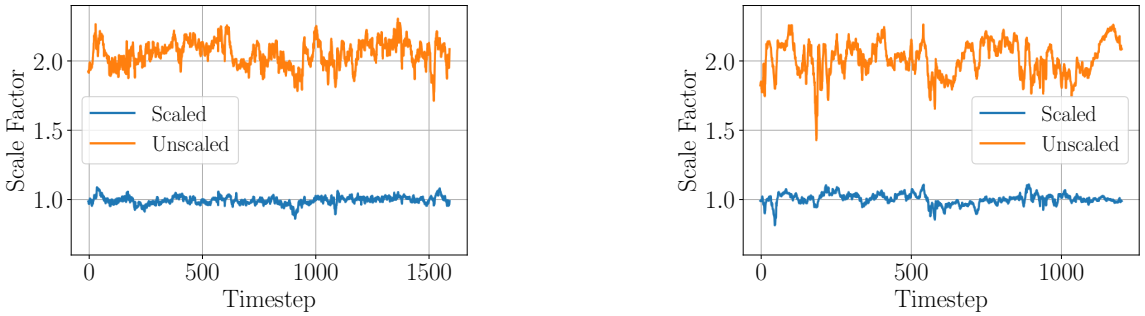
Scale Factor Evaluation

We compare our method with two existing loss functions that are used to resolve scale: a pose supervision loss and a (stereo image) left-right consistency loss (Godard et al., 2017). To implement these two techniques, we directly replace our scale recovery loss with the alternate loss function and train the depth and egomotion networks from scratch. No changes are made to the training procedure[4] or the network structures for this experiment, other than balancing the additional loss term with the existing loss terms by appropriately tuning its weighting factor.

To compare these three scale-resolving approaches, we estimate the scale factor of their depth predictions by extracting the ground plane using our plane segmentation network, and then computing the camera height with Equation (5.8). We compare the estimated camera height to the known camera height to determine the scale factor $s_t = \frac{h_{gt}}{h_t}$ for every image frame within the test and validation sequences. We report the per-sequence mean scale factor in Table 5.1. Comparing our scale recovery technique with the two alternate methods, the difference in scale factor is negligible. However, the alternative approaches require stereo images or ground truth information, while our algorithm requires knowledge of the camera height only. Figure 5.4 illustrates the estimated scale factor determined by our method and compares it with our baseline unscaled model. Lastly, we

[3] For $\mathcal{L}_{TS}$, we found that gradually increasing λ_{TS} through $\lambda_{TS} = 0.6(1 + min(2T, 9))$, with T as the current training epoch, prevented the depth network weights from diverging early in the training process.

[4] For the pose supervision method, we omit the odometry sequences (11-21) from training because no pose labels are available for these sequences.



(a) Sequence 09. **(b)** Sequence 10.

Figure 5.4: Scale factor estimates for the KITTI test sequences. In addition to resolving the metric scale factor, the variance of the scale across the sequences (i.e., the inter-frame scale consistency) is improved with our proposed losses.

Table 5.2: Ablation study demonstrating the effect of our scale recovery loss terms on the average scale factor observed across the validation and test sequences. Inclusion of both proposed loss terms is the most effective for scale recovery.

Loss Type	**Mean Scale Factor (Std. Dev.)**		
	Seq. 05 (val.)	Seq. 09 (test)	Seq. 10 (test)
Unscaled	1.99 (0.196)	2.05 (0.09)	2.02 (0.12)
$\mathcal{L}_{TS}$	1.02 (0.07)	1.05 (0.04)	1.04 (0.05)
$\mathcal{L}_{DS}$	1.02 (0.06)	1.05 (0.04)	1.04 (0.06)
$\mathcal{L}_{TS} + \mathcal{L}_{DS}$	**1.00 (0.04)**	**1.00 (0.03)**	**1.01 (0.04)**

include an ablation study (Table 5.2), which indicates that combining $\mathcal{L}_{\text{DS}}$ and $\mathcal{L}_{\text{TS}}$ results in a scale factor that is the closest to unity.

In addition to resolving metric scale, we observe how the inclusion of our scale recovery losses improves the overall *interframe scale consistency*. The ablation study detailed in Table 5.2 shows how the standard deviation of the scale factor is reduced when either scale recovery loss is applied, and further decreases with the inclusion of both losses. This improvement in scale consistency is also illustrated in Figure 5.4. We consider the improvement in scale consistency to be a noteworthy result, since the baseline system already employs the depth consistency loss, which Bian et al. (2019) previously proposed to improve inter-frame scale consistency. We attribute this improvement as being due to the explicit inclusion of metric information through the scale recovery losses.

Depth Evaluation

To benchmark the accuracy of the depth predictions, we train our system with the Eigen training/validation/test split of the KITTI dataset. Here, we use the standard preprocessing of Zhou et al. (2017) to remove stationary images, and train our system using the same procedure from Section 5.3.2. Table 5.3 depicts the depth accuracy results on the test split. We include three versions of our network:

1. our (unscaled) baseline network whose predictions have been rescaled using ground truth depth; the per-image scale factor is $s_t = \frac{\text{median}(\boldsymbol{\mathcal{D}}_{t,gt})}{\text{median}(\boldsymbol{\mathcal{D}}_{t,pred})}$,

2. the same network whose predictions have been rescaled using the known camera height (the per-image scale factor is $s_t = \frac{h_{gt}}{\hat{h}_t}$), and

Table 5.3: Monocular depth prediction results on the Eigen test split. The scaling method is either a per-image scale factor correction using the ground truth depth (GT) or one that only uses knowledge of the camera height over the ground plane (Cam. Height).

Method	Scaling Method	Error ↓				Accuracy ↑		
		Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Bian et al. (2019)	GT[1]	0.137	1.089	5.439	0.217	0.830	0.942	0.975
Zhao et al. (2020)	GT	0.148	1.091	5.536	0.209	0.802	0.934	0.976
DNet	GT	**0.113**	**0.864**	**4.812**	**0.191**	**0.877**	**0.960**	**0.981**
Ours (w/o $\mathcal{L}_{TS} + \mathcal{L}_{DS}$)	GT	0.130	1.313	5.254	0.206	0.857	0.955	0.979
DNet	Cam. Height	0.118	0.925	4.918	0.199	0.862	0.953	0.979
Ours (w/o $\mathcal{L}_{TS} + \mathcal{L}_{DS}$)	Cam. Height	0.155	1.657	5.615	0.236	0.809	0.924	0.959
Ours	None	0.123	0.996	5.253	0.213	0.840	0.947	0.978

[1] A constant scale factor correction is applied to all frames in the sequence, as opposed to the per-frame "GT" scaling that other methods use for evaluation.

3. our scale-aware network trained with our scale recovery loss, whose scaled predictions do not require any online rescaling.

Comparing these methods in Table 5.3, we see that the accuracy of our proposed method is competitive with existing approaches, despite not requiring any form of scaling at test time. Furthermore, we note that incorporating our scale recovery loss at training time results in more accurate depth predictions than our two other baselines, *unscaled* network predictions that have been rescaled using the known camera height or ground truth. We posit that this increase in accuracy is a result of the improved inter-frame depth consistency that our scale recovery losses provide (as discussed in Section 5.3.3).

Visual Odometry Evaluation

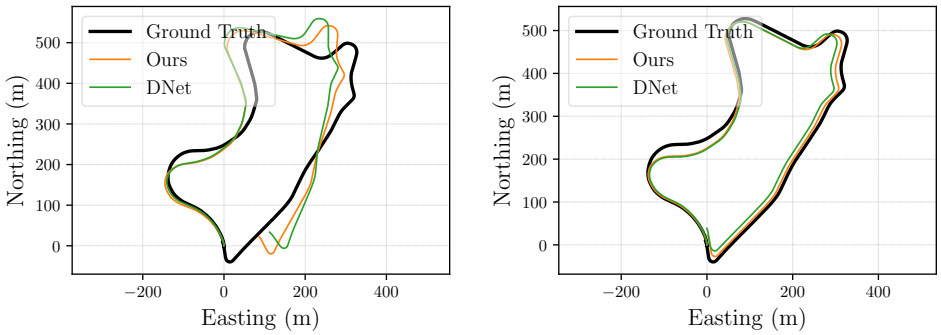
We evaluate the egomotion (VO) accuracy of our scale-aware system on the KITTI validation sequences and present the results in Table 5.4 and Figure 5.5. In Table 5.4, we benchmark the VO accuracy of our method by comparing it with several (monocular) alternatives. First, to illustrate the problem of scale drift in classical estimators, we benchmark against ORB-SLAM2 (Mur-Artal and Tardós, 2017) (without loop closure). Second, we benchmark against two existing self-supervised methods (Bian et al., 2019; Zhao et al., 2020) that promote scale consistency using their proposed depth consistency losses. Third, we compare our system against DNet (Xue et al., 2020). Since the authors of DNet do not report any VO results, we adopt their open-source scale recovery technique and use it to rescale our baseline, unscaled network. We follow the same procedure as DNet to estimate the scale factor online: for a given image, we compute the median camera height using the ground plane pixels (and their associated depth predictions) and determine the scale factor as $s_t = \frac{h_{gt}}{h_t}$. The per-image scale factor is then used to rescale the egomotion estimates from our baseline network.

Table 5.4 lists all mean translation and rotation segment errors for these approaches. We observe that ORB-SLAM2, despite producing an accurate result for sequence 10, suffers from scale drift in sequence 09. When comparing the alternative self-supervised methods with ours, we see that our system has a similar accuracy without requiring any form of ground truth at test time to produce metrically scaled predictions. Lastly, when comparing our system with the baseline that uses the online rescaling method from DNet, we note that our system produces a more accurate translation prediction. This result is verified in the final two rows of Table 5.4, where the predicted orientation is replaced with ground truth to isolate the translation prediction error. We

Table 5.4: Visual odometry results on the KITTI dataset. Our approach produces metrically scaled trajectory estimates without requiring any test-time scaling.

Method	Scaling Method	Seq. 09		Seq. 10	
		t_{err} (%)	r_{err} (°/100m)	t_{err} (%)	r_{err} (°/100m)
With Predicted Orientation					
ORB-SLAM2	GT[1]	15.30	**0.26**	**3.68**	**0.48**
Bian et al. (2019)	GT[1]	8.24	2.19	10.7	4.58
Zhao et al. (2020)	GT[1]	8.13	2.64	9.74	3.58
Ours (w/o $\mathcal{L}_{TS} + \mathcal{L}_{DS}$)	DNet	7.23	1.91	13.98	4.07
Ours	None	**5.93**	1.67	10.54	4.03
With Ground Truth Orientation					
Ours (w/o $\mathcal{L}_{TS} + \mathcal{L}_{DS}$)	DNet	5.14	–	9.67	–
Ours	None	**3.63**	–	**6.14**	–

[1] A constant scale factor correction is applied to all frames in the sequence.



(a) With predicted orientation.

(b) With ground truth orientation.

Figure 5.5: Top-down view of sequence 09, comparing the accuracy of our method with the DNet rescaling method with the predicted orientation (left) and ground truth orientation (right).

hypothesize that this improvement is a result of the incorporation of metric information at training time.

Online Retraining Evaluation

With self-supervised systems, data collected within new environments can be used to update the model parameters, allowing the networks to adapt to changing surroundings. Our final experiment demonstrates how our self-supervised loss formulation accounts for out-of-distribution data through online network retraining to produce metrically scaled predictions. We examine domain adaptation by first training our system on the Oxford RobotCar dataset and then evaluating VO accuracy on the KITTI dataset.

We first train our depth and egomotion networks on a subset of the Oxford RobotCar dataset using the same training procedure, network structure, and hyperparameters as for the KITTI experiments (see Section 5.3.2). Following this, we retrain the networks for a single epoch with KITTI data (using the KITTI training sequences), without changing any parameters (with the exception of setting the camera height back to 1.70 metres). To gauge the effectiveness of our scale recovery loss, retraining is carried out twice, once with the scale recovery loss and once without.

Table 5.5 lists the domain adaptation results. As expected, due to the difference in the images between datasets (e.g., camera parameters and height, as well as significant changes in scene structure/illumination), our depth and egomotion networks trained on the RobotCar dataset do not perform well on the KITTI dataset.

Table 5.5: Results from our retraining experiment. A network pre-trained on the Oxford RobotCar dataset is evaluated on the KITTI dataset. By retraining on KITTI with our scale recovery loss, VO accuracy is significantly improved.

Method	**Seq.** 09		**Seq.** 10	
	t_{err} (%)	r_{err} (°/100 m)	t_{err} (%)	r_{err} (°/100 m)
Original	34.51	9.23	36.38	12.01
Retrained (no $\mathcal{L}_{\text{TS}} + \mathcal{L}_{\text{DS}}$)	34.99	2.25	32.53	**3.45**
Retrained (no $\mathcal{L}_{\text{TS}} + \mathcal{L}_{\text{DS}}$) with DNet Rescaling	7.73	2.25	12.48	**3.45**
Retrained (with $\mathcal{L}_{\text{TS}} + \mathcal{L}_{\text{DS}}$)	**6.93**	**1.70**	**8.82**	3.54

By retraining the models for a single epoch on KITTI using the Equation (5.1) baseline (unscaled) loss, the overall accuracy improves, but the scale factor does not converge to unity. In contrast, when retraining with Equation (5.12), the network adapts to the KITTI environment *and* produces metric predictions.

5.4 Summary and Future Work

In this chapter, we described a novel scale recovery loss that encourages learned SfM predictions to have a uniform, metric scale factor. In contrast to alternative approaches (e.g., pose supervision or stereo consistency losses), our method requires only a stream of monocular images and a known camera height at training time. Notably, our networks can be retrained online, which significantly improves egomotion predictions for out-of-distribution images. Additionally, our loss enforces inter-frame depth consistency during training, boosting overall egomotion estimation accuracy compared to a similar method that only recovers scale at test time.

As future work, we may investigate how our scale recovery framework can take into account changes in the camera height. While we demonstrated that our approach is successful for a wheeled platform (i.e., a car, where the camera height is approximately constant), other wheeled systems may experience larger fluctuations in camera height that might corrupt scale recovery. In these circumstances, an uncertainty-based framework could be used to downweight the scale recovery loss for training samples where the camera height fluctuates. For aerial platforms, where motion is no longer constrained to a plane, altitude information (i.e., from a barometer or altimeter) could replace the fixed camera height in our scale recovery framework to recover scale with our proposed losses.

Chapter 6

Tightly Coupled Networks for Scale-Consistent SfM

In the learned SfM system, the self-supervised photometric reconstruction loss intimately ties depth and egomotion estimation together through view synthesis. To minimize this loss, the depth and egomotion networks, traditionally parameterized as separate (independent) network structures, must implicitly learn to be consistent with each other during training. In particular, the networks must learn a mutually consistent notion of scale. In the previous chapter, we discussed how auxiliary losses can be incorporated to boost scale consistency by incorporating metric information into the training procedure. Although inter-network consistency is implicitly encouraged during training via the addition of these losses, inconsistency in the network predictions may limit convergence during training and degrade overall system accuracy (Wang et al., 2021, 2018a). To mitigate the issues associated with scale inconsistency, it is possible to couple (i.e., link) the depth and egomotion networks in such a way that there is an explicit exchange of information that, for example, allows for scale details to be passed from one network to the other.

On the other hand, it is also possible to *over-couple* the networks, resulting in convergence to a local minimum where neither the depth nor egomotion predictions are representative of reality. Indeed, Godard et al. (2019) note that learning depth and egomotion with a joint network (i.e., through weight sharing between DepthNet and EgoNet) degrades the overall system performance; we have observed the same phenomenon within our own experiments. It is of interest, therefore, to establish a coupling method that sufficiently facilitates information sharing, while preserving the (partial) independence of the network structures.

In this chapter, we investigate and improve upon existing methods to achieve an appropriate degree of network coupling. First, we survey the field and categorize the approaches to coupling described in the literature (Figure 6.1 and table 6.1). We find that most systems rely solely on *indirect coupling* of depth and egomotion via the self-supervised reconstruction loss; others incorporate *direct coupling* by treating one prediction as a function of the other (Wang et al., 2019; Li et al., 2020); and lastly, recent methods (Nabavi et al., 2020; Gu et al., 2021) incorporate a form of direct coupling—that we call *feedback coupling*—to iteratively refine predictions based on successive forward passes through the networks. Building on this taxonomy, we present a novel network structure that ensures the depth and egomotion network predictions are *tightly coupled* at both training and test time by leveraging all three coupling strategies together.

Our approach uses iterative view synthesis (Nabavi et al., 2020) to recursively update the egomotion network input with the most recent synthesized view, which makes the egomotion prediction a function of depth. Ad-

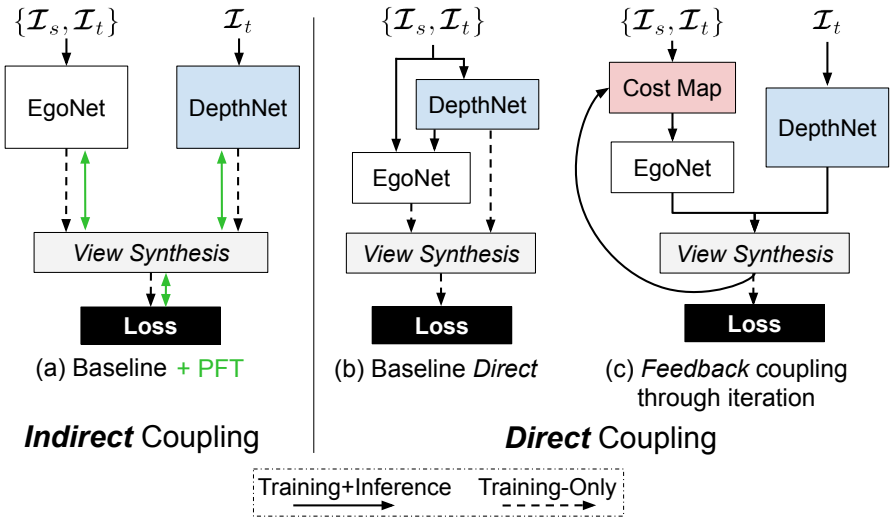


Figure 6.1: Three primary forms of coupling that exist between depth (DepthNet) and egomotion (EgoNet) networks. We show that feedback coupling, which in this case treats the egomotion network input as an iteratively updated cost map (i.e., an iteratively updated photo- or feature-metric input that encodes prediction error), is an effective form of coupling. In all cases, the loss is the photometric difference between a target image ($\mathcal{I}_t$) and a virtual image synthesized by warping a nearby source view ($\mathcal{I}_s$). Test-time optimization through parameter fine-tuning (PFT), illustrated with green arrows, can be applied to direct coupling methods as well (we omit this from the diagram for simplicity).

ditionally, we incorporate test-time optimization (Chen et al., 2019b) for parameter fine-tuning (PFT). Through extensive experiments, we demonstrate that our unique coupling strategy promotes scale consistency between the depth and egomotion predictions, improves generalization, and leads to state-of-the-art accuracy on indoor and outdoor depth and egomotion evaluation benchmarks. In short, the main contributions of this chapter are as follows:

1. we introduce a new taxonomy for categorizing the large variety of learned SfM systems by the degree of *coupling* that exists between their depth and egomotion networks;

2. we develop a novel *tightly coupled* SfM system whose depth and egomotion networks are carefully linked in order to properly share information at training and test time;

3. we present extensive experimental results demonstrating how our proposed system improves in accuracy and generalization compared with a baseline *decoupled* network structure;

4. we elucidate how proper network coupling allows for a mutually consistent scale factor to be shared between the depth and egomotion networks, facilitating inter-network scale consistency.

6.1 Related Work

In what follows, we attempt to broadly categorize the degree of coupling between the depth and egomotion networks in a number of common learned SfM systems. Table 6.1 illustrates how the depth and egomotion networks can be linked using *indirect*, *direct*, and *feedback* coupling, which we describe in detail in the coming sections.

Table 6.1: Recent SfM methods that use the coupling strategies shown in Figure 6.1 between the depth (D) and egomotion (E) networks. Our proposed tightly coupled network structure is the only method that uses all three forms of coupling.

Method	Inference-Time Coupling Strategy		
	Indirect	Direct	Feedback
Baseline: Zhou et al. (2017); Godard et al. (2019); Guizilini et al. (2020a)	–	–	–
PFT: Zhang et al. (2021); Chen et al. (2019b); McCraith et al. (2020) Shu et al. (2020); Kuznietsov et al. (2021)	✓	–	–
Zou et al. (2020)	–	D→E	–
Ambrus et al. (2020)	–	D→E	–
Li et al. (2020)	✓	D→E	–
Wang et al. (2020)	–	E→D	–
ManyDepth (Watson et al., 2021)	✓	E→D	–
Nabavi et al. (2020)	–	D→E	✓
DRO (Gu et al., 2021)	–	D⇄E	✓
Ours	✓	D→E	✓

6.1.1 Indirect Coupling

The majority of self-supervised methods (Zhou et al., 2017; Godard et al., 2019; Guizilini et al., 2020a; Li et al., 2018; Bian et al., 2019; Mahjourian et al., 2018; Vijayanarasimhan et al., 2017) treat depth and egomotion estimation as separate tasks and consequently employ separate networks (see Figure 1a). During training, the independently estimated predictions are coupled as part of the view synthesis procedure, which uses depth and egomotion to reconstruct the target view from a nearby source view. We call this *indirect* coupling because there is no explicit linking of depth and egomotion, rather, the weights of each network are coupled through gradient flow alone. Although this form of coupling is sufficient to jointly learn depth and egomotion, the major drawback is that the networks become decoupled at test time, which prevents contextual information (such as scale) from being passed from one network to the other. Recently, however, it has been proposed that—owing to its self-supervised nature—the reconstruction loss can be retained at test time and be further minimized by a gradient-descent-based optimizer (Li et al., 2020; Zhang et al., 2021; Chen et al., 2019b; Watson et al., 2021; McCraith et al., 2020; Shu et al., 2020; Kuznietsov et al., 2021). In so doing, the indirect link between networks (via gradient flow from the loss function) is preserved at test time, and multiview geometry constraints can be enforced by minimizing the error from multiple source images. Chen et al. (2019b) initially proposed two unique optimization approaches: parameter fine-tuning (PFT), which further optimizes the network weights, and output fine-tuning (OFT), which directly optimizes the depth or egomotion predictions. Our approach uses the former for indirect coupling of the networks at test time.

6.1.2 Direct Coupling

Beyond basic coupling within the loss function, other methods promote consistency between depth and egomotion by explicitly linking depth and egomotion within the network structures. Godard et al. (2019) experimented with weight sharing to effectively merge the networks; however, they report that a baseline 'shared' network structure is less accurate than a system with independent networks. In our own experiments, we also find this to be true. Other methods (Wang et al., 2019; Li et al., 2020; Ambrus et al., 2020; Zou et al., 2020) estimate egomotion as a function of the predicted depth (see Figure 1b), with Ambrus et al. (2020); Zou et al. (2020) providing ablation studies indicating that doing so improves accuracy. Less commonly, egomotion predictions have been applied to directly aid in the estimation of depth (Watson et al., 2021; Wang et al., 2020).

6.1.3 Feedback Coupling

Feedback coupling is a method that enables network *introspection* by reformulating the input as a cost map built with the current depth and egomotion predictions (see Figure 1c). The cost map explicitly encodes error within the input, which the networks can utilize by iteratively updating the current prediction (i.e., through multiple forward passes). As the predictions improve, the cost map is updated, and this process is repeated until convergence (i.e., until the cost map is minimized). We identify two existing self-supervised feedback coupling methods in the literature. Nabavi et al. (2020) use a photometric cost map consisting of the target view and a synthetic 'target' view (generated from a nearby source view); optimal depth and egomotion predictions maximally align the synthesized view with the target view, effectively minimizing the cost map. Gu et al. (2021), the authors of the Deep Recurrent Optimizer (DRO), adopt a similar approach, but replace the photometric cost map with a feature-metric one.[1] Further, DRO employs iterative feedback for egomotion *and* depth estimation, where both networks take the cost map as input. The increased complexity of this system requires a customized training procedure that alternates between training of the depth and egomotion network weights to maintain stability. Finally, other methods (Wei et al., 2020; Tang and Tan, 2019; Teed and Deng, 2020; Ummenhofer et al., 2017; Clark et al., 2018) use iteration for feedback coupling but require supervision to train the relatively complex network structures.

Herein, we demonstrate that feedback-based coupling is a crucial estimation component that promotes scale consistency between depth and egomotion, significantly boosting the overall system accuracy and improving generalization. We extend the feedback coupling approach of Nabavi et al. (2020) by incorporating indirect coupling into our system using a test-time PFT strategy to achieve *tight coupling* of predictions. That is, our approach links depth to egomotion, and vice versa, such that an improvement in one leads to an improvement in the other.

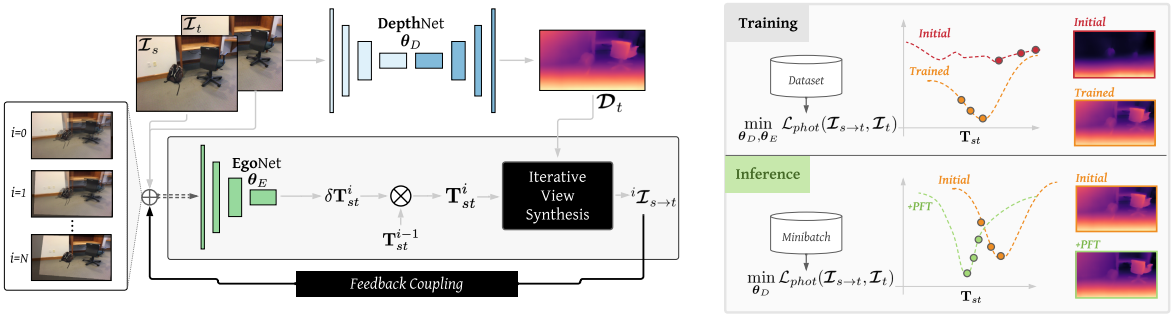
6.2 Methodology

We detail our tightly coupled approach in three parts. First, we introduce our baseline (decoupled) depth and egomotion networks and the self-supervised loss formulation used for training. Second, we describe the modification of this baseline system to include feedback coupling based on iterative view synthesis. Finally, we present our test-time depth optimization technique that forms the final component of our tightly coupled framework for estimating depth and egomotion. See Figure 6.2 for an illustration of our system.

6.2.1 Baseline Depth and Egomotion Framework

For our baseline system, we use the same network structures and loss functions from the previous chapter. We review these details again here briefly. Our system consists of DepthNet, $\boldsymbol{\mathcal{D}}_t = f_{\boldsymbol{\theta}_D}(\boldsymbol{\mathcal{I}}_t)$ and EgoNet, $\boldsymbol{\xi}_{st} = f_{\boldsymbol{\theta}_E}(\boldsymbol{\mathcal{I}}_s, \boldsymbol{\mathcal{I}}_t)$. The exponential map is used to produce the $\mathrm{SE}(3)$ pose change through $\mathbf{T}_{st} = \exp\left(\boldsymbol{\xi}_{st}{}^{\wedge}\right)$. These networks are jointly trained through minimization of the photometric reconstruction loss, consisting of $\mathcal{L}_{phot}$ (with the L_1 and SSIM terms), in addition to the inverse depth smoothness loss $\mathcal{L}_S$ and the depth consistency loss $\mathcal{L}_{DC}$. We also include the pose consistency loss term, $\mathcal{L}_{PC}$ from the previous chapter. Overall,

[1] Note that our definition of a cost map is generalized to encompass the variations in Nabavi et al. (2020) and Gu et al. (2021); the former implicitly encodes the error within the stacked images, while the latter explicitly computes the error within the input using the L_2 norm. 'Minimization' in the former case refers to maximal alignment of the two images, and a zeroing of the input in the latter.



(a) Our proposed network structure.

(b) Our training and test time modes.

Figure 6.2: System overview: (a) We use feedback coupling through iterative view synthesis (Nabavi et al., 2020) to produce an egomotion prediction that is a function of depth. (b) At test time, the depth network weights are updated via gradient descent to further minimize the sample-wise loss. As a result of our unique coupling scheme, both depth and egomotion predictions improve as the depth network weights are updated.

the *per-sample* loss that we use to train the system, averaged across all training samples, is

$$\mathcal{L}_{train} = \frac{1}{HW} \sum_{u,v} (\lambda_{phot} \mathcal{L}_{phot} + \lambda_S \mathcal{L}_S + \lambda_{DC} \mathcal{L}_{DC}) + \frac{1}{6} \sum \lambda_{PC} \mathcal{L}_{PC}. \tag{6.1}$$

Note that within $\mathcal{L}_{phot}$, we use the automasking and minimum reprojection techniques, along with the self-discovered mask, to remove and/or downweight the unstable image regions. See Section 4.2.2 for additional information. This baseline system employs no coupling aside from the indirect coupling that happens during training. To better couple our predictions, we use the feedback coupling approach based on iterative view synthesis (described below).

6.2.2 Feedback-Coupled Egomotion Prediction

In line with Nabavi et al. (2020), we extend the standard egomotion estimation approach to incorporate feedback through iterative view synthesis. We perform multiple forward passes through the network and redefine the egomotion network input as a photometric cost map that is updated after each iteration. We propose (and later will experimentally demonstrate) that including iteration improves convergence during training by making the egomotion *a function of the current depth prediction*. With this added contextual information being provided to the egomotion network, the consistency between the depth and egomotion predictions is improved. Importantly, due to this added link, the networks can account for the inherent scale ambiguity that is present in monocular images.

To incorporate feedback coupling via iterative view synthesis, no network architecture changes are required. The only difference is that multiple passes through the network are made, where the i^{th} pass takes as input a recursively updated cost map based on the images $\{{}^{i}\boldsymbol{\mathcal{I}}_{s\to t}, \boldsymbol{\mathcal{I}}_t\}$. Here, ${}^{i}\boldsymbol{\mathcal{I}}_{s\to t}$ denotes the reconstructed image (created through view synthesis, as Section 4.2.2 describes in detail) after the i^{th} forward pass. With this modified EgoNet input, subsequent passes through the network produce a correction $\delta\boldsymbol{\xi}^i_{st} = f_{\boldsymbol{\theta}_E}({}^{i}\boldsymbol{\mathcal{I}}_{s\to t}, \boldsymbol{\mathcal{I}}_t)$ that better aligns the current reconstructed source image ${}^{i}\boldsymbol{\mathcal{I}}_{s\to t}$ with the target image. For N iterations, corrections are

compounded with the current egomotion prediction to produce an updated estimate through

$$\mathbf{T}_{st}^{i} = \left(\prod_{i=1}^{N} \delta \mathbf{T}_{st}^{i} \right) \mathbf{T}_{st}^{0} \approx \left(\mathbf{I}_4 + \sum_{i=1}^{N} \delta \boldsymbol{\xi}_{st}^{i^\wedge} \right) \mathbf{T}_{st}^{0}, \tag{6.2}$$

where the initial egomotion estimate $\mathbf{T}_{st}^{0}$ is considered an egomotion prior used to initialize the system.[2] We initialize with the identity matrix (i.e., by making the assumption that the camera is stationary) but note that a constant velocity assumption could alternatively be used. In subsequent iterations, the warped image and the target image are used as inputs to produce an egomotion correction. Notably, since ${}^{i}\boldsymbol{\mathcal{I}}_{s \to t}$ is a function of the current depth prediction and the *corrected* egomotion prediction, EgoNet is able to take into account the error that is explicitly encoded into the input cost map (through any misalignment of the two input images) when producing the subsequent egomotion correction.

Feedback Resolves Scale Inconsistency. We hypothesize that feedback coupling, through the added link between EgoNet and DepthNet, will pass scale information from the current depth prediction into EgoNet. In doing so, the networks no longer need to implicitly learn a mutually consistent scale factor, which is challenging to do with monocular images (using the unscaled photometric reconstruction loss). As a result, the inter-network scale inconsistency, present in the baseline decoupled networks at training and test time, can be mitigated. Significantly, removing scale inconsistency removes a source of error from the photometric reconstruction loss, which we believe will improve the gradient flows to the networks during training. Furthermore, the linking of the networks produces a secondary path for gradient flow into the depth network: the gradients are now able to backpropagate through EgoNet and into DepthNet.

To demonstrate the impact of incorporating iteration, we visualize some sample-specific photometric reconstruction loss curves (as a function of egomotion) in Figure 6.3.[3] Here, we observe that through the application of subsequent forward passes, the egomotion predictions gradually converge to the minimum of the loss function. Note that if the input depth prediction is erroneous, the minimum loss will not always align with the ground truth value of egomotion; we address this by refining depth predictions through PFT, which we describe next.

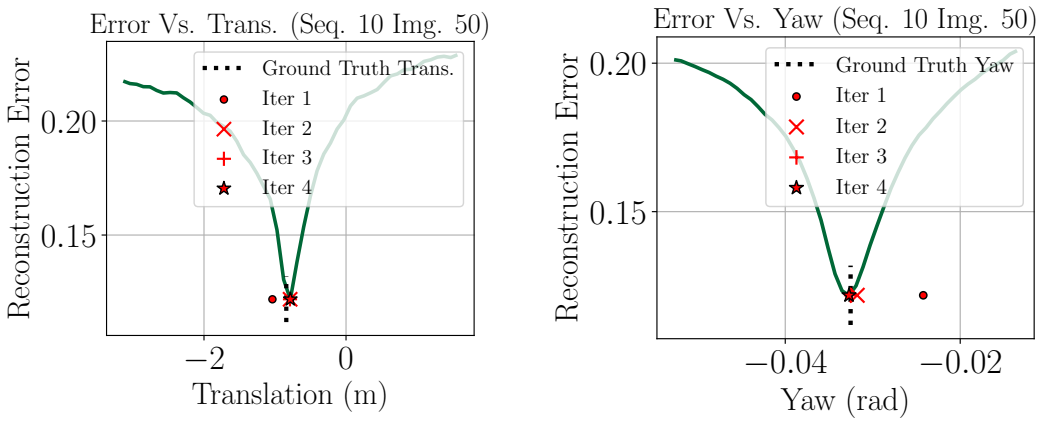
6.2.3 Tightly Coupled Depth and Egomotion Optimization

We extend the feedback coupling method, linking depth with egomotion, by doing the converse: linking egomotion with depth. This additional network coupling is achieved by incorporating a test-time PFT method that refines the depth network weights by further minimizing the self-supervised loss through gradient descent. By incorporating our feedback-coupled egomotion network with our depth optimizer, we can achieve *tightly coupled* optimization of depth and egomotion. The novelty of our optimization procedure with respect to other PFT methods is that we can produce refined depth *and* egomotion predictions through the optimization of our depth network only, because our (feedback-coupled) egomotion predictions are already a function of depth. As the depth prediction is refined by PFT, we recompute an improved egomotion prediction via the iterative egomotion network (whose cost-map input is updated with the refined depth).

To perform PFT at test time, we minimize the same loss as Equation (6.1) but replace the smoothness term with a depth prior that ensures that the optimized depth $\boldsymbol{\mathcal{D}}_t^*$ remains similar to the original depth prediction.

[2] This approximation simplifies the training procedure and has a negligible impact on performance, since the pose corrections are small quantities applied to an already small egomotion prediction.

[3] The 1D loss curves were generated by sampling forward translation or yaw values via a grid search and then plotting the resulting $\mathcal{L}_{phot}$ produced using these sampled values and the current depth prediction. Then, the egomotion network's real prediction at each iteration is overlayed on this curve.



(a) Forward translation.

(b) Orientation (yaw).

Figure 6.3: 1D loss experiments illustrating the impact of adding coupling at test time. We visualize the loss curve (for a sample from the KITTI dataset) as a function of the two primary degrees of freedom for the camera: the forward axis translation, and the yaw axis orientation. Feedback-based coupling iteratively updates the initial egomotion prediction to (approximately) align with the minimum of the loss.

The resulting PFT loss is

$$\mathcal{L}_{PFT} = \frac{1}{HW}\sum_{u,v}\left(\lambda_{phot}\mathcal{L}_{phot} + \lambda_{DC}\mathcal{L}_{DC} + \lambda_{prior}\mathcal{L}_{SSIM}(\boldsymbol{\mathcal{D}}_t^*, \boldsymbol{\mathcal{D}}_t)\right) + \frac{1}{6}\sum \lambda_{PC}\mathcal{L}_{PC}. \tag{6.3}$$

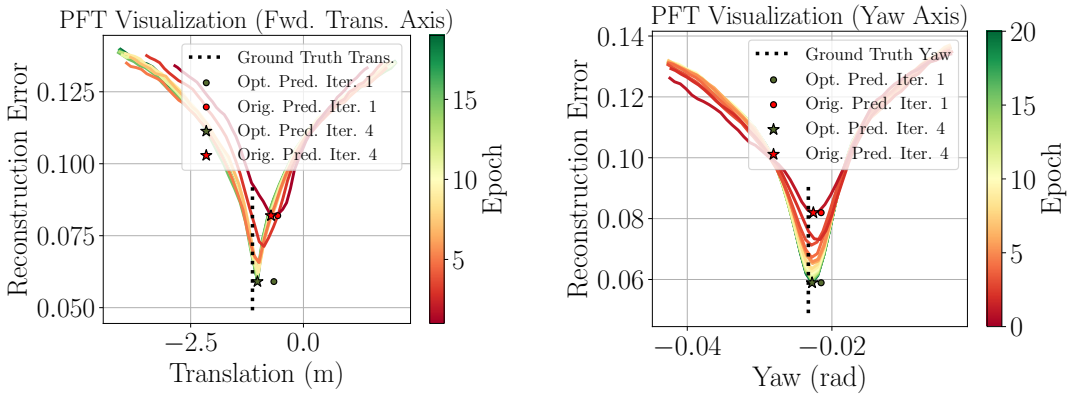
Figure 6.4 illustrates our tightly coupled optimization procedure in one dimension for a single test-time sample. Beginning with the initial (red) loss curve produced by our trained depth network, the egomotion network recursively updates its predictions to converge to the (suboptimal) minimum. Running through each epoch of the test-time depth optimizer shifts the loss to a new (lower) minimum by improving the quality of the depth prediction; then, given the new depth prediction, our egomotion network converges to the new minimum. We visualize an example of the improvement in depth accuracy (and the corresponding reduction in the photometric reconstruction error) in Figures 6.5 and 6.6.

6.3 Experiments

We provide details of our network structure and training procedure below, followed by extensive experimental results on the ScanNet, KITTI, and Oxford RobotCar datasets. Specifically, we evaluate the performance of our tightly coupled system on depth and egomotion benchmarks from KITTI and ScanNet, showing that we achieve state-of-the-art accuracy on the KITTI odometry benchmarks and the ScanNet benchmarks. Further, we include ablation studies that indicate how both feedback and indirect coupling, while useful on their own, complement each other when combined. Finally, we demonstrate additional benefits of feedback coupling—in particular, this coupling improves generalization across datasets, as we demonstrate through cross-dataset evaluation (from KITTI to Oxford RobotCar).

6.3.1 Datasets and Evaluation Metrics

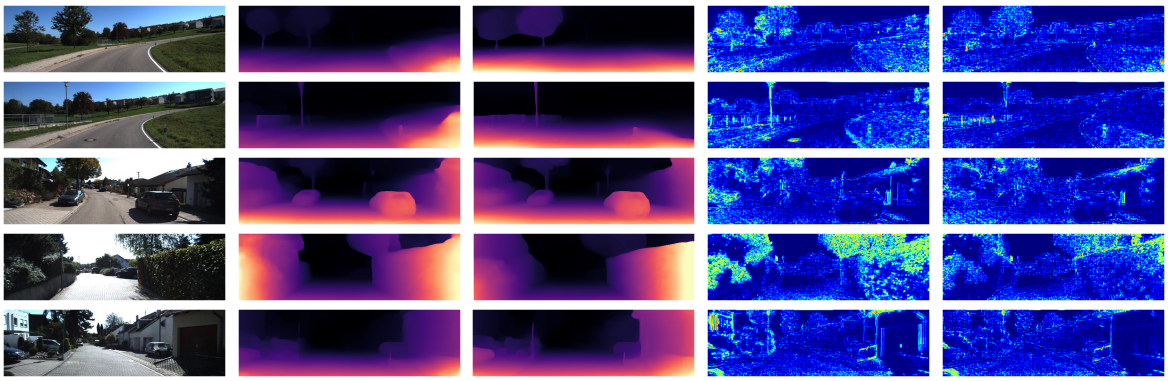
We primarily evaluate our system on KITTI and ScanNet. Additionally, we include a cross-dataset evaluation on Oxford RobotCar. We refer the reader to Appendix A for further details on these datasets and their common



(a) Forward translation.

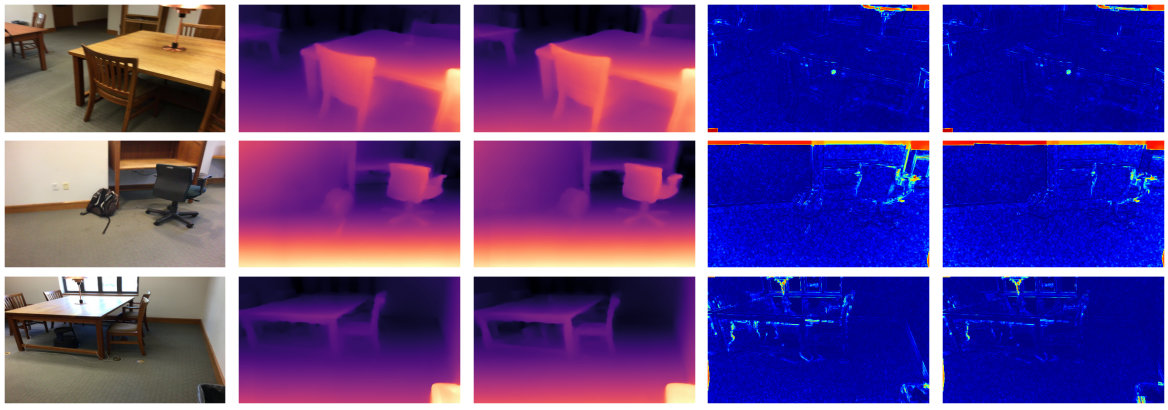
(b) Orientation (yaw).

Figure 6.4: Incorporating indirect coupling into the 1D loss experiment. Through the application of PFT for depth optimization, the minimum of the loss curve reduces and shifts towards to ground truth egomotion value. The egomotion prediction, thanks to feedback coupling, is able to converge to the new minimum without retraining.



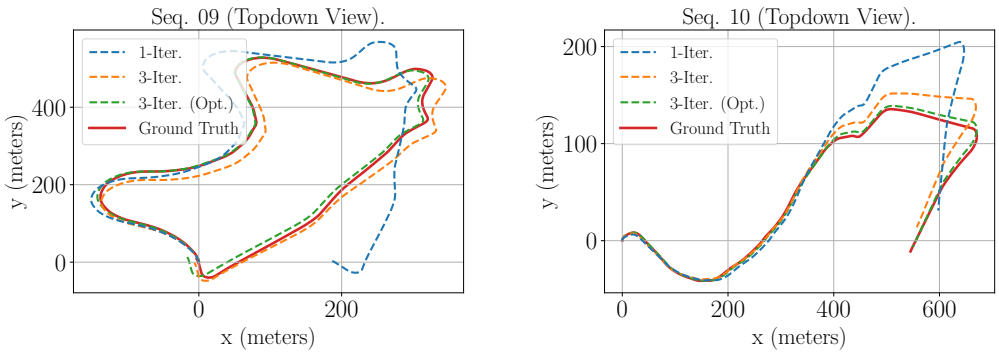
(a) Image **(b)** Depth (no PFT) **(c)** Depth (with PFT) **(d)** Error (no PFT) **(e)** Error (with PFT)

Figure 6.5: Depth predictions and reconstruction errors before and after applying our test-time optimization scheme on KITTI dataset images.



(a) Image **(b)** Depth (no PFT) **(c)** Depth (with PFT) **(d)** Error (no PFT) **(e)** Error (with PFT)

Figure 6.6: Depth predictions and reconstruction errors before and after applying our test-time optimization scheme on ScanNet dataset images.



(a) Seq. 09 topdown trajectory.

(b) Seq. 10 topdown trajectory.

Figure 6.7: KITTI odometry results on the test sequences. Our tightly coupled system leads to improved egomotion accuracy compared with the baseline method (1-iteration, no PFT).

evaluation metrics.

ScanNet Dataset. We follow the training/test split from Tang and Tan (2019), where the first 1,413 sequences are used for training and 2000 image pairs from the remaining 100 sequences are selected for testing. Our networks use images downsized to 256×448. For evaluation on ScanNet, we report the depth and camera pose metrics as described in Tang and Tan (2019) (and Appendix A) and, similar to existing literature, use image-wise rescaling to align the predictions with ground truth for both depth and poses.

KITTI Dataset. Our networks use images downsized to 192×640. For depth evaluation, we follow the Eigen train/test split and follow the standard procedure from Godard et al. (2019) by reporting depth accuracy after per-image median ground truth scaling. For odometry evaluation, we train with sequences 00, 02, and 05 through 08, and test on sequences 09 and 10. We report the average translational and rotational errors ($t_{err}(\%)$, $r_{err}(^o/100\text{ m})$) over possible sub-sequences of length (100, 200, ..., 800) metres. Since we do not employ a scale recovery method in this system, we align the (unscaled) trajectories with ground truth by applying a constant scale factor to the translation values of the estimated trajectory.

Oxford Dataset. We downsize the images to 192×640 and use the training/validation split from Appendix A. The mean segment errors (across the same sub-sequences as KITTI) are used for odometry evaluation.

6.3.2 Implementation Details

We use the same depth and egomotion network structures from the previous chapter, which are outlined in Appendix B. For the KITTI and ScanNet experiments, we pretrain our networks on the Oxford RobotCar dataset. We train our models on an NVIDIA Titan V GPU for 25, 45, and 15 epochs on the KITTI odometry, KITTI Eigen and ScanNet datasets, respectively, using the Adam optimizer ($\beta_1 = 0.9, \beta_2 = 0.999$). We use a learning rate of (1×10^{-4}, 2×10^{-4}) for the depth and egomotion network, respectively, that is halved four times over the duration of training. During training we apply data augmentation in the form of horizontal flipping and modifications to hue, saturation, contrast, and brightness. For the Equation (6.1) training loss, we use $\alpha = 0.85$, $\lambda_{phot} = 1$, $\lambda_S = 0.05$, $\lambda_{DC} = 0.15$, and $\lambda_{PC} = 5$. When training our iterative egomotion network, we only compute the loss once using the egomotion prediction from the final iteration, rather than after every forward

Table 6.2: ScanNet results for the standard two-view test set. The ablation study indicates that more iterations and applying PFT at test time leads to improved performance. We specify the number of iterations during training (x) and test (y) as x/y iter.

	Method	Abs. Rel. ↓	Sq. Rel. ↓	RMSE ↓	RMSE Log ↓	Sc. Inv. ↓	Rot. (°) ↓	Tr (°) ↓	Tr. (cm) ↓
	LSD-SLAM (Engel et al., 2014)	0.268	0.427	0.788	0.330	0.323	4.409	34.36	21.40
Sup.	DeMoN (Ummenhofer et al., 2017)	0.231	0.520	0.761	0.289	0.284	3.791	31.626	15.50
	BANet (Tang and Tan, 2019)	0.161	0.092	0.346	0.214	0.184	1.018	20.577	3.39
	DeepSFM (Wei et al., 2020)	0.227	0.170	0.479	0.271	0.268	1.588	30.613	—
	DeepV2D (Teed and Deng, 2020)	0.069	0.018	0.196	0.099	0.097	0.692	11.731	1.902
	DRO (12 iter.) (Gu et al., 2021)	**0.053**	**0.017**	**0.168**	**0.081**	**0.079**	**0.473**	**9.219**	**1.160**
Self-Sup.	DRO (12 iter.) (Gu et al., 2021)	0.140	0.127	0.496	0.212	0.210	0.691	11.702	1.647
	Ours (4/8 iter.)	**0.088**	**0.038**	**0.260**	**0.125**	**0.122**	**0.547**	**10.482**	**1.290**
	Ours (using GT Depth)	—	—	—	—	—	0.516	10.366	1.247
Ablation	1/1 iter. (no PFT)	0.181	0.117	0.462	0.229	0.223	1.948	43.363	5.612
	1/1 iter.	0.186	0.125	0.462	0.233	0.228	1.948	43.363	5.612
	2/2 iter. (no PFT)	0.126	0.062	0.342	0.167	0.162	1.137	20.790	2.800
	2/4 iter. (no PFT)	0.126	0.062	0.342	0.167	0.162	0.958	17.496	2.270
	2/4 iter.	0.111	0.054	0.302	0.150	0.147	0.821	15.195	1.952
	4/4 iter.	0.092	0.040	0.266	0.129	0.126	0.695	12.340	1.669
	4/8 iter. (no PFT)	0.103	0.044	0.292	0.140	0.137	0.690	12.389	1.635

pass through the egomotion network. In doing so, the iterative approach can be incorporated into the standard self-supervised depth and egomotion pipeline without having to modify the training procedure.

Some Notes on PFT

Our PFT optimization scheme uses the Adam optimizer to update the depth network weights while the egomotion network weights are fixed. Following McCraith et al. (2020), we only update the depth encoder weights instead of updating the full depth network, since this reduces the optimization time and can improve optimization stability. For our Equation (6.3) PFT loss, we use the same hyperparameters as training, while setting $\lambda_{prior} = 0.1$. For every test set sample, 20 optimization 'epochs' are performed. Each epoch involves: forward passes through the depth and (iterative) egomotion network, computation of the loss function using the current predictions, and a backward pass to update the depth network weights using the Adam optimizer. We perform the optimization over a minibatch of three samples (each consisting of a target image and the two adjacent source images), which helps with regularization and leverages available GPU memory to increase the optimization speed. Instead of returning the network predictions from the final epoch, we average the predictions from the last five epochs to prevent a noisy gradient step from negatively impacting the optimization.

We evaluate the runtime of our method, running on an NVIDIA Titan V GPU, and report the operating frequency in Table 6.3. The operating frequency for the baseline (1-iteration, non-PFT) depth and egomotion networks is real-time capable; the increase from a single egomotion iteration to four iterations reduces the overall frequency, but real-time capability is maintained. Applying PFT at test time, however, can cause a significant reduction in the operating frequency, since each PFT epoch requires a new forward pass through the depth and

Table 6.3: A runtime comparison for our proposed methods. The PFT slows down the system at test time, but only a small amount of optimization epochs are required to improve the egomotion accuracy. Notably, real-time performance is achieved using the 'sequential' optimization strategy from McCraith et al. (2020). The translation (t_{err}) and rotation (r_{err}) errors are from KITTI sequence 09.

Mode	# PFT Epochs	Frames Per Sec.	t_{err} (%)	r_{err} (°/100 m)
1-iter	—	56.69	8.42	2.52
4-iter	—	37.11	2.98	0.66
4-iter	20	1.51	1.19	0.32
4-iter	10	3.05	1.24	0.28
4-iter	5	6.20	1.35	0.40
4-iter	1	16.16	1.65	0.52
4-iter PFT (Sequential[1] McCraith et al. (2020))	1	21.6	1.59	0.47

[1] In the sequential method, the network is continuously optimized throughout the sequence without resetting the model back to its initial weights (like we do in our original PFT method) after optimizing each minibatch.

egomotion networks, followed by an optimization step. As shown in Table 6.3, however, only a small number of optimization steps are required to improve performance.

Lastly, in the course of evaluating our approach on the KITTI odometry dataset, we identified that sample-wise optimization often increases *inter-frame* scale drift, since the PFT procedure can independently shift the scale factor for individual samples within the test sequences. To promote a uniform scale factor across an entire sequence, we incorporate a (self-supervised) online scale recovery module based on DNet (Xue et al., 2020). For each image, we estimate the camera height (relative to the ground plane) using our depth prediction and then normalize the corresponding translation prediction using this quantity. We refer the reader to Table 6.5 for results from ablation experiments that show the accuracy of our method without applying this rescaling. It is important to note that this inter-frame scale inconsistency differs from the *inter-network* scale inconsistency that we address in this chapter through the introduction of the tightly coupled networks.

6.3.3 Experimental Results

ScanNet Results

Table 6.2 presents our results on the ScanNet test split. Our method significantly outperforms the self-supervised variant of DRO and is competitive with supervised methods. We additionally include the pose results from our egomotion network when using the available ground truth depth (instead of our predicted depth) to iteratively warp the source image. The improved pose accuracy indicates that our egomotion network functions with an arbitrary depth map that was not present during training. Our ablation study reveals that feedback coupling (through iteration) is crucial for improving accuracy—without this, even the PFT has very little impact.

KITTI Results

Table 6.4 and Figure 6.7 present the performance of our tightly coupled system on the KITTI odometry test sequences. Our proposed method achieves state-of-the-art egomotion accuracy compared with other learning-based methods. Our ablation study (Table 6.5) indicates that coupling is crucial for achieving this level of accuracy. Including feedback and applying our PFT strategy are both highly effective to improve accuracy; incorporating both leads to the best performance. Notably, our coupling approach produces better egomotion estimates than the more simplistic direct coupling methods (Ambrus et al., 2020; Wang et al., 2019) that treat depth as an input into the egomotion network.

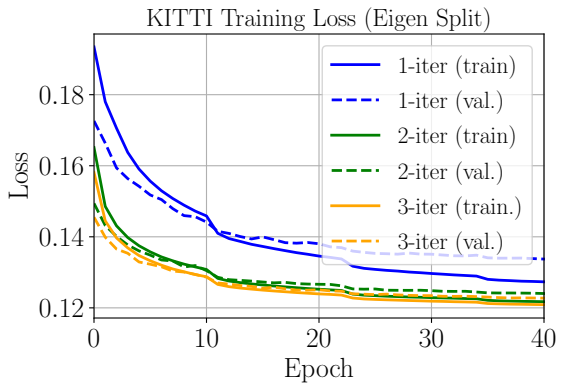


Figure 6.8: Training losses for the Eigen split. Increasing the amount of egomotion iterations during training leads to improved convergence on the training set and improved generalization on the validation set.

Table 6.4: KITTI odometry results on the standard test sequences. We report the mean translation and rotation segment errors. Our proposed method significantly outperforms other learning-based methods, including others that use PFT at test time.

Method	Test-Time PFT	Seq. 09		Seq. 10	
		t_{err} (%)	r_{err} (°/100 m)	t_{err} (%)	r_{err} (°/100 m)
SfMLearner (Zhou et al., 2017)		11.32	4.07	15.25	4.06
SC-SfMLearner (Bian et al., 2019)		8.24	2.19	10.7	4.58
Ambrus et al. (2020)		6.72	1.69	9.52	1.59
MonoDepth2 (Godard et al., 2019)		17.80	3.86	12.41	5.33
ManyDepth[1] (Watson et al., 2021)		13.36	2.93	10.18	4.25
Wang et al. (2019)		9.30	3.50	7.21	3.90
Zou et al. (2020)		3.49	1.00	11.80	1.80
Shu et al. (2020)	✓	8.75	2.11	10.67	4.91
Li et al. (2020)	✓	5.89	3.34	4.79	0.83
DOC (Zhang et al., 2021)	✓	2.02	0.61	2.29	1.10
Ours (4-iter)	✓	**1.19**	**0.30**	**1.34**	**0.37**

[1] We generated this with their publicly available model.

Table 6.7 shows the depth accuracy of our system on the Eigen test split; notably, we are competitive with other monocular, self-supervised methods, but lag behind some recent approaches like ManyDepth (Watson et al., 2021). We emphasize the utility of our method in producing accurate depth *and* egomotion estimates, whereas ManyDepth and Shu et al. (2020) are only able to estimate depth accurately. Figure 6.8 illustrates how our performance is in part due to the use of feedback coupling, which improves convergence and generalization during training as more iterations (i.e., forward passes of the egomotion network) are applied.

Feedback Coupling Experiments

In our final set of experiments we present evidence that supports the value of using feedback in egomotion estimation. First, we evaluate how feedback during iteration improves generalization to unseen data. This effect is demonstrated by evaluating our network on image pairs with inter-frame perspective changes that are significantly larger than those within the dataset (which can occur as a result of increased camera velocity or decreased camera frame rate). We modify the KITTI dataset by skipping images (i.e., adopting a stride greater than one) and then evaluate test sequence odometry error as a function of the increased perspective change. Figure 6.9a shows that as the number of iterations increases, the generalization performance improves. More-

over, error is reduced by applying additional iterations (beyond the amount used during training) at test time. In a second generalization experiment, we conduct a cross-dataset evaluation of our KITTI-trained model on the Oxford RobotCar dataset. Table 6.6 depicts these results, which indicate that iteration is crucial for generalizing to unseen data.

Next, we verify how *robust* the feedback coupling mechanism is by demonstrating the ability to minimize error when large perturbations are applied to the initial (first iteration) egomotion prediction. Concretely, we apply pose perturbations (in the forward translation direction, and along the yaw axis) to the first iteration prediction from EgoNet, which impacts the synthesized image for subsequent passes through the network. The translation perturbation is sampled from a uniform distribution in the range $[-\alpha, \alpha]$, where $\alpha \in \{0, 0.1, 0.25, 0.5, 0.75, 1.0, 1.5\}$ metres, and the yaw perturbation is sampled from a uniform distribution in the range $[-\beta, \beta]$, where $\beta \in \{0, 0.1, 0.25, 0.5, 1, 3, 5\}$ degrees. A random perturbation is applied to every sample within KITTI test sequences 09 and 10, and we report the resulting errors (averaged across the sequence) in Figure 6.9b. Here, we include the effects of applying translation and rotation perturbations independently but note that a similar result occurs even if both perturbations are applied together. As a baseline, we report the error for our single-iteration model with random noise added to its one-shot prediction (this is the error range we expect to see if our iterative network cannot correct for the added error). As demonstrated in Figure 6.9b, the increase in error from adding perturbations can be completely mitigated by applying only two extra iterations. During these subsequent iterations, the network, which takes as input the (erroneously) warped source image, produces *corrections* that effectively compensate for the initial perturbation.

Finally, we verify our claim that appropriate coupling of the network predictions improves inter-network scale consistency. For this experiment, we rescale the depth predictions for the KITTI test sequences by a constant scale factor and then observe how the scale of the translation predictions changes in response to the modified depth. Concretely, we simulate scale drift between the depth and egomotion networks by varying the scale factor s of the depth predictions through $\boldsymbol{\mathcal{D}}_{scaled} = s\boldsymbol{\mathcal{D}}_{pred}$. The same scale factor is applied to all predictions within the test sequences. This is repeated with scale factors in the range $s \in \{0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3\}$. We compute the change in translation norm (relative to the original translation norm with $s = 1$) of our iterative egomotion network predictions to observe how the translation predictions change as a result of the rescaled depth predictions. Figure 6.9c illustrates that the average change in the translation norms (for the iterative models) are proportional to the applied depth scaling factor. This provides support for the notion that the egomotion network is able to infer scale from depth predictions via our feedback coupling strategy.

Table 6.5: Ablation study on the KITTI dataset showing the effect of feedback coupling and indirect coupling (through PFT) at test time. Including both leads to the lowest error.

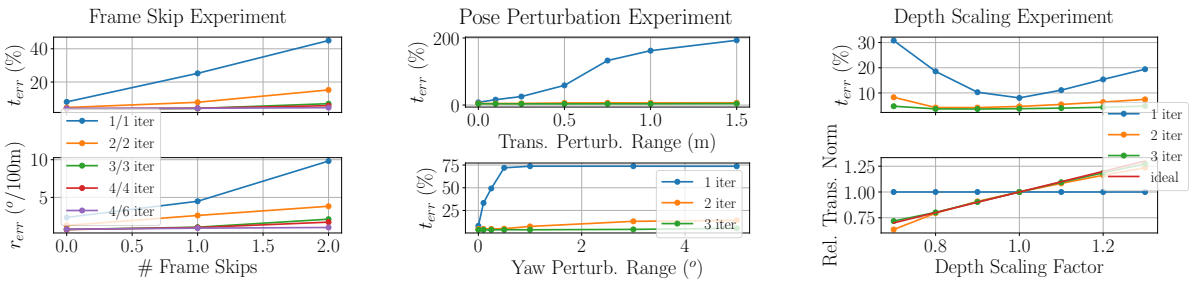
# Egomotion Iter. (train/test)	PFT Params	Seq. 09		Seq. 10	
		t_{err} (%)	r_{err} (°/100 m)	t_{err} (%)	r_{err} (°/100 m)
1 / 1	–	8.42	2.52	9.10	4.11
1 / 3	–	6.16	1.51	10.13	2.74
2 / 2	–	4.49	1.22	6.86	2.09
4 / 4	–	2.98	0.66	3.38	1.02
4 / 6	–	3.02	0.69	3.03	0.93
1 / 1	$\boldsymbol{\theta}_D + \boldsymbol{\theta}_E$	2.00	0.51	2.36	0.95
4 / 4	$\boldsymbol{\theta}_D$	1.19	0.30	1.34	0.37
4 / 4 (no DNet scaling)	$\boldsymbol{\theta}_D$	2.36	0.30	2.36	0.38
4 / 4 (no $\mathcal{L}_{prior}$)	$\boldsymbol{\theta}_D$	1.79	0.28	2.13	0.46
4 / 4	$\boldsymbol{\theta}_D + \boldsymbol{\theta}_E$	1.28	0.31	1.36	0.35

Table 6.6: Cross-dataset evaluation results on the Oxford RobotCar dataset (sequence 2014-11-18-13-20-12). The translation and rotation mean segment errors are reported. When training on KITTI and testing on Oxford, our iterative approach generalizes significantly better than the baseline 1-iteration model.

Training Dataset	**Subseq.** 0		**Subseq.** 1	
	t_{err} (%)	r_{err} (°/100 m)	t_{err} (%)	r_{err} (°/100 m)
Ox. (1-iter)	7.75	3.26	9.50	3.43
KITTI (1-iter)	36.45	12.05	32.99	13.81
Rel. Change	-370.32%	-269.63%	-247.26%	-302.62%
Ox. (3-iter)	4.84	2.18	6.46	2.38
KITTI (3-iter)	5.83	3.11	11.44	4.54
Rel. change	-20.45%	-42.66%	-77.09%	-90.76%

Table 6.7: Monocular depth prediction results for self-supervised methods on the Eigen test split (Eigen and Fergus, 2015). We report results from our depth network following test time PFT. In all other cases, we report the results from the image resolution closest to ours.

Method	**Test-Time PFT**	**Error** ↓				**Accuracy** ↑		
		Abs. Rel.	Sq. Rel.	RMSE	RMSE Log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Bian et al. (2019)		0.128	1.047	5.234	0.208	0.846	0.947	0.976
Nabavi et al. (2020)		0.160	1.195	5.916	0.245	0.774	0.917	0.964
MonoDepth2 (Godard et al., 2019)		0.115	0.903	4.863	0.193	0.877	0.959	0.981
PackNet (Guizilini et al., 2020a)		0.107	0.803	4.566	0.197	0.876	0.957	0.980
Guizilini et al. (2020b)		0.111	0.785	4.601	0.189	0.878	—	—
DRO (12 iter.) (Gu et al., 2021)		**0.088**	0.797	4.464	0.212	0.899	0.959	0.980
GLNet (Chen et al., 2019b)	✓	0.099	0.796	4.743	0.186	0.884	0.955	0.979
ManyDepth (Watson et al., 2021)	✓	0.090	0.713	4.261	0.170	0.914	**0.966**	**0.983**
McCraith et al. (2020)	✓	0.089	0.747	4.275	0.173	0.912	0.964	0.982
Shu et al. (2020)	✓	**0.088**	**0.712**	**4.137**	**0.169**	**0.915**	0.965	0.982
CoMoDA (Kuznietsov et al., 2021)	✓	0.102	0.871	4.596	0.183	0.898	0.961	0.981
Ours (4-iter)	✓	0.097	0.791	4.383	0.178	0.896	0.961	0.982



(a) **(b)** **(c)**

Figure 6.9: Summary of the experiments demonstrating the utility of using an iterative egomotion network. Benefits include: (a) improved generalization to unseen data, (b) robustness to initial prediction error, and (c) improved inter-network scale consistency.

6.4 Summary and Future Work

In this chapter, we demonstrated that an appropriate *coupling* of depth and egomotion networks improves performance in self-supervised SfM. We introduced a taxonomy of coupling methods and discussed the potential benefits of incorporating each method to promote consistency between depth and egomotion predictions at training and at test time. We noted the particular importance of feedback coupling as a method that iteratively updates an initial prediction to further minimize the self-supervised reconstruction loss. Building on these insights, we presented our own *tightly coupled* approach and, through extensive experiments, showed that our system was consistently accurate on indoor and outdoor datasets and achieved state-of-the-art accuracy on several key benchmarks.

Future work could investigate how *temporal* coupling can be incorporated into our tightly coupled framework in order to share information between adjacent samples. There are some recent methods that do so using recurrent neural network structures for the depth and egomotion networks (Lee et al., 2020; Vaishakh et al., 2020). An interesting extension to these methods would be to include feedback coupling to iteratively update predictions across longer temporal sequences. Doing so could potentially enforce consistency between all of the predictions within the window, like sliding window bundle adjustment does within classical VO or SfM methods.

Chapter 7

A Hybrid System for Robust, Self-Supervised VIO

The previous three chapters diverged from the aided INS framework proposed in the first chapter to focus on a particular data-driven model: the learned SfM pipeline that applies neural networks trained end-to-end to produce depth and egomotion predictions. The final contribution of this thesis is an approach for utilizing the learned SfM system as a data-driven measurement model within the broader aided inertial navigation framework. The resulting system is a hybrid visual-inertial odometry (VIO) estimator that is capable of maintaining tracking under significantly degraded visual operating conditions. In this chapter, we describe the system in detail and contrast it with alternative classical and learned VIO estimators. We show that our hybrid VIO formulation has a number of attributes that make it superior to these alternatives. Our experiments demonstrate the robustness of our algorithm to visually degraded conditions, relative to the performance of classical estimators.

The hybrid VIO system (illustrated in Figure 7.1) uses a differentiable filter (DF) (Kloss et al., 2021) to incorporate learned egomotion measurements into an uncertainty-aware state estimator. IMU measurements are used to propagate the state forward in time through an integration-based process model. Our work builds upon that of Li and Waslander (2020), who used supervised pose labels to train a similar system. We relax this training requirement by replacing the supervised loss with the self-supervised photometric reconstruction loss that can be leveraged to train the full system end to end. In short, the main contributions of this chapter are as follows:

1. we incorporate the egomotion network component of our tightly coupled, learned SfM system as a relative pose measurement model within a hybrid, EKF-based VIO estimator;
2. we employ our self-supervised loss formulation to train this hybrid system end to end, leveraging the sequential nature and differentiability of the EKF to backpropagate from the loss function, through the filter, and into the network components;
3. we show that heteroscedastic uncertainty estimates for our network-based egomotion measurements can be learned with our self-supervised formulation, facilitating principled sensor fusion within the filter;
4. we demonstrate how, unlike other self-supervised formulations for learned VIO, our approach facilitates scale recovery by incorporating a scale parameter as part of the EKF state;
5. we present extensive experimental results demonstrating how our proposed hybrid estimator is more robust to failure than a number of state-of-the-art, classical VIO approaches under degraded conditions.

7.1 Related Work

In this section, we separate existing approaches to VIO into three categories: classical, learned, and hybrid, in reference to the amount of (or the lack of) machine learning involved in each case. We identify several key limitations of approaches based purely on classical or learning frameworks, and then discuss how hybrid systems can leverage the benefits—and mitigate the limitations—of purely classical and learning-based VIO systems.

7.1.1 Classical Approaches to VIO

Similar to the classical VO algorithms introduced in Chapter 4, sparse (feature-based) VIO algorithms generally consist of a front-end and a back-end (Gui et al., 2015) component.[1] The front-end detects and tracks features across images. The back-end estimates the 3D locations of tracked features and the camera trajectory using a filtering or optimization-based framework. The formulation can be either loosely coupled or tightly coupled. Loosely coupled methods estimate the system state with each sensor modality independently and then combine the estimates in a final stage. Conversely, tightly coupled methods incorporate each modality within a joint estimation framework.[2] To determine the motion, the reprojection error between predicted and observed feature locations is minimized. The reprojection error may be expressed indirectly in terms of pixel coordinates, or directly in terms of a photometric (pixel intensity) loss based on the brightness constancy assumption. The available IMU information is used within a motion model to propagate the state between camera measurements in filter-based estimators, or is incorporated as a motion constraint in optimization-based estimators (Forster et al., 2015).

Initially, filter-based algorithms were the prevalent choice for VIO estimators. One well known algorithm is the multi-state constraint Kalman Filter (MSCKF) (Mourikis and Roumeliotis, 2007), but its world-centric formulation has been shown to be inconsistent (Li and Mourikis, 2013). Robot-centric formulations such as ROVIO (Bloesch et al., 2017) and R-VIO (Huai and Huang, 2018) have been proposed as alternatives to the world-centric framework. These robocentric approaches reformulate the VIO problem by estimating poses with respect to a moving, local frame (i.e., a robocentric frame), rather than with respect to a fixed, global navigation frame. The robocentric approach improves filter consistency (i.e., producing uncertainty estimates that represent the true uncertainty of the system) relative to world-centric approaches (Huai and Huang, 2018).

Optimization-based approaches have also been proposed that perform sliding window bundle adjustment over a window of frames, while relying on IMU preintegration (Forster et al., 2015) for computational efficiency. VINS-Mono (Qin et al., 2018) and OK-VIS (Leutenegger et al., 2015) are popular open-source systems that optimize over a sliding window of keyframes. Although more compute intensive, optimization methods tend to be more accurate than their filter-based counterparts.

Like the classical, feature-based VO estimators that we introduced in Chapter 4, the primary failure mode in VIO estimators is the loss of feature tracking within the front-end, which happens when there is a lack of stable image features. This lack of stable image features can be a result of rapid camera motion, poor illumination, or having dynamic objects or limited texture within the scene. Recent work has explored data-driven replacements to classical estimators. Data-driven replacements, by relying on fewer modelling assumptions, have potential to improve robustness in scenarios that violate the assumptions of classical systems.

[1] We only discuss sparse, feature-based methods in this section because the vast majority of VIO estimators fall within this category.
[2] Note that the term 'tightly coupled' here is distinct from our tightly coupled depth and egomotion network framework from Chapter 6.

7.1.2 Learning-Based Approaches to VIO

In existing data-driven VIO systems, learned models (commonly neural networks) are trained (Gurturk et al., 2021; Chen et al., 2019a; Clark et al., 2017; Almalioglu et al., 2022; Wei et al., 2021) to model the complex relationship between sensor measurements and egomotion. These systems all adopt a similar 'feature fusion' procedure, where the neural network learns to map the raw (visual and inertial) measurements to a 6-DOF egomotion prediction in an end-to-end manner. Internally, the network extracts and combines sensor-specific features (e.g., through concatenation of the two feature vectors). The resulting multimodal features are then fed into a final network block that predicts the egomotion. This scheme exists in both supervised (Gurturk et al., 2021; Chen et al., 2019a; Clark et al., 2017) and self-supervised (Almalioglu et al., 2022; Wei et al., 2021) settings, with the self-supervised methods minimizing the photometric reconstruction loss to jointly train depth and egomotion networks.

While end-to-end approaches are effective, they do not utilize the standard relationship between inertial measurements and robot/vehicle dynamics. Ignoring this relationship burdens the network with learning well-modelled kinematics from scratch and prevents the network from utilizing the metric information in the inertial data. For example, to effectively utilize the specific force measurements from the IMU, the network must learn to implicitly track the velocity and global orientation of the IMU. Additionally, metric information available from inertial measurements cannot easily be utilized because reconstruction-based losses do not account for absolute scale. Consequently, the depth and egomotion predictions are only accurate up to a scale factor. Our aim is to resolve these issues with our hybrid approach that, in contrast to the feature-fusion approach, combines visual and inertial information in a probabilistic manner and utilizes domain knowledge when appropriate.

7.1.3 Hybrid Approaches to VIO

Differentiable filters (DFs) have been proposed as a way to impose prior knowledge on the network structure by combining *perception* (i.e., through a measurement model that maps sensory observations to the state) and *prediction* (i.e., through a process model that determines how the state changes over time) in a Bayesian manner (Haarnoja et al., 2016; Kloss et al., 2021). Being fully differentiable, DFs can be trained end-to-end to produce uncertainty-aware measurement and process models that account for sensor noise characteristics. The DF is particularly useful for replacing (brittle) handcrafted models with networks that directly map high dimensional, nonlinear measurements (e.g., raw images) onto the state. The hybrid DF framework has been used in applications such as camera relocalization (Zhou et al., 2020), object tracking (Haarnoja et al., 2016), and VIO. For VIO, Chen et al. (2021b) learn noise covariance matrices for the process and measurement models within a system based on a differentiable unscented Kalman filter. Chen et al. (2021a) learn uncertainty-aware process and measurement models for a linear Kalman filter that maintains an estimate of a high-dimensional latent state used for VIO. Li and Waslander (2020) present a DF that combines a classical IMU-based process model with a learned relative pose measurement model. Their network is trained end-to-end by minimizing a pose supervision loss. We extend the approach of Li and Waslander (2020) by using a DF with a relative pose model, but we train our network with a photometric reconstruction loss. We show that the system accuracy is improved by using self-supervised learning for end-to-end training. To the best of our knowledge, we are the first to train a differentiable filter in a fully self-supervised manner.

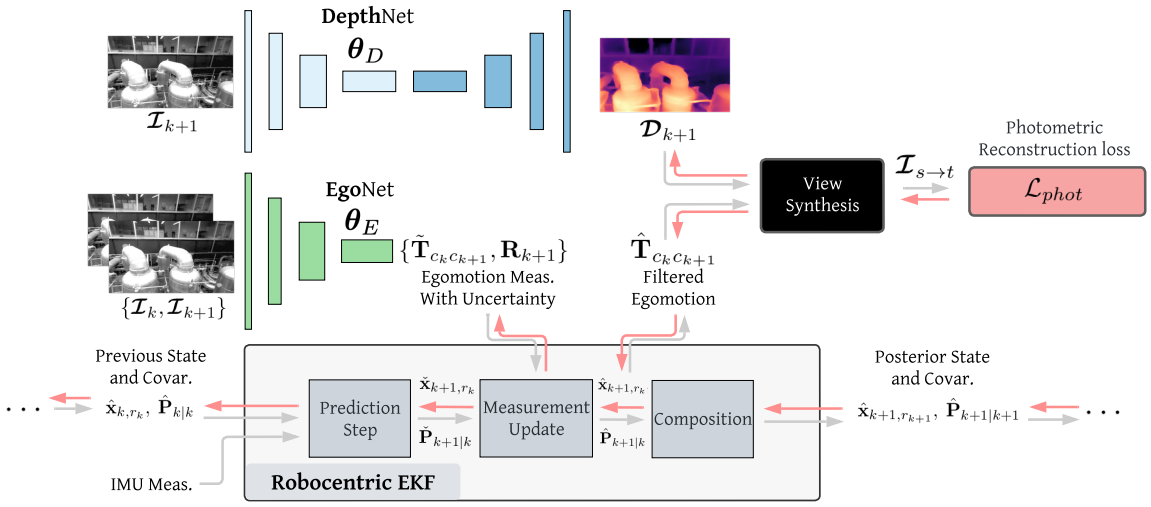


Figure 7.1: Overview of our hybrid method that combines an IMU-based process model with a learned relative pose measurement model through a robocentric EKF. Our self-supervised formulation can train this system end-to-end by minimizing a photometric reconstruction loss. Notably, we pass the egomotion predictions into the filter, and then use the a posteriori egomotion estimate to compute the photometric reconstruction loss.

7.2 Methodology

Our hybrid approach to VIO, illustrated in Figure 7.1, augments the learned SfM system with a robocentric EKF back-end. By doing so, the photometric reconstruction loss is computed with the refined, a posteriori egomotion estimate, rather than the direct network output. This posterior estimate, notably, is a function of the inertial measurements used to propagate the state via the process model. To properly incorporate the egomotion measurements in the filter, we additionally learn a heteroscedastic measurement noise covariance model that produces a covariance matrix along with every egomotion measurement. Since the filter structure is fully differentiable, the whole system can be trained end-to-end by minimizing the photometric reconstruction loss. Notably, the EKF provides a temporal connection between images within a sequence, which results in the system being a form of recurrent neural network. This temporal connection allows the egomotion network to learn how to inflate the measurement covariance to mitigate errors at future timesteps.

In Section 7.2.1, we introduce the robocentric EKF framework that we use within our hybrid VIO system. In Section 7.2.2, we discuss how our learned SfM system from the previous chapter can be incorporated as an uncertainty-aware measurement model within the EKF. Finally, in Section 7.2.3, we discuss how this system—the robocentric EKF with a learned, uncertainty-aware measurement model—can be trained end-to-end using the standard self-supervised losses from the learned SfM pipeline.

7.2.1 Robocentric EKF Formulation

Our robocentric EKF formulation is based on the approach from Li (2020). Our decision to use this structure is motivated by the presence of a relative pose within the robocentric state, which facilitates the inclusion of a relative pose measurement model. This allows us to incorporate our egomotion network (which produces relative pose measurements) as a data-driven measurement model with ease.[3]

[3]This is also an alternative to stochastic cloning (Roumeliotis and Burdick, 2002), which is an another technique that has been proposed for incorporating relative pose measurements.

The robocentric state, $\mathbf{x}_{\tau,r_k}$, has the following components (in set notation to accommodate the rotation matrices) at the latest camera measurement timestep, k, and the latest IMU measurement timestep, τ,

$$\mathbf{x}_{\tau,r_k} = \left\{ \mathbf{C}_{r_k i}, \quad \mathbf{r}_{r_k}^{ir_k}, \quad \mathbf{g}_{r_k} \quad | \quad \mathbf{C}_{r_k v_\tau}, \quad \mathbf{r}_{r_k}^{v_\tau r_k}, \quad \mathbf{v}_{v_\tau}^{v_\tau i}, \quad \mathbf{b}_{\omega,\tau}, \quad \mathbf{b}_{a,\tau} \right\}, \tag{7.1}$$

where the vertical bar separates the *robot* state (left) from the *IMU* state (right). The robot state consists of the absolute pose, $\{\mathbf{C}_{r_k i}, \mathbf{r}_{r_k}^{ir_k}\}$, at the most recent camera timestep, k (acquired at time t_k), with respect to the global reference frame, $\underrightarrow{\mathcal{F}}_i$. Further, the robot state includes the gravity vector, $\mathbf{g}_{r_k} = \mathbf{C}_{r_k i}\mathbf{g}_i$, expressed in $\underrightarrow{\mathcal{F}}_{r_k}$, which is the robocentric reference frame aligned with the robot pose at time t_k. In the robocentric formulation, the robot state and the robocentric reference frame are held fixed at the most recent camera measurement timestep while the IMU state is propagated using the high-rate IMU measurements. The IMU state contains the *relative* pose of the vehicle at the discrete timestep τ, with $t_\tau \in [t_k, t_{k+1}]$. This relative pose, $\{\mathbf{C}_{r_k v_\tau}, \mathbf{r}_{r_k}^{v_\tau r_k}\}$, relates the current vehicle frame, $\underrightarrow{\mathcal{F}}_{v_\tau}$ to the most recent robocentric frame $\underrightarrow{\mathcal{F}}_{r_k}$. Further, the IMU state includes the velocity of the IMU, $\mathbf{v}_{v_\tau}^{v_\tau i}$, and the IMU bias terms, $\mathbf{b}_{\omega,\tau}$ and $\mathbf{b}_{a,\tau}$.

The robocentric EKF is composed of three primary steps: the prediction step, the measurement update step, and the composition step. Figure 7.1 depicts how the robocentric state is updated throughout these steps. Starting with an initial state, $\hat{\mathbf{x}}_{k,r_k}$, the prediction step propagates the state to $\check{\mathbf{x}}_{k+1,r_k}$ using IMU measurements and a nonlinear process model based on Euler integration. The robot state, however, is unchanged, and remains aligned with $\underrightarrow{\mathcal{F}}_{r_k}$. Notably, after propagating the IMU state through the process model the relative pose within the IMU state becomes $\{\check{\mathbf{C}}_{r_k v_{k+1}}, \check{\mathbf{r}}_{r_k}^{v_{k+1} r_k}\}$. This pose directly represents the egomotion of the vehicle between adjacent camera measurements. Therefore, at this point in time, visual odometry or egomotion measurements (e.g., from the egomotion network within our learned SfM system) can be directly incorporated into the system within the measurement update step. Applying the measurement update produces the a posteriori state, $\hat{\mathbf{x}}_{k+1,r_k}$. Again, the robot state remains at $\underrightarrow{\mathcal{F}}_{r_k}$. Finally, a composition step shifts the robocentric frame from $\underrightarrow{\mathcal{F}}_{r_k}$ to $\underrightarrow{\mathcal{F}}_{r_{k+1}}$, and the state becomes $\hat{\mathbf{x}}_{k+1,r_{k+1}}$. This process is repeated as new measurements are received. Further details for the robocentric formulation can be found in Li (2020); Huai and Huang (2018).

Prediction Step

The prediction step propagates the IMU state from t_k to t_{k+1} using a nonlinear process model based on Euler integration of the received IMU measurements between subsequent camera frames. The IMU state covariance is also propagated during this step. The continuous-time dynamics of the true IMU states with respect to $\underrightarrow{\mathcal{F}}_{r_k}$, using the IMU measurement model from Section 2.3.1, are

$$\begin{aligned}
\dot{\mathbf{C}}_{r_k v_\tau} &= \mathbf{C}_{r_k v_\tau}(\boldsymbol{\omega}_{v_\tau}^{v_\tau i})^\wedge, \\
&= \mathbf{C}_{r_k v_\tau}(\boldsymbol{\omega}_{m,\tau} - \mathbf{b}_{\omega,\tau} - \mathbf{n}_{\omega,\tau})^\wedge, \\
\dot{\mathbf{r}}_{r_k}^{v_\tau r_k} &= \mathbf{C}_{r_k v_\tau}\mathbf{v}_{v_\tau}^{v_\tau i}, \\
\dot{\mathbf{v}}_{v_\tau}^{v_\tau i} &= \mathbf{a}_{v_\tau}^{v_\tau i} - (\boldsymbol{\omega}_{v_\tau}^{v_\tau i})^\wedge \mathbf{v}_{v_\tau}^{v_\tau i}, \\
&= (\mathbf{a}_{m,\tau} - \mathbf{C}_{v_\tau r_k}\mathbf{g}_{r_k} - \mathbf{b}_{a,\tau} - \mathbf{w}_{a,\tau}) - (\boldsymbol{\omega}_{m,\tau} - \mathbf{b}_{\omega,\tau} - \mathbf{w}_{\omega,\tau})^\wedge \mathbf{v}_{v_\tau}^{v_\tau i}, \\
\dot{\mathbf{b}}_{\omega,\tau} &= \mathbf{w}_{b_\omega,\tau}, \\
\dot{\mathbf{b}}_{a,\tau} &= \mathbf{w}_{b_a,\tau}.
\end{aligned} \tag{7.2}$$

Note that the dynamics for the robot state are zero, since $\underrightarrow{\mathcal{F}}_{r_k}$ is stationary during the prediction step. Although the other terms in Equation (7.2) are straighforward, the derivation for $\dot{\mathbf{v}}_{v_\tau}^{v_\tau i}$ is

$$\begin{aligned}
\mathbf{v}_{v_\tau}^{v_\tau i} &= \mathbf{C}_{r_k v_\tau}^\top \mathbf{v}_{r_k}^{v_\tau i}, \\
\dot{\mathbf{v}}_{v_\tau}^{v_\tau i} &= \mathbf{C}_{r_k v_\tau}^\top \dot{\mathbf{v}}_{r_k}^{v_\tau i} + \dot{\mathbf{C}}_{r_k v_\tau}^\top \mathbf{v}_{r_k}^{v_\tau i}, \\
&= \mathbf{C}_{r_k v_\tau}^\top \mathbf{a}_{r_k}^{v_\tau i} + (\mathbf{C}_{r_k v_\tau}(\boldsymbol{\omega}_{v_\tau}^{v_\tau i})^\wedge)^\top \mathbf{v}_{r_k}^{v_\tau i}, \\
&= \mathbf{a}_{v_\tau}^{v_\tau i} - (\boldsymbol{\omega}_{v_\tau}^{v_\tau i})^\wedge \mathbf{C}_{r_k v_\tau}^\top \mathbf{v}_{r_k}^{v_\tau i}, \\
&= \mathbf{a}_{v_\tau}^{v_\tau i} - (\boldsymbol{\omega}_{v_\tau}^{v_\tau i})^\wedge \mathbf{v}_{v_\tau}^{v_\tau i}.
\end{aligned} \tag{7.3}$$

Similar to the ESKF presented in Chapter 3, the continuous-time state dynamics are separated into the nominal and error state dynamics, $\dot{\bar{\mathbf{x}}}$ and $\delta\dot{\mathbf{x}}$. The nominal IMU state dynamics are

$$\begin{aligned}
\dot{\bar{\mathbf{C}}}_{r_k v_\tau} &= \bar{\mathbf{C}}_{r_k v_\tau}(\boldsymbol{\omega}_{m,\tau} - \bar{\mathbf{b}}_{\omega,\tau})^\wedge, \\
\dot{\bar{\mathbf{r}}}_{r_k}^{v_\tau r_k} &= \bar{\mathbf{C}}_{r_k v_\tau} \bar{\mathbf{v}}_{v_\tau}^{v_\tau i}, \\
\dot{\bar{\mathbf{v}}}_{v_\tau}^{v_\tau i} &= (\mathbf{a}_{m,\tau} - \bar{\mathbf{C}}_{v_\tau r_k} \bar{\mathbf{g}}_{r_k} - \bar{\mathbf{b}}_{a,\tau}) - (\boldsymbol{\omega}_{m,\tau} - \bar{\mathbf{b}}_{\omega,\tau})^\wedge \bar{\mathbf{v}}_{v_\tau}^{v_\tau i}, \\
\dot{\bar{\mathbf{b}}}_{\omega,\tau} &= \mathbf{0}, \\
\dot{\bar{\mathbf{b}}}_{a,\tau} &= \mathbf{0}.
\end{aligned} \tag{7.4}$$

Through integration of these equations, the nominal state process model is determined. This process model, $\check{\mathbf{x}}_{\tau,r_k} = f(\hat{\mathbf{x}}_{k,r_k}, \mathbf{0})$, propagates the IMU state from t_k to t_τ, while keeping the robot state fixed:

$$\begin{aligned}
\check{\mathbf{C}}_{r_k v_\tau} &= \int_{t_k}^{t_\tau} \check{\mathbf{C}}_{r_k v_s} \left(\boldsymbol{\omega}_{m,s} - \hat{\mathbf{b}}_{\omega,k}\right)^\wedge ds, \\
\check{\mathbf{v}}_{v_\tau}^{v_\tau i} &= \check{\mathbf{C}}_{r_k v_\tau}^\top \left(\hat{\mathbf{v}}_{r_k}^{v_k i} - \hat{\mathbf{g}}_{r_k} \Delta t + \int_{t_k}^{t_\tau} \check{\mathbf{C}}_{r_k v_s} \left(\mathbf{a}_{m,s} - \hat{\mathbf{b}}_{a,k}\right) ds \right), \\
\check{\mathbf{r}}_{r_k}^{v_\tau r_k} &= \hat{\mathbf{v}}_{r_k}^{v_k i} \Delta t - \frac{1}{2} \hat{\mathbf{g}}_{r_k} \Delta t^2 + \int_{t_k}^{t_\tau} \int_{t_k}^{s} \check{\mathbf{C}}_{r_k v_\mu} (\mathbf{a}_{m,\mu} - \hat{\mathbf{b}}_{a,k}) \, d\mu ds.
\end{aligned} \tag{7.5}$$

In these expressions, $\Delta t = t_\tau - t_k$. Discrete integration of the process model is performed using Euler's method (see Section 2.3.1 for details). We omit the bias terms here, since they are held constant during this interval. The prediction step is repeated until a new camera measurement arrives at t_{k+1}, at which point the state becomes $\check{\mathbf{x}}_{k+1,r_k}$.[4]

In parallel, the error state process model is used for covariance propagation. The error state vector for our system is

$$\delta\mathbf{x}_{\tau,r_k} = \begin{bmatrix} \delta\boldsymbol{\phi}_{r_k i}^\top & \delta\mathbf{r}_{r_k}^{i r_k \top} & \delta\mathbf{g}_{r_k}^\top & | & \delta\boldsymbol{\phi}_{r_k v_\tau}^\top & \delta\mathbf{r}_{r_k}^{v_\tau r_k \top} & \delta\mathbf{v}_{v_\tau}^{v_\tau i \top} & \delta\mathbf{b}_{\omega,\tau}^\top & \delta\mathbf{b}_{a,\tau}^\top \end{bmatrix}^\top, \tag{7.6}$$

where $\delta\boldsymbol{\phi}_{r_k i}$ and $\delta\boldsymbol{\phi}_{r_k v_\tau}$ are the orientation error states parameterized as vectors in the Lie algebra. Herein, we provide the linearized, continuous-time error state dynamics,

$$\delta\dot{\mathbf{x}}_{\tau,r_k} = \mathbf{F}_\tau \delta\mathbf{x}_{\tau,r_k} + \mathbf{G}_\tau \mathbf{w}_\tau, \tag{7.7}$$

[4] We acknowledge the abuse in notation when defining the state as $\mathbf{x}_{k+1,r_k}$ at t_{k+1}. This notation implies that $\tau = k+1$ when the new camera measurement arrives, when in reality τ has been propagated many times in between the camera timesteps k and $k+1$. Nevertheless, we use this notation to be consistent with Li (2020) and Huai and Huang (2018).

and refer to Li (2020) for a full derivation. In this expression, $\mathbf{w}_\tau = \begin{bmatrix}\mathbf{w}_{\omega,\tau}^\top & \mathbf{w}_{b_\omega,\tau}^\top & \mathbf{w}_{a,\tau}^\top & \mathbf{w}_{b_a,\tau}^\top\end{bmatrix}^\top$ is the vector of noise terms with the noise covariance matrix $\mathbf{Q}_\tau = \text{diag}(\begin{bmatrix}\sigma_\omega^2\mathbf{1}_{1\times3} & \sigma_{b_\omega}^2\mathbf{1}_{1\times3} & \sigma_a^2\mathbf{1}_{1\times3} & \sigma_{b_a}^2\mathbf{1}_{1\times3}\end{bmatrix})$ and the matrices $\mathbf{F}_\tau$ and $\mathbf{G}_\tau$ are

$$\mathbf{F}_\tau = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & -(\boldsymbol{\omega}_{v_\tau}^{v_\tau i})^\wedge & \mathbf{0} & \mathbf{0} & -\mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & -\bar{\mathbf{C}}_{r_k v_\tau}(\bar{\mathbf{v}}_{v_\tau}^{v_\tau i})^\wedge & \mathbf{0} & \bar{\mathbf{C}}_{r_k v_\tau} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -\bar{\mathbf{C}}_{r_k v_\tau}^\top & -(\bar{\mathbf{C}}_{r_k v_\tau}^\top \bar{\mathbf{g}}_{r_k})^\wedge & \mathbf{0} & -(\boldsymbol{\omega}_{v_\tau}^{v_\tau i})^\wedge & -(\bar{\mathbf{v}}_{v_\tau}^{v_\tau i})^\wedge & -\mathbf{I}_3 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}, \tag{7.8}$$

$$\mathbf{G}_\tau = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ -\mathbf{I}_3 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ -(\bar{\mathbf{v}}_{v_\tau}^{v_\tau i})^\wedge & \mathbf{0} & -\mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_3 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_3 \end{bmatrix}. \tag{7.9}$$

The solution to the differential equation in Equation (7.7), when assuming that $\mathbf{F}_\tau$ is constant between the time interval $\delta t = t_{\tau+1} - t_\tau$, is

$$\begin{aligned} \delta\mathbf{x}_{\tau+1,r_k} &= \boldsymbol{\Phi}_{\tau+1,\tau}\delta\mathbf{x}_{\tau,r_k} + \int_\tau^{\tau+1} \boldsymbol{\Phi}_{\tau+1,s}\mathbf{G}_s\mathbf{w}_s ds, \\ &= \boldsymbol{\Phi}_{\tau+1,\tau}\delta\mathbf{x}_{\tau,r_k} + \mathbf{w}'_\tau. \end{aligned} \tag{7.10}$$

The error state transition matrix, $\boldsymbol{\Phi}_{\tau+1,\tau}$, is defined as

$$\boldsymbol{\Phi}_{\tau+1,\tau} = \exp\left(\int_{t_\tau}^{t_{\tau+1}} \mathbf{F}_s\, ds\right) \approx \mathbf{I}_{24} + \mathbf{F}_s\delta t + \mathbf{F}_s^2\delta t^2. \tag{7.11}$$

As discussed in Farrell (2008), the noise covariance propagation through Equation (7.10) can be approximated as

$$\begin{aligned} \check{\mathbf{P}}_{\tau+1|k} &\approx \boldsymbol{\Phi}_{\tau+1,\tau}\check{\mathbf{P}}_{\tau|k}\boldsymbol{\Phi}_{\tau+1,\tau}^\top + \mathbf{G}_\tau\mathbf{Q}_\tau\mathbf{G}_\tau^\top\delta t, \\ &\approx \boldsymbol{\Phi}_{\tau+1,\tau}\check{\mathbf{P}}_{\tau|k}\boldsymbol{\Phi}_{\tau+1,\tau}^\top + \mathbf{Q}'_\tau. \end{aligned} \tag{7.12}$$

Note that the subscript $\tau|k$ denotes the covariance of the IMU state at τ, and the robot state at k. Since Equation (7.12) only propagates the IMU state covariance, the first index is updated while the robot state covariance remains fixed. The robot covariance will be updated after the measurement update during the composition step.

Measurement Update Step

The measurement update is applied when each new camera measurement is received. When the state is expressed with respect to $\underrightarrow{\mathcal{F}}_{r_k}$, the measurement is applied to the predicted state, $\check{\mathbf{x}}_{k+1,r_k}$, at time t_{k+1}. As we noted earlier, the predicted IMU state at this timestep contains the vehicle/IMU pose at t_{k+1}, expressed with respect to the robot pose at t_k. Therefore, a relative pose measurement model can be used without the need for stochastic cloning. In this section, we introduce the general relative pose measurement model, and in Section 7.2.2, we will discuss how our tightly coupled, learned SfM system is used to provide these relative pose measurements.

To facilitate visual egomotion measurements, the relative pose measurement model, $h(\mathbf{x}_{k+1,r_k}, \mathbf{0})$, transforms the IMU pose to be expressed within the camera reference frame, $\underrightarrow{\mathcal{F}}_{c_k}$, instead of $\underrightarrow{\mathcal{F}}_{r_k}$. This transformation is

$$\begin{aligned} \mathbf{r}_{c_k}^{c_{k+1}c_k} &= \mathbf{C}_{r_k c_k}^\top \mathbf{C}_{r_k v_{k+1}} \mathbf{r}_{v_{k+1}}^{c_{k+1}v_{k+1}} + \mathbf{C}_{r_k c_k}^\top \left(\mathbf{r}_{r_k}^{v_{k+1}r_k} - \mathbf{r}_{r_k}^{c_k v_k} \right), \\ \mathbf{C}_{c_k c_{k+1}} &= \mathbf{C}_{r_k c_k}^\top \mathbf{C}_{r_k v_{k+1}} \mathbf{C}_{v_{k+1} c_{k+1}}, \end{aligned} \tag{7.13}$$

and uses the known extrinsic transformation between the camera and IMU, $\{\mathbf{C}_{v_k c_k}, \mathbf{r}_{v_k}^{c_k v_k}\}$.[5] The measurement residual, $\boldsymbol{\epsilon}_{k+1}$, compares the current egomotion measurement, $\{\tilde{\mathbf{C}}_{c_k c_{k+1}}, \tilde{\mathbf{r}}_{c_k}^{c_{k+1}c_k}\}$, with the transformed pose and is evaluated at the predicted state:

$$\boldsymbol{\epsilon}_{k+1} = \begin{bmatrix} \boldsymbol{\epsilon}_{\phi,k+1} \\ \boldsymbol{\epsilon}_{r,k+1} \end{bmatrix} = \begin{bmatrix} \log\left(\tilde{\mathbf{C}}_{c_k c_{k+1}} \check{\mathbf{C}}_{c_k c_{k+1}}^\top \right)^\vee \\ \tilde{\boldsymbol{r}}_{c_k}^{c_{k+1}c_k} - \check{\mathbf{r}}_{c_k}^{c_{k+1}c_k} \end{bmatrix} + \mathbf{n}_{k+1}. \tag{7.14}$$

We assume the measurement noise, $\mathbf{n}_{k+1} = \begin{bmatrix} \mathbf{n}_{\phi,k+1}^\top & \mathbf{n}_{r,k+1}^\top \end{bmatrix}^\top$, is normally distributed through $\mathbf{n}_{k+1} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_{k+1})$. This heteroscedastic measurement covariance model can be learned; we employ this strategy within our system and discuss it further in Section 7.2.2.

To perform the measurement update within the EKF, the measurement Jacobian $\mathbf{H}_{k+1}$ is required and is found by differentiating Equation (7.14) with respect to $\delta\check{\mathbf{x}}_{k+1,r_k}$. The measurement Jacobian is defined as

$$\mathbf{H}_{k+1} = \begin{bmatrix} \mathbf{0}_{3\times 9} & -\mathbf{C}_{v_k c_k}^\top \mathbf{J}_\ell(-\check{\phi}_{r_k v_{k+1}})^{-1} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 9} \\ \mathbf{0}_{3\times 9} & \mathbf{C}_{v_k c_k}^\top \check{\mathbf{C}}_{r_k v_{k+1}} \mathbf{r}_{v_k}^{c_k v_k \wedge} & -\mathbf{C}_{v_k c_k}^\top & \mathbf{0}_{3\times 9} \end{bmatrix}. \tag{7.15}$$

We refer to Appendix D for the full derivation of the measurement Jacobian. Using the derived equations, the EKF measurement update is

$$\begin{aligned} \mathbf{K}_{k+1} &= \check{\mathbf{P}}_{k+1|k} \mathbf{H}_{k+1}^\top \left(\mathbf{H}_{k+1} \check{\mathbf{P}}_{k+1|k} \mathbf{H}_{k+1}^\top + \mathbf{R}_{k+1} \right)^{-1}, \\ \hat{\mathbf{P}}_{k+1|k} &= \left(\mathbf{I}_{24} - \mathbf{K}_{k+1} \mathbf{H}_{k+1} \right) \check{\mathbf{P}}_{k+1|k}, \\ \delta\hat{\mathbf{x}}_{k+1,r_k} &= \mathbf{K}_{k+1} \boldsymbol{\epsilon}_{k+1}. \end{aligned} \tag{7.16}$$

Finally, the a posteriori state $\hat{\mathbf{x}}_{k+1,r_k}$ is produced by injecting the error state estimate $\delta\hat{\mathbf{x}}_{k+1,r_k}$ into the nominal

[5] Since the extrinsic transformation is constant at all timesteps, and $\underrightarrow{\mathcal{F}}_{r_k}$ is aligned with $\underrightarrow{\mathcal{F}}_{v_k}$ when a camera measurement is received, Equation (7.13) is based on the following relations being equal: $\{\mathbf{C}_{v_k c_k}, \mathbf{r}_{v_k}^{c_k v_k}\} = \{\mathbf{C}_{r_k c_k}, \mathbf{r}_{r_k}^{c_k r_k}\} = \{\mathbf{C}_{v_{k+1} c_{k+1}}, \mathbf{r}_{v_{k+1}}^{c_{k+1} v_{k+1}}\} = \{\mathbf{C}_{r_{k+1} c_{k+1}}, \mathbf{r}_{r_{k+1}}^{c_{k+1} r_{k+1}}\}$.

state through $\hat{\mathbf{x}}_{k+1,r_k} = \check{\mathbf{x}}_{k+1,r_k} \oplus \delta\hat{\mathbf{x}}_{k+1,r_k}$. The error-state injection for each component of the state is

$$\begin{aligned}
\hat{\mathbf{C}}_{r_k i} &= \check{\mathbf{C}}_{r_k i} \exp\left(\delta\hat{\boldsymbol{\phi}}_{r_k i}^{\wedge}\right), \\
\hat{\mathbf{r}}_{r_k}^{i r_k} &= \check{\mathbf{r}}_{r_k}^{i r_k} + \delta\hat{\mathbf{r}}_{r_k}^{i r_k}, \\
\hat{\mathbf{g}}_{r_k} &= \check{\mathbf{g}}_{r_k} + \delta\hat{\mathbf{g}}_{r_k}, \\
\hat{\mathbf{C}}_{r_k v_{k+1}} &= \check{\mathbf{C}}_{r_k v_{k+1}} \exp\left(\delta\hat{\boldsymbol{\phi}}_{r_k v_{k+1}}^{\wedge}\right), \\
\hat{\mathbf{r}}_{r_k}^{v_{k+1} r_k} &= \check{\mathbf{r}}_{r_k}^{v_{k+1} r_k} + \delta\hat{\mathbf{r}}_{r_k}^{v_{k+1} r_k}, \\
\hat{\mathbf{v}}_{v_{k+1}}^{v_{k+1} i} &= \check{\mathbf{v}}_{v_{k+1}}^{v_{k+1} i} + \delta\hat{\mathbf{v}}_{v_{k+1}}^{v_{k+1} i}, \\
\hat{\mathbf{b}}_{\omega,k+1} &= \check{\mathbf{b}}_{\omega,k+1} + \delta\hat{\mathbf{b}}_{\omega,k+1}, \\
\hat{\mathbf{b}}_{a,k+1} &= \check{\mathbf{b}}_{a,k+1} + \delta\hat{\mathbf{b}}_{a,k+1}.
\end{aligned} \tag{7.17}$$

Note that the corrections applied to the first two terms (the robot pose) are zero, since these terms are unobservable with a relative pose measurement model. At test time, we use an IEKF to iteratively recompute the terms in Equation (7.17) as the measurement Jacobian is updated.

Composition Step

In the robocentric formulation, the composition step shifts the robot state forward from $\underrightarrow{\mathcal{F}}_{r_k}$ to $\underrightarrow{\mathcal{F}}_{r_{k+1}}$, (i.e., the state is updated from $\hat{\mathbf{x}}_{k+1,r_k}$ to $\hat{\mathbf{x}}_{k+1,r_{k+1}}$). Specifically, the composition step compounds the (relative) IMU pose with the robot pose and updates the gravity vector direction to be expressed within $\underrightarrow{\mathcal{F}}_{r_{k+1}}$. Then, the IMU pose, which is expressed relative to the robot pose, is reset to identity. The composition step is summarized as

$$\begin{aligned}
\hat{\mathbf{C}}_{r_{k+1} i} &= \hat{\mathbf{C}}_{r_k v_{k+1}}^{\top} \hat{\mathbf{C}}_{r_k i}, \\
\hat{\mathbf{r}}_{r_{k+1}}^{i r_{k+1}} &= \hat{\mathbf{C}}_{r_k v_{k+1}}^{\top} \left(\hat{\mathbf{r}}_{r_k}^{i r_k} - \hat{\mathbf{r}}_{r_k}^{v_{k+1} r_k}\right), \\
\hat{\mathbf{g}}_{r_{k+1}} &= \hat{\mathbf{C}}_{r_k v_{k+1}}^{\top} \hat{\mathbf{g}}_{r_k}, \\
\hat{\mathbf{C}}_{r_{k+1} v_{k+1}} &= \mathbf{I}_3, \\
\hat{\mathbf{r}}_{r_{k+1}}^{v_{k+1} r_{k+1}} &= \mathbf{0}_{3\times 1}, \\
\hat{\mathbf{v}}_{v_{k+1}}^{v_{k+1} i} &= \hat{\mathbf{v}}_{v_{k+1}}^{v_{k+1} i}, \\
\hat{\mathbf{b}}_{\omega,k+1} &= \hat{\mathbf{b}}_{\omega,k+1}, \\
\hat{\mathbf{b}}_{a,k+1} &= \hat{\mathbf{b}}_{a,k+1}.
\end{aligned} \tag{7.18}$$

The state covariance is accordingly updated through

$$\hat{\mathbf{P}}_{k+1|k+1} = \mathbf{U}_{k+1}\hat{\mathbf{P}}_{k+1|k}\mathbf{U}_{k+1}^{\top}, \tag{7.19}$$

$$\mathbf{U}_{k+1} = \frac{\partial \delta\hat{\mathbf{x}}_{k+1,r_{k+1}}}{\partial \delta\hat{\mathbf{x}}_{k+1,r_k}} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0} & \mathbf{0} & -\hat{\mathbf{C}}_{r_{k+1}i}^{\top} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{C}}_{r_k v_{k+1}}^{\top} & \mathbf{0} & (\hat{\mathbf{r}}_{r_{k+1}}^{ir_{k+1}})^{\wedge} & -\hat{\mathbf{C}}_{r_k v_{k+1}}^{\top} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \hat{\mathbf{C}}_{r_k v_{k+1}}^{\top} & \hat{\mathbf{g}}_{r_{k+1}}^{\wedge} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_3 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_3 \end{bmatrix}. \tag{7.20}$$

Essentially, this operation moves the uncertainty from the (relative) IMU pose to the robot state and then resets the IMU pose covariance to zero as the pose is reset to identity. The velocity and bias covariance terms remain unchanged since the composition step does not impact these values.

7.2.2 The Learned Relative Pose Measurement Model

We produce a relative pose measurement using the learned SfM framework that we introduced in the previous chapters. This measurement is applied during the measurement update step, to correct the predicted state, $\check{\mathbf{x}}_{k+1,r_k}$. For a target image, $\boldsymbol{\mathcal{I}}_{k+1}$, and a source image, $\boldsymbol{\mathcal{I}}_k$, the learned SfM predictions are

$$\boldsymbol{\mathcal{D}}_{c_{k+1}} = f_{\boldsymbol{\theta}_D}(\boldsymbol{\mathcal{I}}_{k+1}), \tag{7.21}$$

$$\tilde{\boldsymbol{\xi}}_{c_k c_{k+1}} = f_{\boldsymbol{\theta}_E}(\boldsymbol{\mathcal{I}}_k, \boldsymbol{\mathcal{I}}_{k+1}). \tag{7.22}$$

Note that our predictions are expressed with respect to the camera frame, $\underrightarrow{\mathcal{F}}_c$, while our state is expressed with respect to $\underrightarrow{\mathcal{F}}_v$. The exponential map is used to produce the $\mathrm{SE}(3)$ IMU pose measurement through $\tilde{\mathbf{T}}_{c_k c_{k+1}} = \exp\left(\tilde{\boldsymbol{\xi}}_{c_k c_{k+1}}^{\wedge}\right)$. Then, from $\tilde{\mathbf{T}}_{c_k c_{k+1}}$, we can extract the relative translation and rotation measurements, $\{\tilde{\mathbf{C}}_{c_k c_{k+1}}, \tilde{\mathbf{r}}_{c_k}^{c_{k+1}c_k}\}$, and incorporate these measurements into the EKF via the Equation (7.14) measurement residual.[6]

Our depth and egomotion predictions are produced from the tightly coupled system that we presented in Chapter 6. Feedback coupling is used to iteratively update the egomotion prediction. Currently, we do not apply the test-time optimization scheme; we leave this as future work. The depth and egomotion networks are trained end-to-end with the filter by minimizing the standard self-supervised losses. The training procedure is discussed further in Section 7.2.3.

Uncertainty-Aware Measurement Model

To leverage the learned egomotion measurements, the iterative egomotion network predicts a measurement covariance matrix,

$$\mathbf{R}_{k+1} = \begin{bmatrix} \boldsymbol{\Sigma}_{\phi_{k+1}} & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \boldsymbol{\Sigma}_{r_{k+1}} \end{bmatrix}, \tag{7.23}$$

[6] In practice, we make the assumption that $\exp\left(\boldsymbol{\xi}^{\wedge}\right) \approx \mathbf{I}_4 + \boldsymbol{\xi}^{\wedge}$, which allows us to extract the relative translation measurement from $\boldsymbol{\xi}_{c_k c_{k+1}}$ directly, rather than having to compute the Jacobian, $\mathbf{J}_{\ell}$.

where $\mathbf{\Sigma}_{r_k}$ and $\mathbf{\Sigma}_{\phi_k}$ are diagonal covariance submatrices. Similar to Li and Waslander (2020), the dimensionality of the final layer of the egomotion network is increased from six to twelve in order for the network to produce the six diagonal elements of $\mathbf{R}_k$. Each of the additional six network outputs, w_i, populates the diagonal of $\mathbf{R}_k$ using

$$\sigma_i^2 = \sigma_0^2 10^{\beta \tanh(w_i)}, \tag{7.24}$$

where σ_0^2 is a base covariance and $\beta \in \mathbb{R}_{>0}$ is a control parameter. This form is chosen such that an unrestricted network output, w_i, can either increase or decrease the covariance from its original value, σ_0^2, to a new value within a fixed range. The range is determined through the selection β: since the $tanh$ output is within $[-1, 1]$, the covariance output is constrained to be within β orders of magnitude of σ_0^2.

With our feedback coupling scheme, we perform multiple forward passes through the egomotion network to iteratively update the uncertainty predictions. The prediction from each forward pass is summed together to form w_i before computing Equation (7.24). We find that iteratively updating the uncertainty predictions significantly improves performance. As additional forward passes improve the alignment of the input images, the remaining errors are more easily detected by the network. Accordingly, the network can inflate the measurement covariance if large discrepancies (i.e., misalignments of the images) are still present after the final iteration.

7.2.3 End-to-End Training with the Differentiable EKF

Our hybrid system is trained end-to-end by minimizing the same self-supervised losses that we employed in the previous two chapters, namely, the photometric reconstruction loss consisting of $\mathcal{L}_{phot}$ (with the L_1 and SSIM terms), in addition to the inverse depth smoothness loss $\mathcal{L}_S$ and the depth consistency loss $\mathcal{L}_{DC}$. We also include the pose consistency loss term, $\mathcal{L}_{PC}$ from the previous chapter. Overall, the *per-sample* loss that we use to train the system is

$$\mathcal{L} = \frac{1}{HW} \sum_{u,v} \left(\lambda_{phot} \mathcal{L}_{phot} + \lambda_S \mathcal{L}_S + \lambda_{DC} \mathcal{L}_{DC} \right) + \frac{1}{6} \sum \lambda_{PC} \mathcal{L}_{PC}, \tag{7.25}$$

and is averaged across all training samples. Note that within $\mathcal{L}_{phot}$, we use the automasking and minimum reprojection techiques, along with the self-discovered mask, to remove and/or downweight the unstable image regions.

Owing to the addition of the EKF, several modifications are made to the baseline training procedure of the learned SfM system. The most notable difference is that we use the a posteriori egomotion estimate to compute the reconstruction loss, instead of using the egomotion network prediction directly. This modification is illustrated in Figure 7.1. We outline three other primary differences next.

Sequential Training. First, to leverage the recurrent nature of the network, we extend the number of frames per sample from the standard of three (i.e., one target image and two adjacent source images) to an arbitrary length N. The longer sequence length is important for training the uncertainty model, as the network must learn to inflate the covariance for erroneous measurements that negatively impact future state estimates.[7] By reducing the impact of erroneous (e.g., corrupted or outlier) measurements, future estimates will be more accurate and the reconstruction loss will be reduced. We visualize a training sample in Figure 7.2 consisting of ten timesteps. Here,

[7] Importantly, all of the components of the EKF are differentiable, allowing for gradients to flow from the loss and into the networks (at the current timestep and at previous timesteps) during backpropagation. This is crucial for training the uncertainty model.

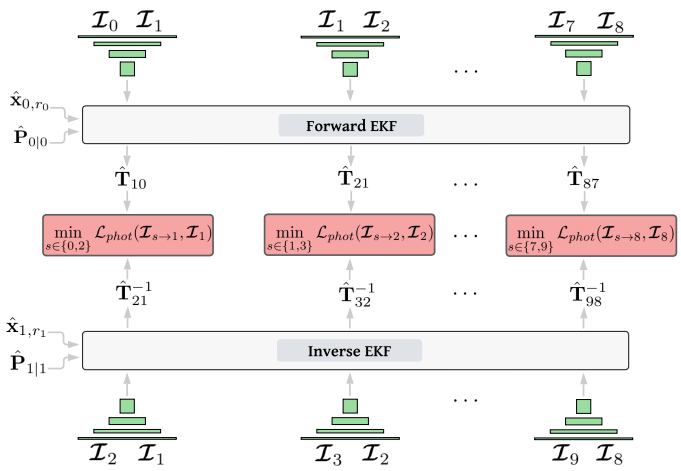


Figure 7.2: Overview of our training procedure. We use the egomotion estimates from a forward and inverse EKF to produce the two target image reconstructions required for the minimum reprojection loss. Note that the IMU measurements, depth predictions, and view synthesis module are omitted from this diagram.

the EKF propagates from the initial state $\mathbf{x}_0$ and incorporates the learned egomotion measurements. Then, the a posteriori IMU pose is used to compute the minimum reprojection loss after each timestep. Since the minimum reprojection loss requires two image reconstructions, we generate 'forward' and 'inverse' egomotion estimates. The 'forward' egomotion prediction produces $\mathcal{I}_{t-1\to t}$, and the 'inverse' prediction (i.e., the source-target image inputs are swapped) produces $\mathcal{I}_{t+1\to t}$. As demonstrated in Figure 7.2, we employ two separate EKFs to produce these poses during training.

Pose Initialization Scheme. The second difference is the requirement to initialize the training samples with an accurate estimate of the state (and state covariance) at the first timestep. When using supervised training, this condition is trivial to enforce: the filter is initialized with the ground truth pose. Since ground truth information is not available in the self-supervised formulation, we initialize the training samples using the most recent pose estimate from our hybrid VIO system instead. These pose estimates are generated for all training and validation sequence frames at the start of every epoch, and remain fixed for each epoch. As training progresses, the pose estimates improve, so the initialization accuracy increases. This increase in initialization accuracy allows the training to converge further.

Scale Recovery. Lastly, we employ a scale recovery strategy that encourages the egomotion predictions from EgoNet to become metrically scaled during training. We achieve this by adding a scale parameter, λ_τ, to the end of the IMU state vector during the pose preprocessing stage. The continuous time dynamics model for the scale is $\dot{\lambda}_\tau = 0$ and error state is $\delta\dot{\lambda}_\tau = 0$. With this scale factor augmented to the state, the updated translation measurement residual is

$$\boldsymbol{\epsilon}_{r,k} = \tilde{\boldsymbol{r}}_{c_k}^{c_{k+1}c_k} - \check{\lambda}_{k+1}\check{\mathbf{r}}_{c_k}^{c_{k+1}c_k} + \mathbf{n}_{r,k+1}. \tag{7.26}$$

The inclusion of this scale parameter allows us to produce metrically scaled pose initializations for the first training epoch, despite using an *unscaled* depth and egomotion network. By including metric pose initializations during training, the scale factors of our depth and egomotion predictions naturally converge to unity. This

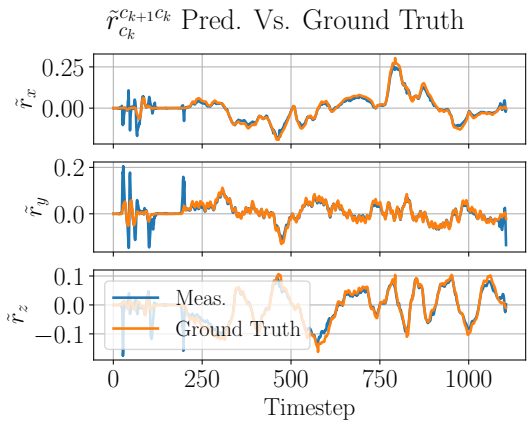


Figure 7.3: The raw translation predictions for EuRoC validation sequence `MH05`. The scale of the predictions closely matches that of the ground truth, indicating that the networks are scale-aware.

convergence happens without explicitly including the scale parameter in the state at training time; only the metric pose initializations are needed to train the scale-aware networks. We note that the scale parameter could potentially be included at test time to mitigate scale drift in novel (out-of-distribution) environments. Figure 7.3 demonstrates the scale-aware translation predictions from our fully trained egomotion network. Note that with the addition of the scale factor, the measurement Jacobian is updated to

$$\mathbf{H}_{k+1} = \begin{bmatrix} \mathbf{0}_{3\times 9} & -\mathbf{C}_{v_k c_k}^{\top} \mathbf{J}_{\ell}(-\check{\boldsymbol{\phi}}_{r_k v_{k+1}})^{-1} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 9} & \mathbf{0}_{1\times 3} \\ \mathbf{0}_{3\times 9} & \check{\lambda}_{k+1} \mathbf{C}_{v_k c_k}^{\top} \check{\mathbf{C}}_{r_k v_{k+1}} \mathbf{r}_{v_k}^{c_k v_k \wedge} & -\check{\lambda}_{k+1} \mathbf{C}_{v_k c_k}^{\top} & \mathbf{0}_{3\times 9} & -\mathbf{C}_{v_k c_k}^{\top} (\check{\boldsymbol{\phi}}_{r_k v_{k+1}}^{\wedge} \mathbf{r}_{v_k}^{c_k v_k} + \check{\mathbf{r}}_{r_k}^{v_{k+1} r_k}) \end{bmatrix}. \tag{7.27}$$

See Appendix D for a full derivation.

7.3 Experiments

In this section, we provide additional training and implementation details, along with the experimental results from evaluating our system on the EuRoC dataset.

7.3.1 Datasets and Evaluation Metrics

We train and evaluate our system on the EuRoC dataset (see Appendix A for additional details). For training, we use all sequences except `MH05`, `V103`, and `V203`, which are held out for validation (similar to Almalioglu et al. (2022)). In our experiments, the images are undistorted using the known radial-tangental lens distortion parameters and downsized to 256×448. The left and right images are treated as independent sequences to effectively double the amount of training data. For evaluation, we report the standard metrics for monocular approaches: the average translation RMSE, after $\mathrm{Sim}(3)$ alignment with ground truth, and also the rotational RMSE for some experiments.

7.3.2 Implementation Details

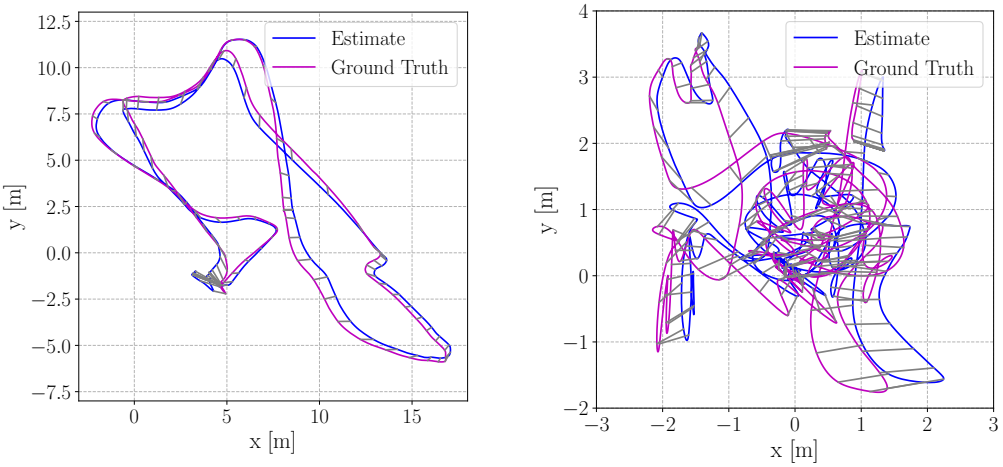
Our system is trained in PyTorch using an Nvidia Quadro RTX 8000 GPU. We use same tightly coupled depth and egomotion network structures from Chapter 6, and initialize our training with the four-iteration model trained on ScanNet. Five EgoNet iterations are applied at training and test time. The EuRoC IMU and image data used to train our system are downsampled by a factor of two to reduce the per-epoch training time and increase the perspective change between frames. The training samples consist of subsequences that are one second (i.e., ten images at ten Hz) in length. Adjacent training samples have an overlap of 0.3 s. The depth and egomotion networks are trained, in minibatches of six samples, via gradient descent (using the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$) for 25 epochs with a learning rate of $1e^{-4}$ that is halved every seven epochs. For image augmentation during training, we randomly apply brightness, contrast, saturation, and hue transformations. We also apply random horizontal flips (with $p = 0.5$). The same augmentation is applied to every image within the same training sample.[8] For our training losses, we set $\alpha = 0.15$, $\lambda_{phot} = 1$, $\lambda_S = 0.05$, $\lambda_{DC} = 0.15$, and $\lambda_{PC} = 5$. The constants for uncertainty prediction are $\beta = 4$ and $\sigma_0^2 = 1$. Our IMU noise parameters are $\sigma_\omega = 1e^{-3}$, $\sigma_a = 0.1$, $\sigma_{b_\omega} = 1e^{-5}$, and $\sigma_{b_a} = 0.01$. The covariance initialization $\mathbf{P}_{0|0} = \text{diag}(\mathbf{0}_{1\times6}, \sigma_{g_0}\mathbf{1}_{1\times3}, \mathbf{0}_{1\times6}, \sigma_{v_0}\mathbf{1}_{1\times3}, \sigma_{b_{a_0}}\mathbf{1}_{1\times3}, \sigma_{b_{\omega_0}}\mathbf{1}_{1\times3})$ used during training consists of $\sigma_{g_0} = 0.1$, $\sigma_{v_0} = 0.01$, $\sigma_{b_{a_0}} = 1$, and $\sigma_{b_{\omega_0}} = 0.1$. We use the current a posteriori estimate to initialize the first state of each training sample (see Section 7.2.3 for more details). To produce these estimates, we initialize the first state of every sequence with the ground truth information provided in the dataset. Since the IMU is stationary in these instances, we alternatively could initialize the position/velocity to zero and initialize the orientation by observing the direction of the gravity measurement from the accelerometer. Sensor biases for every training sample are initialized to zero.

We find that when using the procedure outlined above, our system can be trained end-to-end without experiencing any numerical instabilities. We are able to backpropagate gradients through the filter equations using the standard automatic differentiation method in PyTorch. This stability is in part a result of initializing the system with pretrained depth and egomotion networks. Training the networks from scratch would be more challenging when using this sequential training method, since the filter could diverge when erroneous egomotion measurements are provided early on during training.

7.3.3 Experimental Results

Table 7.1 reports the average translation RMSE, after $\text{Sim}(3)$ alignment, for various VIO algorithms operating on the EuRoC sequences. Similar to other self-supervised methods (Almalioglu et al., 2022), we report the training sequence results. We include comparisons with classical systems (ROVIO and VINS-Mono), a learning-based system (SelfVIO), and the hybrid supervised variant of the differentiable EKF from Li and Waslander (2020). Notably, our system is significantly more accurate than the supervised variant from Li and Waslander (2020), but SelfVIO yields better performance overall. We tentatively attribute this result to the adversarial loss applied in SelfVIO, which may lead to more accurate depth predictions. However, SelfVIO cannot produce metrically scaled predictions and instead relies on $\text{Sim}(3)$ alignment to recover scale. Figure 7.4 visualizes the accuracy of our system on `MH05` and `V103`. Finally, when comparing our system with ROVIO and VINS-Mono, we see that ours is unable to achieve the same level of accuracy under nominal operating conditions. There are a number of reasons why this shortcoming in performance exists. First, these methods jointly optimize depth and egomotion through bundle adjustment, while our filter only estimates egomotion (while keeping the initial depth prediction

[8] Since the IMU data cannot be augmented in a similar manner, the egomotion predictions, prior to use within the EKF, are appropriately altered to represent the egomotion of the unflipped image.



(a) Sequence MH05.

(b) Sequence V103.

Figure 7.4: Topdown trajectory estimate for two EuRoC test sequences after $\mathrm{Sim}(3)$ alignment.

Table 7.1: Translation RMSE for the EuRoC sequences (using the ROVIO and VINS-Mono results reported in Delmerico and Scaramuzza (2018)). Our self-supervised system outperforms the supervised variant by a significant margin.

Sequence	ROVIO (Bloesch et al., 2017)	VINS-Mono (Qin et al., 2018)	Self-VIO (Almalioglu et al., 2022)	EKF-VIO (Li and Waslander, 2020)	Ours
MH01	0.21	0.27	**0.19**	1.17	0.51
MH02	0.25	**0.12**	0.15	1.56	0.78
MH03	0.25	**0.13**	0.21	1.89	0.69
MH04	0.49	0.23	**0.16**	2.12	1.00
MH05†	0.52	0.35	**0.29**	1.96	0.80
V101	0.10	**0.07**	0.08	2.07	0.43
V102	0.10	0.10	**0.09**	2.20	0.61
V103†	0.14	0.13	**0.10**	2.83	0.72
V201	0.12	**0.08**	0.11	1.49	0.20
V202	0.14	**0.08**	**0.08**	2.22	0.81
V203†	0.14	0.21	**0.11**	—	0.84

† These sequences are within the held-out validation set.

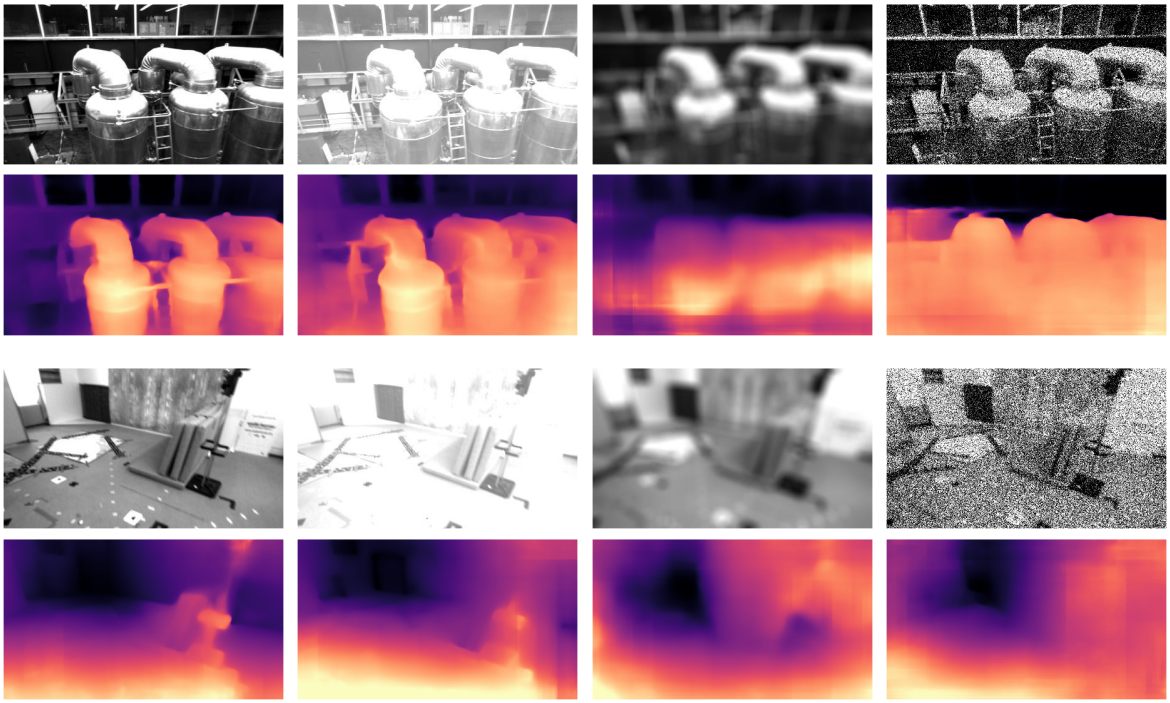


Figure 7.5: Images from the EuRoC dataset validation sequences, and the resulting image after our applied brightness, blur, and shot noise corruptions. Despite severe image degradation, the depth network reasonably estimates the scene depth.

fixed). Any error in the depth prediction currently can lead to the iterative egomotion network prediction converging to a sub-optimal value. Second, these methods optimize depth and egomotion over a window of images, while our approach is fundamentally only a two-frame estimator. Third, these feature-based approaches operate with full resolution images, from which extracted feature locations can be precisely determined. Conversely, our network-based approach is currently constrained to operate with lower resolution data. In future work, we hope to address these limitations to improve the baseline accuracy. Despite these aforementioned limitations, our hybrid system, by no longer relying on feature detection and matching, is expected to outperform the classical estimators in more challenging environments. Next, we investigate how robust our hybrid method is to a number of realistic visual degradations. For this experiment, our VIO approach and other VIO approaches are tested on EuRoC validation sequences but with degraded image streams.

Image Corruption

The EuRoC images are corrupted in three ways: by applying brightness transformations, by applying defocus blurring, and by adding shot noise. The corruptions are applied using the `ImgAug` library[9], and a severity level of five is selected. Only one type of corruption is applied at a time. Figure 7.5 shows various example corrupted EuRoC images. For our experiments, we apply each corruption to all images within a 20 second window every 40 seconds (i.e., half the images are corrupted).

[9] See https://imgaug.readthedocs.io/en/latest/

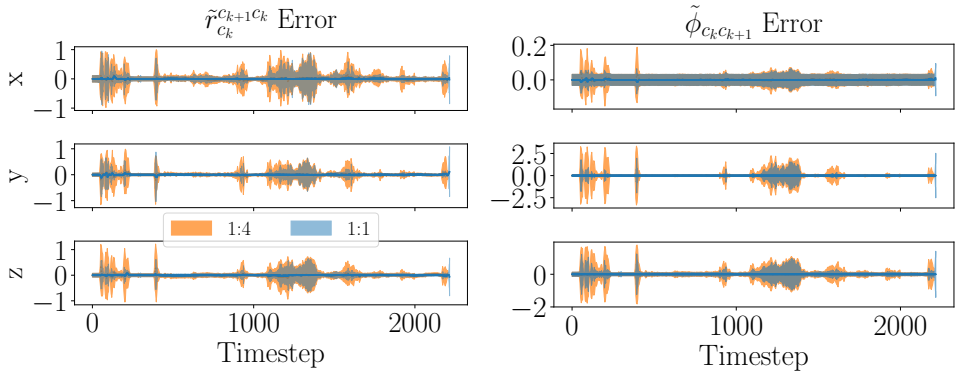


Figure 7.6: The learned relative egomotion measurement errors with their associated 1σ uncertainty bound for the full sequence ($1:1$) and quarter-frame sequence ($1:4$). The learned measurement covariance naturally inflates for the $1:4$ sequence that contains larger perspective change between frames.

Table 7.2: The mean error for three variations of our approach in the (corrupted) Table 7.3 sequences. With the vision-only approaches, the network trained with our hybrid system is more accurate than the same network trained without the EKF.

Trans. RMSE			**Rot. RMSE**		
Full	EKF Removed	Vision-Only	Ours	EKF Removed	Vision-Only
1.02	1.52	2.16	**11.37**	17.09	29.43

Frame Skipping

To simulate larger perspective changes, or a reduced camera/IMU frame rate, we downsample the image and IMU data streams across the full validation sequences. In our notation, $1:\mathrm{X}$ refers a downsample rate of X, where only one of X frames is maintained (e.g., $1:2$ removes half of the frames). We test downsample rates of $\mathrm{X} \in \{2, 3, 4\}$.

Degradation Experiment Results

We evaluate our system, along with several others, on the degraded data. We test the classical estimators VINS-Mono (Qin et al., 2018), ROVIO (Bloesch et al., 2017), and R-VIO (Huai and Huang, 2018). We also test our (tightly coupled, vision-only) learned SfM system from Chapter 6. SelfVIO (Almalioglu et al., 2022) is not publicly available for evaluation. For VINS-Mono, ROVIO, and R-VIO, we use the open-source implementations with the default EuRoC parameters, running on Ubuntu 20.04 with ROS Noetic. Table 7.3 depicts the experimental results for sequences `MH05` and `V103`. We observe that degraded conditions cause the classical estimators to fail for a significant number of the trials. During these failures, the classical estimators either lose feature tracking or diverge (resulting in 100+ metres of error). The most robust classical system is VINS-Mono, although it cannot maintain feature tracking throughout the frame skip experiment. On the other hand, our system performs consistently in most trials. Notably, the extreme $1:4$ case causes little to no increase in error for our system. Figure 7.6 plots the measurement error and the predicted covariance for the $1:1$ and $1:4$ cases. The covariance predictions shown in Figure 7.6 inflate as the camera motion becomes more extreme.

Ablation Study

Figure 7.7 shows the relative translation and rotation errors before and after the inclusion of our iterative egomotion network and our EKF components. From Figure 7.7, it is apparent that integrating the gyroscope mea-

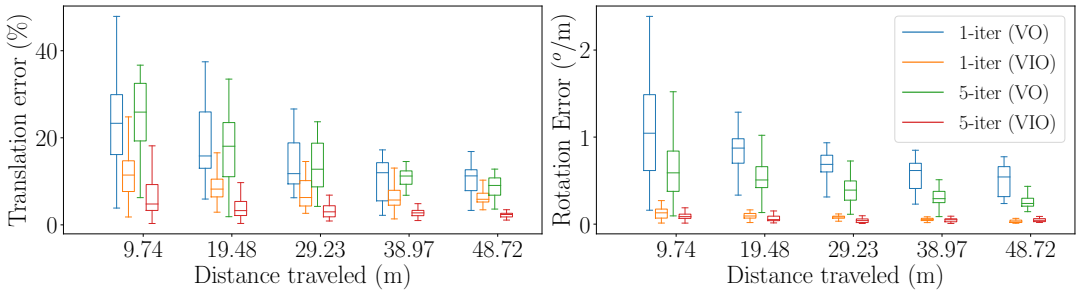


Figure 7.7: Ablation study showing the effect of adding the iterative egomotion network (i.e., 1-iter to 5-iter) and the EKF (i.e., VO to VIO) to our system. These error metrics are generated for sequence `MH05`.

surements in the EKF is crucial for accurate orientation estimation. The iterative egomotion network, when paired with the EKF, reduces the overall translation error also.

Table 7.2 lists the mean accuracy for three versions of our system on the corrupted sequences from Table 7.3. In sequence, these versions of the system are our proposed hybrid system, the standalone egomotion network (i.e., our hybrid system with the EKF removed), and the same egomotion network trained without the EKF (i.e., our tightly coupled EgoNet from Chapter 6). Notably, the presence of the EKF during training improves the accuracy of the raw egomotion network output.

Table 7.3: Results for the degraded EuRoC validation sequences. We consider a failure when 1) feature tracking is lost, or 2) the optimization diverges, resulting in rapid error growth on the order of 100+ m. Failures are indicated with an x. Despite not encountering these degradations during training, our hybrid approach maintains its accuracy.

	Trans. RMSE (Sim(3))					**Rot. RMSE** (Sim(3))				
Corruption Type	Ours	Ours (vision-only)	VINS-Mono	ROVIO	R-VIO	Ours	Ours (vision-only)	VINS-Mono	ROVIO	R-VIO
MH05 — Nominal	0.93	4.04	**0.28**	1.01	0.46	3.57	11.35	12.95	4.09	**2.36**
Brightness	1.11	4.16	**0.29**	1.64	x	6.32	14.56	15.32	**4.38**	x
Blur	1.48	3.85	**0.38**	1.20	0.48	5.56	36.06	13.18	3.47	**1.68**
Shot	2.07	4.70	**0.63**	1.54	0.99	12.66	46.77	14.08	5.55	**4.54**
(1 : 2) — Nominal	0.70	2.74	**0.27**	x	x	**4.77**	9.17	16.28	x	x
Brightness	1.09	2.72	**0.34**	x	x	**7.17**	12.69	16.52	x	x
Blur	1.71	3.92	**0.71**	x	x	**6.95**	35.96	17.41	x	x
Shot	2.43	4.15	**0.67**	x	x	**10.93**	34.60	14.20	x	x
(1 : 3) — Nominal	0.69	2.18	**0.62**	x	x	**4.26**	9.41	17.14	x	x
(1 : 4) — Nominal	0.67	1.93	**0.41**	x	x	**6.08**	10.66	15.65	x	x
V103 — Nominal	0.43	0.94	**0.15**	0.21	0.21	9.79	15.86	5.92	**2.70**	6.89
Brightness	0.53	0.92	**0.17**	0.21	x	9.09	14.17	6.25	**3.88**	x
Blur	0.82	0.81	**0.25**	0.26	x	17.35	33.19	5.71	**4.61**	x
Shot	0.59	1.33	0.50	**0.23**	x	15.12	124.24	9.50	**3.55**	x
(1 : 2) — Nominal	**0.72**	**0.72**	x	x	x	**15.07**	15.52	x	x	x
Brightness	0.86	**0.70**	x	x	x	**15.56**	16.19	x	x	x
Blur	0.91	**0.62**	x	x	x	**24.64**	30.70	x	x	x
Shot	**0.84**	1.26	x	x	x	**16.57**	64.03	x	x	x
(1 : 3) — Nominal	0.73	**0.68**	x	0.73	x	15.77	17.38	x	**8.94**	x
(1 : 4) — Nominal	1.02	**0.88**	x	x	x	**20.13**	36.07	x	x	x
# Failures	**0**	**0**	6	11	15	**0**	**0**	6	11	15
Avg.	**1.02**	2.16	—	—	—	**11.37**	29.43	—	—	—

7.4 Summary and Future Work

In this chapter, we demonstrated how our tightly coupled, learned SfM system from Chapter 6 can be incorporated as a measurement model within an EKF-based visual-inertial odometry system. Our resulting hybrid VIO system was able to effectively maintain an accurate egomotion estimate even when operating under significantly degraded conditions that caused other state-of-the-art feature-based VIO estimators to fail. We attribute this increased robustness to the replacement of the feature-based front-end. We removed this brittle component and instead used our data-driven model, while leaving the rest of the classical estimation framework (namely, the EKF structure with an IMU-based process model) unchanged. Notably, this facilitated a principled sensor fusion scheme for combining visual and inertial data and further allowed for metric scale to be recovered with our system. Owing to the differentiable nature of the filter, our hybrid system was trained end-to-end using the self-supervised photometric reconstruction loss. Overall, we believe that the work presented in this chapter adds to the growing amount of evidence that hybrid models are well suited for self-localization applications within complex operating environments.

As future work, there are several improvements that can be made to this system. First, the depth predictions produced from DepthNet could be refined to boost the accuracy of the iterative egomotion network and to improve the image reconstructions used within the reconstruction loss. This can be accomplished either by using the test-time PFT strategy from the previous chapter, or by incorporating the map into the state estimate. Second, it would be interesting to investigate how spatiotemporal self-calibration can be incorporated into our system. Finally, we note that our approach operates frame-to-frame, and so is fundamentally limited compared to alternative feature-based VIO methods that optimize over a larger window of data. For this reason, our accuracy under nominal operating conditions is limited compared to these alternative existing approaches. We would like to investigate how our system can incorporate multi-frame constraints across larger windows of data.

Chapter 8

Conclusion

In this thesis, we investigated the role of data-driven learning within inertial, visual, and visual-inertial egomotion estimation pipelines. Our motivation was to promote the long-term capabilities of modern (and future) mobile autonomy agents. As wide-scale deployment progresses, these systems will be required to operate with a higher degree of independence in increasingly more challenging environments. To fulfill these requirements, self-localization algorithms for mobile agents must be developed that go beyond what existing 'classical' algorithms are capable of. To this end, we explored how contemporary (deep) learning techniques can be applied to augment or replace these classical algorithms in order to improve overall robustness when subjected to challenging operating conditions.

We divided our work into three sections, each of which discussed a unique form of self-localization using inertial and/or visual sensors. In the first section, we examined how data-driven modelling can be used to achieve robust zero-velocity detection within a zero-velocity-aided inertial navigation system. We evaluated our system within the domain of pedestrian navigation, demonstrating how a foot-mounted IMU can be used to accurately estimate the trajectory of a wearer in real-time—making our system well-suited for applications where humans and robots are required to operate (and navigate) within a shared space. In the second section, we shifted our focus to data-driven visual egomotion estimation using a learned SfM system. In particular, we introduced (and provided motivation for using) the learned SfM based on neural network models for depth and egomotion estimation. We then substantially improved the overall accuracy of the learned SfM system and demonstrated techniques for improving generalization at test time. In the third section, we extended our learned SfM system by incorporating it as the measurement model for a vision-aided INS. We showed that, by eschewing the conventional feature-based front-end, this hybrid system is more robust than existing classical visual-inertial estimators.

The optimal way to apply learning-based methods to the domain of self-localization remains an open area of research. Herein, we have considered two ways that learning can be applied: through *hybrid* systems that incorporate learning into classical estimation frameworks or through *end-to-end* systems that rely solely on learned models for localization. We have found that hybrid approaches can leverage the benefits of both paradigms: the preservation of classical estimation components maintains the accuracy and interpretability that they provide, while the addition of data-driven components improves the robustness of the system. Specifically, we focussed on hybrid systems that incorporate robust, data-driven measurement models into the traditional aided navigation framework and found that these systems are particularly effective for inertial and visual-inertial navigation.

8.1 Summary of Contributions

In Chapter 3, we demonstrated how our novel techniques for data-driven zero-velocity detection can significantly improve the accuracy of the zero-velocity-aided navigation pipeline for pedestrian localization. In short, the contributions from this chapter were as follows:

1. the development of two robust, data-driven zero-velocity detectors that accurately identify zero-velocity events (i.e., the midstance phase) from a stream of foot-mounted inertial sensor measurements;

2. the incorporation of these detectors within a zero-velocity-aided INS (based on an EKF) to provide robust, zero-velocity pseudo-measurements;

3. the evaluation of our system on several newly collected datasets, demonstrating that our data-driven approaches consistently outperform a number of common 'classical' zero-velocity detectors on walking, running, stair-climbing, and mixed-motion trajectories.

In Chapter 4, we introduced the learned SfM framework that employs neural network models for depth and egomotion estimation. We identified that this approach, although still nascent, has the potential to replace its classical counterparts in robust self-localization systems. Notably, being fully self-supervised, this system can benefit from life-long learning by continually improving as new data are collected. At the end of the chapter, we presented our proof-of-concept system that uses the learned SfM framework to train a network that produces self-supervised pose corrections to a classical VO estimator. We used the promising results of this proof-of-concept system as motivation for further development of the learned SfM system. The two subsequent chapters discussed our contributions in this domain.

In Chapter 5, we identified how scale ambiguity—one of the common problems associated with monocular cameras—negatively impacts the learned SfM system. We accordingly proposed a novel scale recovery loss that only requires knowledge of the camera height (making it suitable for wheeled vehicle applications) in order to constrain the networks to produce metrically scaled predictions. In short, the contributions from this chapter included:

1. a novel scale recovery formulation based on self-supervised losses that enforce the measured camera height (over a ground plane) to be similar to the a priori known camera height;

2. a self-supervised framework for training a ground plane segmentation network, developed as part of our scale recovery formulation, that outputs the likelihood of each pixel belonging to the ground plane;

3. the improvement of the interframe scale consistency of the network predictions through the incorporation of our losses.

In Chapter 6, we demonstrated how coupling of the (traditionally independent) depth and egomotion network structures in the learned SfM formulation can have a significant impact on performance. We presented a *tightly coupled* network structure that resulted in state-of-the-art accuracy (relative to other learned SfM systems) and improved generalization. In short, the contributions from this chapter were:

1. the introduction of the notion of *network coupling* for the depth and egomotion networks within the learned SfM system;

2. a *tightly coupled* SfM system, the depth and egomotion networks of which are carefully linked in order to properly share information at training and test time;

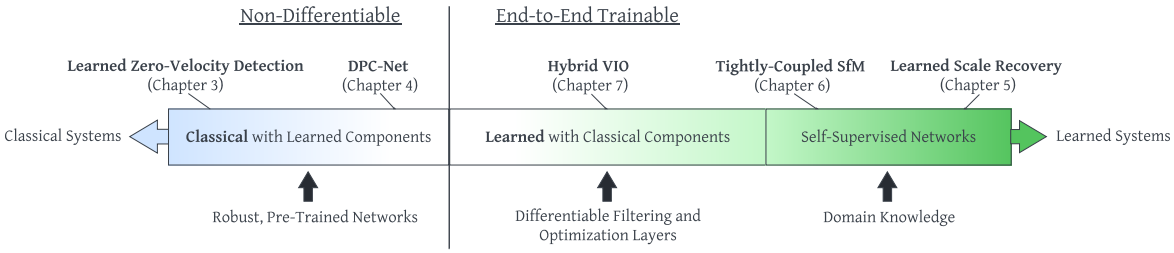


Figure 8.1: Illustrating the spectrum of hybrid approaches that combine classical estimation and data-driven (learned) components. We divide this spectrum into three major categories and show which category each one of our proposed systems falls within.

3. The discovery that proper network coupling allows for a mutually consistent scale factor to be shared between the depth and egomotion networks, facilitating inter-network scale consistency.

Finally, having substantially improved the learned SfM system, we decided to incorporate it within an INS to produce a robust visual-inertial egomotion estimator. In Chapter 7, we detailed this novel VIO system. In short, the contributions from this chapter were:

1. the incorporation of our learned SfM system as a relative pose measurement model within a hybrid, EKF-based VIO system;

2. a novel self-supervised training procedure that can be used to train a *scale-aware* and *uncertainty-aware* egomotion network capable of producing metrically scaled predictions with associated measurement covariance estimates;

3. experimental results demonstrating how our proposed hybrid approach is more robust to failure than a number of state-of-the-art, classical VIO approaches when degraded operating conditions are present.

8.2 Future Research Directions

Throughout this thesis, we investigated how data-driven models can be incorporated with classical estimation frameworks. Within our proposed systems, there was a varying degree of learning involved, and the strategies used to combine classical components (and domain knowledge) with data-driven models varied between each chapter. In Figure 8.1, we attempt to place our systems within three broad categories of hybrid systems. The first category (in blue) augments classical systems with data-driven (learned) components. In this category, the full system is nondifferentiable, which prevents it from being trained end-to-end. Instead, data-driven components must be pretrained prior to being incorporated. Our zero-velocity-aided INS from Chapter 3 and our self-supervised DPC-Net from Chapter 4 fall within this first category.

The other two categories (in green) contain the end-to-end trainable (i.e., fully differentiable) systems we proposed in the latter chapters of this thesis. By merging classical and learned components together in a way that is differentiable, learned components embedded at various points within the hybrid system can be trained by minimizing downstream losses that are directly representative of the target function of the system (e.g., for a self-localization system, minimizing a pose supervision loss directly). Conversely, learned components in nondifferentiable hybrid systems commonly need to be trained with 'intermediate' labels that may be more

arduous to acquire (e.g., in Chapter 3, we had to collect zero-velocity labels for our foot-mounted INS, rather than directly using the ground truth poses from our VICON system as labels).

We believe that future work should focus on developing hybrid systems that incorporate classical estimation components while being end-to-end trainable. These systems leverage the full benefits of modern deep learning techniques while being able to preserve the benefits that come with classical estimation frameworks (e.g., producing Bayesian uncertainty estimates, having increased interpretability, and providing flexibility to incorporate domain knowledge where needed). Through end-to-end training, the learned components can be optimized to work directly with the broader classical estimation framework of which they are a part. Further, end-to-end training enables the inclusion of a diverse range of learning-based components that would be significantly more challenging train outside of the hybrid system. Data-driven heteroscedastic uncertainty models (such as the one we developed in Chapter 7), for example, are one particular type of model can be learned through end-to-end training. Without end-to-end training, uncertainty labels—which are difficult to acquire—would be required to train such a model.

There are many promising avenues of research that involve developing hybrid, end-to-end trainable systems. In addition to work being done with differentiable filtering, alternative systems incorporate differentiable optimization layers within neural networks for SfM and SLAM (Tang and Tan, 2019; Teed and Deng, 2021). Further, Jatavallabhula et al. (2020) propose a fully differentiable SLAM system that enables data-driven models to be incorporated at various stages within the front-end or back-end.

Lastly, we emphasize the importance of self-supervised learning within data-driven perception systems. With the availability of self-supervised labels, loss functions can be evaluated at training *and* test time, enabling a new mode of operation that is based on *prediction error minimization*. Prediction error minimization involves refining the original network prediction (e.g., with subsequent forward passes or by applying corrections produced from an auxiliary network) in such a way that the loss is further minimized. This novel mode of operation was explored in Chapter 6 with our iterative feedback coupling strategy for egomotion estimation. We demonstrated that by applying successive forward passes through the egomotion network, the predictions progressively improved to further reduce the photometric reconstruction loss. This resulted in greater accuracy and better generalization of the egomotion predictions. In many ways, this framework for predictive error minimization is similar to the human perception system; the brain relies on a similar (albeit more complex) form of error minimization that attempts to match top-down (generative) predictions about the world with the incoming sensory inputs (Clark, 2013). Drawing inspiration from the effectiveness of human perception and the promising results from our iterative feedback coupling experiments, we believe that predictive error minimization should be further explored. One avenue for future work could involve incorporating self-supervised losses from multiple sensor modalities to provide a stronger supervisory signal for error minimization. Another research direction might investigate how other types of generative models can be applied to reconstruct sensor data from latent variables. View synthesis through depth and egomotion estimation is one example of generative model that we used, but other models such as neural radiance fields (Mildenhall et al., 2020) are gaining popularity and should be examined.

Appendices

Appendix A

Visual Datasets and Evaluation Metrics

The data-driven methods for visual (and visual-inertial) egomotion estimation presented in this thesis rely on several existing (open source) datasets for training and evaluation. For our purposes, use of these datasets enables the comparison of our methodology with related literature through standard evaluation benchmarks. In this section, we briefly introduce the datasets—and their associated, specific evaluation metrics—that are relevant to the thesis. For visual egomotion estimation, we utilize four datasets: the KITTI and Oxford RobotCar datasets for autonomous driving, and the ScanNet and EuRoC datasets for indoor navigation. The foot-mounted inertial navigation datasets, which we collected ourselves, are described in Chapter 3.

A.1 The KITTI Dataset

The KITTI odometry dataset (Geiger et al., 2012) is a driving dataset collected in Karlsruhe, Germany. For data collection, the vehicle was equipped with two forward-facing Point Grey Flea2 colour global shutter cameras (operating at 20 Hz). All intrinsic and extrinsic transform parameters are provided with the dataset. There are 22 available sequences, which amounts to 41,000 images captured over a driving distance of 39.2 km. Ground truth poses are available from a high-quality INS for the first ten sequences (00–10). Vehicle traverses are divided into 'city', 'residential', and 'road' categories.

For monocular VO, the standard training/test split uses sequences 00–08 for training and 09–10 for validation. We report the average translational and rotational errors, normalized by segment length ($t_{err}(\%)$, $r_{err}(^o/100\text{ m})$), over all possible sub-sequences of length $(100, 200, \ldots, 800)$ metres. To evaluate the performance of unscaled monocular systems, a constant scale factor is applied to all of the relative translation estimates within a sequence to align with the ground truth translation. For a sequence with relative translation estimates, $\mathbf{r}_k$, and ground truth translation, $\mathbf{r}'_k$, the per-sequence scale factor is

$$s = \sum_i \frac{\|\mathbf{r}'_i\|}{\|\mathbf{r}_i\|}. \tag{A.1}$$

For depth evaluation, we follow the standard 'Eigen' train/test split (Eigen and Fergus, 2015) and evaluate the standard error metrics. For an estimated pixel depth, d_i, and a ground truth depth, d'_i, the error metrics are

(a) The KITTI data collection vehicle (from http://www.cvlibs.net/datasets/kitti/).

(b) Sample images from KITTI.

Figure A.1: The KITTI dataset.

(a) The Oxford RobotCar data collection vehicle (from https://robotcar-dataset.robots.ox.ac.uk/).

(b) Sample images from Oxford RobotCar (with the lower pixels cropped to remove the hood of the data collection vehicle).

Figure A.2: The Oxford RobotCar dataset.

Abs Rel:	$\frac{1}{N}\sum_{i \in N}\frac{\lvert d_i' - d_i \rvert}{d_i'}$	Sq Rel:	$\frac{1}{N}\sum_{i \in N}\frac{(d_i' - d_i)^2}{d_i'}$
RMSE:	$\sqrt{\frac{1}{N}\sum_{i \in N}(d_i' - d_i)^2}$	RMSE Log:	$\sqrt{\frac{1}{N}\sum_{i \in N}(\log d_i' - \log d_i)^2}$

Errors are averaged across all N pixels in the test dataset. A second metric is $\delta < \gamma$, which is the percentage of test set pixels where $\delta = \max(\frac{d_i'}{d_i}, \frac{d_i}{d_i'})$ is within the error threshold, γ. This metric is evaluated for $\delta < 1.25$, $\delta < 1.25^2$, and $\delta < 1.25^3$. Prior to evaluation of the depth metric, a per-image scale factor is applied to align the unscaled depth predictions with ground truth.

A.2 The Oxford RobotCar Dataset

The Oxford RobotCar dataset (Maddern et al., 2017) is an autonomous driving dataset spanning more than 1,000 km and collected over a period of one year in central Oxford, United Kingdom. For data collection, the vehicle was equipped with a forward-facing Point Grey Bumblebee XB3 global shutter trinocular stereo camera operat-

(a) Room-level reconstructions from ScanNet (from http://www.scan-net.org/).

(b) Sample images from ScanNet.

Figure A.3: The ScanNet dataset.

ing at 16 Hz. Images were stored as lossless, compressed PNG files in unrectified 8-bit raw Bayer format. Ground truth position information was provided by a GPS-INS system. We use the publicly-available development kit[1] to perform Bayer demosaicing and undistortion of the images. Undistortion is performed via a look-up table that is provided by Point Grey (as part of a factory calibration).

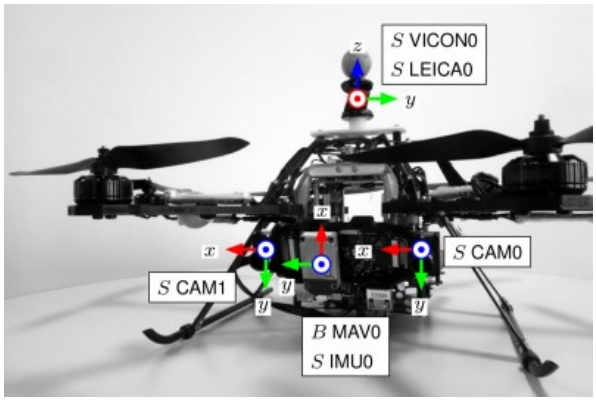
In our experiments, we use a subset of the dataset for training and validation. Sequences `2014-11-18-13-20-12`, `2015-07-08-13-37-17`, `2015-07-10-10-01-59`, and `2015-08-12-15-04-18` are used for training, but the first and second subsequences of `2014-11-18-13-20-12` are held out for validation. For evaluation, we follow the KITTI odometry benchmark and report the translation and rotation mean segment errors for all possible subsequences of length $(100, 200, \ldots, 800)$ metres.

A.3 The ScanNet Dataset

ScanNet (Dai et al., 2017a) is a large RGB-D video dataset with 2.5 million camera views from 1,513 indoor sequences. The images were captured with an RGB-D *Structure Sensor*, which incorporates a global shutter RGB camera operating at 30 Hz. Ground truth poses are available for odometry evaluation, and the camera intrinsics are provided. Ground truth was generated using a dense 3D reconstruction method called BundleFusion (Dai et al., 2017b). We follow the training/test split from Tang and Tan (2019), where the first 1,413 sequences are used for training and 2,000 image pairs from the remaining 100 sequences are selected for testing. For evaluation on ScanNet, we report the depth and camera pose error metrics as described in Tang and Tan (2019) and, similar to existing literature, use image-wise rescaling to align our predictions with ground truth (for both depth and poses). The depth metrics are the same *Abs Rel*, *Sq Rel*, and *RMSE* metrics used for KITTI, in addition to the *scale invariant (Sc-Inv)* error (Eigen et al., 2014),

$$e_{sc-inv} = \sqrt{\frac{1}{N}\left(\sum_i z_i^2\right) - \frac{1}{N^2}\left(\sum_i z_i\right)^2}, \quad z_i = \log d_i - \log d_i'. \tag{A.2}$$

[1] See https://github.com/ori-mrg/robotcar-dataset-sdk.



(a) The data collection drone for EuRoC (from https://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets).

(b) Sample images from EuRoC in the machine hall (top) and Vicon room (bottom).

Figure A.4: The EuRoC dataset.

The three ScanNet pose metrics represent the error between the predicted translation and rotation, $\{\mathbf{r}_i,\ \mathbf{C}_i\}$, and the ground truth $\{\mathbf{r}'_i,\ \mathbf{C}'_i\}$. The mean angular error (in degrees) is

$$e_{rot} = \frac{1}{N}\sum_{i\in N} \cos^{-1}\frac{tr(\mathbf{C}_i^T\mathbf{C}'_i)-1}{2}. \tag{A.3}$$

The angular error (in degrees) between the predicted translation vectors $\mathbf{r}_i$ and the ground truth $\mathbf{r}'_i$ is

$$e_{trans-deg} = \frac{1}{N}\sum_{i\in N} \cos^{-1}\frac{\mathbf{r}_i\cdot\mathbf{r}'_i}{||\mathbf{r}_i||\,||\mathbf{r}'_i||}, \tag{A.4}$$

and the distance (in centimetres) between the endpoints of the (scale-aligned) predicted and ground truth translation vectors is

$$e_{trans-cm} = \frac{100}{N}\sum_{i\in N}\sqrt{\sum_{n\in\{x,y,z\}}\left(\mathbf{r}'_{i,n}-s\mathbf{r}_{i,n}\right)^2},\quad s=\frac{\mathbf{r}'_i\cdot\mathbf{r}_i}{\mathbf{r}_i\cdot\mathbf{r}_i}. \tag{A.5}$$

A.4 The EuRoC Dataset

The EuRoC dataset (Burri et al., 2016), contains visual and inertial measurements collected from on board an AscTec Firefly micro aerial vehicle (MAV). For data collection, the MAV was equipped with a (grayscale) global shutter stereo camera, operating at 20 Hz, and a Skybotix IMU sensor, operating at 200 Hz. The visual-inertial data were synchronized on board also. Intrinsic and extrinsic calibration parameters are provided. In total, 11 sequences were collected within a machine hall and in two rooms that contained Vicon motion capture systems. In the machine hall, 3D position ground truth was obtained using a Leica MS50 MultiStation laser tracker. In the Vicon-equipped rooms, 6-DOF pose ground truth was obtained from the Vicon data (i.e., from markers on the drone). The vehicle was manually operated throughout these sequences and underwent rapid movements that produced motion blur in the image stream, making this dataset particularly challenging for odometry estimation.

Appendix B

Network Structures

In this appendix, we provide specific details about the network structures used for our learned SfM systems described in Chapters 5 to 7. These (coupled) networks consist of a depth network (DepthNet), an egomotion network (EgoNet), and a plane segmentation network (SegNet, see Chapter 5).

Egomotion network (EgoNet). Table B.1 describes the layers of our egomotion network. The network input consists of a stacked source-target image pair with an optional 2D optical flow input that we use for the scale recovery system in Chapter 5.[1] The flow input is omitted from the feedback-coupled egomotion network used in the later chapters. Convolutional layers are applied to the input, reducing the dimensionality while increasing the number of channels. Following each convolutional layer, weight standardization (WS) (Qiao et al., 2019), group normalization (GN) (Wu and He, 2018), and ReLU activations are applied after all but the last convolution. Omission of the ReLU activation allows for both positive and negative values to be present in the egomotion prediction. The final convolution layer reduces the number of channels to six; the average pooling layer then reduces the spatial resolution to 1×1.

[1]The optical flow vectors, generated using the Gunnar Farneback algorithm Farnebäck (2003), are incorporated in part because Zhou et al. (2019a) demonstrate that using intermediate representations can improve performance on vision-based tasks.

Table B.1: The network layers for EgoNet. Note that strided convolutions are used instead of pooling layers for dimensionality reduction to preserve the spatial information present within the input.

EgoNet Layers					
Layer	**Kernel Size**	**Stride**	**In/Out Channels**	**Input**	**Activation**
conv1	3	1	6/16	stacked-img-pair[1]	WS, GN, ReLU
conv2	3	2	16/32	conv1	WS, GN, ReLU
conv3	3	3	32/64	conv2	WS, GN, ReLU
conv4	3	2	64/128	conv3	WS, GN, ReLU
conv5	3	2	128/256	conv4	WS, GN, ReLU
conv6	3	2	256/256	conv5	WS, GN, ReLU
conv7	3	2	256/256	conv6	WS, GN, ReLU
poseconv	1	1	256/6	conv7	—
avgpool	—[2]	—	6/6	poseconv	—

[1] There are 8 input channels if a 2D optical flow estimate between the stacked images is included for estimation.
[2] The kernel size is selected to be the same size as the remaining spatial resolution, which is a function of the original image resolution. The average pooling layer reduces the $6 \times M \times N$ feature map to a $6 \times 1 \times 1$ pose output.

Table B.2: The network details for the decoder of DepthNet. Each layer merges the previous layer with the *skip connection* from the ResNet encoder layers (enc5, enc4, enc3, enc2). To increase the spatial resolution at each layer, upsampling layers (indicated with ↑), via nearest neighbour interpolation, are applied prior to the upconv convolutions. The final DCP layers incorporate information from the last three decoder levels to perform disparity prediction. Channel concatenation is indicated with ⊕.

DepthNet Decoder Layers					
Layer	**Kernel Size**	**Stride**	**In/Out Channels**	**Input**	**Activation**
upconv5	3	1	512/256	↑enc5	ELU
iconv5	3	1	256/256	upconv5	ELU
upconv4	3	1	256/128	↑iconv5	ELU
iconv4	3	1	128/128	upconv4 + enc4	ELU
upconv3	3	1	128/64	↑iconv4	ELU
iconv3	3	1	64/64	upconv3 + enc3	ELU
upconv2	3	1	64/64	↑iconv3	ELU
iconv2	3	1	64/64	upconv2 + enc2	ELU
upconv1	3	1	64/32	↑iconv2	ELU
iconv1	3	1	32/32	upconv1	ELU
Depth Prediction with DCP Layers					
dcp3	3	1	64/8	iconv3	ELU
dcp2	3	1	64/8	iconv2	ELU
dcp1	3	1	32/8	iconv1	ELU
disp	3	1	(8+8+8)/1	dcp3 ⊕ ↑dcp2 ⊕ ↑dcp1	sigmoid

Depth Network (DepthNet). The depth network is a U-Net (Ronneberger et al., 2015) encoder-decoder The encoder is a standard ResNet18 network (He et al., 2016) pretrained on ImageNet. We refer the reader to He et al. (2016) for additional details on ResNet18. Table B.2 describes the layers of our depth network decoder, which is based on Monodepth2 (Godard et al., 2019). Starting at the bottleneck (i.e., the middle region of the U-Net, where the feature map has the smallest spatial resolution), the decoder gradually increases the spatial resolution back to the input size. The decoder consists of a series of upsampling and 2D convolution layers. The upsampling scheme we use is nearest neighbour interpolation. At various points in the decoder, the feature maps are merged, via *skip connections*, with the feature maps from the intermediate layers of the Resnet18 encoder (i.e., *enc4*, *enc3*, and *enc2* in Table B.2). These feature maps are merged by addition. To produce a depth prediction at the end of the network, we adopt the 'dense connected prediction' (DCP) convolutional layers from DNet (Xue et al., 2020). Here, the feature-map outputs from the final three *iconv* layers are passed through the *dcp* convolution layers, combined (via concatenation), and passed into a final *disp* convolution layer. The output of the *disp* convolution is the disparity prediction, which is passed into Equation (4.2) to produce the inverse depth prediction. Finally, the inverse depth prediction is inverted to yield the depth prediction (whose per-pixel values are within $[d_{min}, d_{max}]$). For our experiments, we set the d_{min}, d_{max} hyperparameters for each given dataset.

Plane Segmentation Network (SegNet) SegNet is designed to be similar to DepthNet, since the overall functionality of the two networks is roughly the same (both output per-pixel predictions). The encoder consists of a pretrained Resnet18 network. The decoder that follows is described in Table B.3.

Table B.3: The network details for the decoder of SegNet. Like DepthNet, skip connections merge information from the encoder at various stages of the decoder.

SegNet Decoder Layers					
Layer	**Kernel Size**	**Stride**	**In/Out Channels**	**Input**	**Activation**
upconv5	3	1	512/256	↑enc5	ELU
iconv5	3	1	256/256	upconv5	ELU
upconv4	3	1	256/128	↑iconv5	ELU
iconv4	3	1	128/128	upconv4 + enc4	ELU
upconv3	3	1	128/64	↑iconv4	ELU
iconv3	3	1	64/64	upconv3 + enc3	ELU
upconv2	3	1	64/64	↑iconv3	ELU
iconv2	3	1	64/64	upconv2 + enc2	ELU
upconv1	3	1	64/32	↑iconv2	ELU
iconv1	3	1	32/32	upconv1	ELU
plane1	3	1	32/1	iconv1	sigmoid

Appendix C

Error State Kalman Filter Details

This appendix provides additional details pertaining to the zero-velocity-aided error state EKF from Chapter 3. Starting from the true state dynamics, we show how the state can be separated into nominal and error state components. The error state dynamics are linear with respect to the state and can integrated to produce the discrete-time error state process model. This process model is used to propagate the state covariance forward in time as IMU measurements are received. We describe these steps within this appendix and refer the reader to Solà (2015) for additional details.

To begin, we reintroduce our state, $\mathbf{x}_\tau$, which consists of the position, $\mathbf{r}_i^{v_\tau i}$, velocity, $\mathbf{v}_i^{v_\tau i}$, and orientation, $\mathbf{q}_{iv_\tau}$, of the IMU at the discrete timestep τ (corresponding to time t_τ). In the ESKF formulation, the true state is separated into the *nominal* (error-free) state, $\bar{\mathbf{x}}_\tau$, and the *error state*, $\delta\mathbf{x}_\tau$:

$$\mathbf{x}_\tau = \begin{bmatrix} \mathbf{r}_i^{v_\tau i} \\ \mathbf{v}_i^{v_\tau i} \\ \mathbf{q}_{iv_\tau} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{r}}_i^{v_\tau i} + \delta\mathbf{r}_i^{v_\tau i} \\ \bar{\mathbf{v}}_i^{v_\tau i} + \delta\mathbf{v}_i^{v_\tau i} \\ \bar{\mathbf{q}}_{iv_\tau} \otimes \exp(\frac{\delta\boldsymbol{\phi}_{iv_\tau}}{2}) \end{bmatrix}. \tag{C.1}$$

As discussed in Section 2.2.3, the error state for the rotation is defined in $\mathbb{R}^3$. Applying the quaternion exponential map yields the quaternion error state on S^3. This error state is then composed with the nominal (orientation) state. In the ESKF formulation, the nominal state and error state are both propagated as IMU measurements (consisting of the specific force, $\mathbf{a}_{m,\tau}$ and angular velocity $\boldsymbol{\omega}_{m,\tau}$) are received. The nominal state quantities ($\bar{\mathbf{r}}_i^{v_\tau i}$, $\bar{\mathbf{v}}_i^{v_\tau i}$, and $\bar{\mathbf{q}}_{iv_\tau}$) are large-signal and are integrated using a non-linear process model, while the error state quantities ($\delta\mathbf{r}_i^{v_\tau i}$, $\delta\mathbf{v}_i^{v_\tau i}$, and $\delta\boldsymbol{\phi}_{iv_\tau}$) are small-signal, linear, and are estimated within the ESKF along with their covariance. Note that since the error state accounts for all of the sources of noise within the system, the error state covariance represents the full uncertainty of the true state. Next, we derive the continuous-time dynamics for the nominal and error states. Starting with the true state dynamics,

$$\begin{aligned} \dot{\mathbf{r}}_i^{v_\tau i} &= \mathbf{v}_i^{v_\tau i}, \\ \dot{\mathbf{v}}_i^{v_\tau i} &= \mathbf{a}_i^{v_\tau i}, \\ \dot{\mathbf{q}}_{iv_\tau} &= \frac{1}{2}\mathbf{q}_{iv_\tau} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{v_\tau}^{v_\tau i} \end{bmatrix}. \end{aligned} \tag{C.2}$$

We can substitute the IMU measurement model from Section 2.3.1 (while assuming there are no biases) to yield

$$\begin{aligned}\dot{\mathbf{r}}_i^{v_\tau i} &= \mathbf{v}_i^{v_\tau i},\\ \dot{\mathbf{v}}_i^{v_\tau i} &= \mathbf{C}\{\mathbf{q}_{iv_\tau}\}(\mathbf{a}_{m,\tau} - \mathbf{w}_{a,\tau}) - \mathbf{g}_i,\\ \dot{\mathbf{q}}_{iv_\tau} &= \frac{1}{2}\mathbf{q}_{iv_\tau} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{m,\tau} - \mathbf{w}_{\omega,\tau} \end{bmatrix}.\end{aligned} \tag{C.3}$$

The continuous-time dynamics of the nominal state and the error state are found by substituting Equation (C.1) into Equation (C.3) and separating the nominal values from the error values. First, we provide the resulting equations, and leave the derivation until the end of this appendix. The continuous-time nominal state dynamics are

$$\begin{aligned}\dot{\bar{\mathbf{r}}}_i^{v_\tau i} &= \bar{\mathbf{v}}_i^{v_\tau i},\\ \dot{\bar{\mathbf{v}}}_i^{v_\tau i} &= \mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}\mathbf{a}_{m,\tau} - \mathbf{g}_i,\\ \dot{\bar{\mathbf{q}}}_{iv_\tau} &= \frac{1}{2}\bar{\mathbf{q}}_{iv_\tau} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{m,\tau} \end{bmatrix}.\end{aligned} \tag{C.4}$$

The linearized, continuous-time error state dynamics are

$$\begin{aligned}\delta\dot{\mathbf{p}}_i^{v_\tau i} &= \delta\mathbf{v}_i^{v_\tau i},\\ \delta\dot{\mathbf{v}}_i^{v_\tau i} &= -\mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}(\mathbf{a}_{m,\tau})^\wedge\delta\boldsymbol{\phi}_{iv_\tau} - \mathbf{w}_{a,\tau},\\ \delta\dot{\boldsymbol{\phi}}_{iv_\tau} &= -(\boldsymbol{\omega}_{m,\tau})^\wedge\delta\boldsymbol{\phi}_{iv_\tau} - \mathbf{w}_{\omega,\tau}.\end{aligned} \tag{C.5}$$

These continuous-time equations are integrated to produce the discrete-time kinematics (process) models for the nominal and error states. The error state process model is also used for covariance propagation, which we discuss in the next section.

Derivation of the Error State Process Model and Covariance Propagation. The Equation (C.5) error state dynamics are linear with respect to the state and the noise perturbation vectors, and therefore can be expressed as

$$\delta\dot{\mathbf{x}}_\tau = \mathbf{F}_\tau\delta\mathbf{x}_\tau + \mathbf{G}_\tau\mathbf{w}_\tau, \tag{C.6}$$

with

$$\mathbf{F}_\tau = \begin{bmatrix} \mathbf{0} & \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -\mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}(\mathbf{a}_{m,\tau})^\wedge \\ \mathbf{0} & \mathbf{0} & -(\boldsymbol{\omega}_{m,\tau})^\wedge \end{bmatrix}, \quad \mathbf{G}_\tau = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ -\mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & -\mathbf{I}_3 \end{bmatrix}, \quad \mathbf{w}_\tau = \begin{bmatrix} \mathbf{w}_{a,\tau} \\ \mathbf{w}_{\omega,\tau} \end{bmatrix}. \tag{C.7}$$

Note that $\mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}$ is evaluated with the most recent state posterior, $\hat{\mathbf{q}}_{iv_\tau}$. The solution to Equation (C.6), from Farrell (2008), is

$$\delta\mathbf{x}_\tau = \boldsymbol{\Phi}_{\tau+1,\tau}\delta\mathbf{x}_\tau + \int_{t_\tau}^{t_{\tau+1}} \boldsymbol{\Phi}_{\tau+1,s}\mathbf{G}_s\mathbf{w}_s ds, \tag{C.8}$$

$$= \boldsymbol{\Phi}_{\tau+1,\tau}\delta\mathbf{x}_\tau + \mathbf{w}'_\tau. \tag{C.9}$$

where $\mathbf{\Phi}_{\tau+1,\tau}$, the error state transition matrix, propagates the discrete-time error state from t_τ to $t_{\tau+1}$, and is determined through

$$\mathbf{\Phi}_{\tau+1,\tau} = \exp\left(\int_{\tau}^{\tau+1} \mathbf{F}_s ds\right) \tag{C.10}$$

$$\approx \mathbf{I}_9 + \mathbf{F}_\tau \delta t. \tag{C.11}$$

Note that we make the assumption that $\mathbf{F}_\tau$ is constant across the δt interval to compute $\mathbf{\Phi}_{\tau+1,\tau}$. The covariance propagation through the error state model is straightforward for the first term in Equation (C.8), but less so for the second term, which requires the computation of $\mathbf{Q}'_\tau = \mathbb{E}\left[\mathbf{w}'_\tau \mathbf{w}'^\top_\tau\right]$:

$$\check{\mathbf{P}}_{\tau+1} = \mathbf{\Phi}_{\tau+1,\tau} \hat{\mathbf{P}}_\tau \mathbf{\Phi}^\top_{\tau+1,\tau} + \mathbf{Q}'_\tau. \tag{C.12}$$

A common solution (Farrell, 2008), which assumes $\mathbf{w}_\tau$ is a white noise process (i.e., having constant power at all frequencies, see Farrell (2008) for additional information), is

$$\mathbf{Q}'_\tau \approx \mathbf{G}_\tau \mathbf{Q}_\tau \mathbf{G}^\top_\tau \delta t, \tag{C.13}$$

where $\mathbf{Q}_\tau$ is the IMU input covariance, $\mathbf{Q}_\tau = \mathbb{E}\left[\mathbf{w}_\tau \mathbf{w}^\top_\tau\right] = \begin{bmatrix} \sigma_a^2 \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \sigma_\omega^2 \mathbf{I}_3 \end{bmatrix}$.

Derivation of the Nominal and Error State Dynamics Models. Here, we derive the Equation (C.4) and Equation (C.5) dynamics models starting from Equation (C.3). By substituting the nominal and error state composition for $\mathbf{x}_\tau$ using $\mathbf{x}_\tau = \bar{\mathbf{x}}_\tau \oplus \delta\mathbf{x}_\tau$, the state dynamics become

$$\dot{\bar{\mathbf{r}}}_i^{v_\tau i} + \delta\dot{\mathbf{r}}_i^{v_\tau i} = \bar{\mathbf{v}}_i^{v_\tau i} + \delta\mathbf{v}_i^{v_\tau i}, \tag{C.14}$$

$$\dot{\bar{\mathbf{v}}}_i^{v_\tau i} + \delta\dot{\mathbf{v}}_i^{v_\tau i} \approx \mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}(\mathbf{I}_3 + \delta\boldsymbol{\phi}^\wedge_{iv_\tau})(\mathbf{a}_{m,\tau} - \mathbf{w}_{a,\tau}) - \mathbf{g}_i, \tag{C.15}$$

$$\dot{\bar{\mathbf{q}}}_{iv_\tau} \otimes \delta\mathbf{q}_{iv_\tau} + \bar{\mathbf{q}}_{iv_\tau} \otimes \delta\dot{\mathbf{q}}_{iv_\tau} = \frac{1}{2}(\bar{\mathbf{q}}_{iv_\tau} \otimes \delta\mathbf{q}_{iv_\tau}) \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{m,\tau} - \mathbf{w}_{\omega,\tau} \end{bmatrix}. \tag{C.16}$$

The nominal and error state terms are separated in these three equations to more easily calculate their respective dynamics. For Equation (C.14), separating the nominal terms from the error terms is simple: the nominal state terms are $\dot{\bar{\mathbf{r}}}_i^{v_\tau i} = \bar{\mathbf{v}}_i^{v_\tau i}$ and the error state terms are $\delta\dot{\mathbf{r}}_i^{v_\tau i} = \delta\mathbf{v}_i^{v_\tau i}$. To separate the terms in Equation (C.15), we expand the right hand side,

$$\dot{\bar{\mathbf{v}}}_i^{v_\tau i} + \delta\dot{\mathbf{v}}_i^{v_\tau i} \approx \mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}(\mathbf{I}_3 + \delta\boldsymbol{\phi}^\wedge_{iv_\tau})(\mathbf{a}_{m,\tau} - \mathbf{w}_{a,\tau}) - \mathbf{g}_i, \tag{C.17}$$

$$\approx \mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}\mathbf{a}_{m,\tau} - \mathbf{g}_i + \mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}\delta\boldsymbol{\phi}^\wedge_{iv_\tau}\mathbf{a}_{m,\tau} - \mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}\mathbf{w}_{a,\tau} - \mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}\delta\boldsymbol{\phi}^\wedge_{iv_\tau}\mathbf{w}_{a,\tau}. \tag{C.18}$$

The first two terms in this final expression are the nominal values, which can be isolated to produce $\dot{\bar{\mathbf{v}}}_i^{v_\tau i} = \mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}\mathbf{a}_{m,\tau} - \mathbf{g}_i$. The three final terms are functions of the error state: using the relation $\mathbf{a}^\wedge\mathbf{b} = -\mathbf{b}^\wedge\mathbf{a}$ to modify the first error term and removing the third (higher-order) error term, we arrive at the final error state

dynamics that are linear with respect to the error state:

$$\delta\dot{\mathbf{v}}_i^{v_\tau i} = -\mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}\mathbf{a}_{m,\tau}^{\wedge}\delta\boldsymbol{\phi}_{iv_\tau} - \mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}\mathbf{w}_{a,\tau}, \tag{C.19}$$

$$= -\mathbf{C}\{\bar{\mathbf{q}}_{iv_\tau}\}\mathbf{a}_{m,\tau}^{\wedge}\delta\boldsymbol{\phi}_{iv_\tau} - \mathbf{w}_{a,\tau}. \tag{C.20}$$

In the second expression, we omit the rotation operation on the white noise term, since the mean and covariance of the noise is unaffected by a rotation when it is assumed to be isotropic and uncorrelated. Finally, the orientation error state dynamics can be determined by manipulating Equation (C.16). Substituting Poisson's kinematical equation, $\dot{\bar{\mathbf{q}}}_{iv_\tau} = \frac{1}{2}\bar{\mathbf{q}}_{iv_\tau} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{m,\tau} \end{bmatrix}$, in the left hand side gives us

$$\frac{1}{2}\bar{\mathbf{q}}_{iv_\tau} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{m,\tau} \end{bmatrix} \otimes \delta\mathbf{q}_{iv_\tau} + \bar{\mathbf{q}}_{iv_\tau} \otimes \dot{\delta\mathbf{q}}_{iv_\tau} = \frac{1}{2}\bar{\mathbf{q}}_{iv_\tau} \otimes \delta\mathbf{q}_{iv_\tau} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{m,\tau} - \mathbf{w}_{\omega,\tau} \end{bmatrix}, \tag{C.21}$$

$$\frac{1}{2}\bar{\mathbf{q}}_{iv_\tau} \otimes \left(\begin{bmatrix} 0 \\ \boldsymbol{\omega}_{m,\tau} \end{bmatrix} \otimes \delta\mathbf{q}_{iv_\tau} + 2\dot{\delta\mathbf{q}}_{iv_\tau} \right) = \frac{1}{2}\bar{\mathbf{q}}_{iv_\tau} \otimes \left(\delta\mathbf{q}_{iv_\tau} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{m,\tau} - \mathbf{w}_{\omega,\tau} \end{bmatrix} \right). \tag{C.22}$$

We can isolate the terms within the brackets on each side and solve for $\delta\dot{\mathbf{q}}_{iv_\tau}$,

$$2\delta\dot{\mathbf{q}}_{iv_\tau} = \delta\mathbf{q}_{iv_\tau} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{m,\tau} - \mathbf{w}_{\omega,\tau} \end{bmatrix} - \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{m,\tau} \end{bmatrix} \otimes \delta\mathbf{q}_{iv_\tau}. \tag{C.23}$$

Using the Equation (2.18) matrix-form multiplication for quaternions to manipulate the right side, while approximating $2\delta\dot{\mathbf{q}}_{iv_\tau} \approx \begin{bmatrix} 0 \\ \delta\dot{\boldsymbol{\phi}}_{iv_\tau} \end{bmatrix}$ and $\delta\mathbf{q}_{iv_\tau} \approx \begin{bmatrix} 1 \\ \frac{\delta\boldsymbol{\phi}_{iv_\tau}}{2} \end{bmatrix}$, we get the following relation:

$$\begin{bmatrix} 0 \\ \delta\dot{\boldsymbol{\phi}}_{iv_\tau} \end{bmatrix} = \left(\left[\begin{bmatrix} 0 \\ \boldsymbol{\omega}_{m,\tau} - \mathbf{n}_{\omega,\tau} \end{bmatrix} \right]_R - \left[\begin{bmatrix} 0 \\ \boldsymbol{\omega}_{m,\tau} \end{bmatrix} \right]_L \right) \delta\mathbf{q}_{iv_\tau}, \tag{C.24}$$

$$= \begin{bmatrix} 0 & -\mathbf{w}_{\omega,\tau}^{\top} \\ -\mathbf{w}_{\omega,\tau} & (-2\boldsymbol{\omega}_{m,\tau} + \mathbf{w}_{\omega,\tau})^{\wedge} \end{bmatrix} \begin{bmatrix} 1 \\ \frac{\delta\boldsymbol{\phi}_{iv_\tau}}{2} \end{bmatrix}. \tag{C.25}$$

Finally, solving the bottom row of Equation (C.24) (and ignoring the second-order term $\frac{\mathbf{w}_{\omega,\tau}^{\wedge}\delta\boldsymbol{\phi}_{iv_\tau}}{2}$) gives us

$$\delta\dot{\boldsymbol{\phi}}_{iv_\tau} = -(\boldsymbol{\omega}_{m,\tau})^{\wedge}\delta\boldsymbol{\phi}_{iv_\tau} - \mathbf{w}_{\omega,\tau}. \tag{C.26}$$

Appendix D

Derivation of the Relative Pose Measurement Jacobian

In this appendix, we derive the measurement Jacobians for the relative pose measurement model of our hybrid VIO system from Chapter 7. We include derivations with and without the inclusion of the scale parameter in the state, since we use both versions within our system. As previously shown, the measurement residual $\boldsymbol{\epsilon}_{k+1}$ is

$$\boldsymbol{\epsilon}_{k+1} = \begin{bmatrix} \boldsymbol{\epsilon}_{\phi,k+1} \\ \boldsymbol{\epsilon}_{r,k+1} \end{bmatrix} = \begin{bmatrix} \log\left(\tilde{\mathbf{C}}_{c_k c_{k+1}} \mathbf{C}_{c_k c_{k+1}}^\top\right)^\vee \\ \tilde{\boldsymbol{r}}_{c_k}^{c_{k+1}c_k} - \mathbf{r}_{c_k}^{c_{k+1}c_k} \end{bmatrix}, \tag{D.1}$$

where $\{\mathbf{C}_{c_k c_{k+1}}, \mathbf{r}_{c_k}^{c_{k+1}c_k}\}$ is the (relative) IMU pose that has been transformed to the current camera frame in order to be 'compatible' with the camera-centric egomotion measurements. This transformation, which uses the known extrinsic calibration between the camera and IMU, $\{\mathbf{C}_{v_k c_k}, \mathbf{r}_{v_k}^{c_k v_k}\}$, is

$$\mathbf{r}_{c_k}^{c_{k+1}c_k} = \mathbf{C}_{v_k c_k}^\top \mathbf{C}_{r_k v_{k+1}} \mathbf{r}_{v_k}^{c_k v_k} + \mathbf{C}_{v_k c_k}^\top \left(\mathbf{r}_{r_k}^{v_{k+1} r_k} - \mathbf{r}_{v_k}^{c_k v_k}\right), \tag{D.2}$$

$$\mathbf{C}_{c_k c_{k+1}} = \mathbf{C}_{v_k c_k}^\top \mathbf{C}_{r_k v_{k+1}} \mathbf{C}_{v_k c_k}. \tag{D.3}$$

Note that these equations originate from $\mathbf{T}_{c_k c_{k+1}} = \mathbf{T}_{c_k r_k} \mathbf{T}_{r_k v_{k+1}} \mathbf{T}_{c_{k+1} v_{k+1}}^{-1}$. The measurement Jacobian $\mathbf{H}_{k+1} = \frac{\partial \boldsymbol{\epsilon}_{k+1}}{\partial \delta \mathbf{x}_{k+1,r_k}}\Big|_{\check{\mathbf{x}}_{k+1,r_k}}$ is found by differentiating Equation (D.1) with respect to the error state:

$$\mathbf{H}_{k+1} = \begin{bmatrix} \mathbf{0}_{3\times 9} & -\mathbf{C}_{v_k c_k}^\top \mathbf{J}_\ell(-\check{\boldsymbol{\phi}}_{r_k v_{k+1}})^{-1} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 9} \\ \mathbf{0}_{3\times 9} & \mathbf{C}_{v_k c_k}^\top \check{\mathbf{C}}_{r_k v_{k+1}} \mathbf{r}_{v_k}^{c_k v_k \wedge} & -\mathbf{C}_{v_k c_k}^\top & \mathbf{0}_{3\times 9} \end{bmatrix}. \tag{D.4}$$

We derive the three non-zero terms of this measurement Jacobian below. First, the term in the upper row (the Jacobian of the rotation measurement) is derived. Then, the two terms in the second row (the Jacobian of the translation measurement) are derived.

Derivation of the Rotation Measurement Jacobian. To compute the Jacobian of $\boldsymbol{\epsilon}_{\phi,k+1}$ with respect to the error state, we use the first-order Baker-Campbell-Hausdorff (BCH) (Barfoot, 2017) formula to express the

rotation error as a subtraction,

$$\boldsymbol{\epsilon}_{\phi,k+1} = \log\left(\tilde{\mathbf{C}}_{c_k c_{k+1}} \mathbf{C}_{c_k c_{k+1}}^{\top}\right)^{\vee}, \tag{D.5}$$

$$= \log\left(\exp\left(\tilde{\boldsymbol{\phi}}_{c_k c_{k+1}}^{\wedge}\right) \exp\left(\boldsymbol{\phi}_{c_k c_{k+1}}{}^{\wedge}\right)^{\top}\right)^{\vee}, \tag{D.6}$$

$$\approx \tilde{\boldsymbol{\phi}}_{c_k c_{k+1}} - \boldsymbol{\phi}_{c_k c_{k+1}}. \tag{D.7}$$

This approximation assumes that the rotation vectors are small, which is usually valid in high-rate odometry applications such as ours. Now, through a series of manipulations, we can make $\boldsymbol{\epsilon}_{\phi,k+1}$ linear with respect to the error state. Substituting $\boldsymbol{\phi}_{c_k c_{k+1}} = \log\left(\mathbf{C}_{c_k c_{k+1}}\right)^{\vee}$ and using Equation (D.3) this expression becomes

$$\boldsymbol{\epsilon}_{\phi,k+1} = \tilde{\boldsymbol{\phi}}_{c_k c_{k+1}} - \log\left(\mathbf{C}_{v_k c_k}^{\top} \mathbf{C}_{r_k v_{k+1}} \mathbf{C}_{v_k c_k}\right)^{\vee}. \tag{D.8}$$

Letting $\mathbf{C}_{r_k v_{k+1}} = \exp\left(\boldsymbol{\phi}_{r_k v_{k+1}}{}^{\wedge}\right)$, we obtain

$$\boldsymbol{\epsilon}_{\phi,k+1} = \tilde{\boldsymbol{\phi}}_{c_k c_{k+1}} - \log\left(\mathbf{C}_{v_k c_k}^{\top} \exp\left(\boldsymbol{\phi}_{r_k v_{k+1}}{}^{\wedge}\right) \mathbf{C}_{v_k c_k}\right)^{\vee}, \tag{D.9}$$

$$= \tilde{\boldsymbol{\phi}}_{c_k c_{k+1}} - \log\left(\exp\left(\mathbf{C}_{v_k c_k}^{\top} \boldsymbol{\phi}_{r_k v_{k+1}}{}^{\wedge}\right)\right)^{\vee}, \tag{D.10}$$

$$= \tilde{\boldsymbol{\phi}}_{c_k c_{k+1}} - \mathbf{C}_{v_k c_k}^{\top} \boldsymbol{\phi}_{r_k v_{k+1}}, \tag{D.11}$$

$$= \tilde{\boldsymbol{\phi}}_{c_k c_{k+1}} - \mathbf{C}_{v_k c_k}^{\top} \log\left(\mathbf{C}_{r_k v_{k+1}}\right)^{\vee}, \tag{D.12}$$

where the second line makes use of the identity $\mathbf{C} \exp\left(\mathbf{u}^{\wedge}\right) \mathbf{C}^{\top} = \exp\left((\mathbf{C}\mathbf{u})^{\wedge}\right)$ (Barfoot, 2017). Separating $\mathbf{C}_{r_k v_{k+1}}$ into its nominal and error state components, $\mathbf{C}_{r_k v_{k+1}} = \bar{\mathbf{C}}_{r_k v_{k+1}} \exp\left(\delta\boldsymbol{\phi}_{r_k v_{k+1}}{}^{\wedge}\right)$, this becomes

$$\boldsymbol{\epsilon}_{\phi,k+1} = \tilde{\boldsymbol{\phi}}_{c_k c_{k+1}} - \mathbf{C}_{v_k c_k}^{\top} \log\left(\bar{\mathbf{C}}_{r_k v_{k+1}} \exp\left(\delta\boldsymbol{\phi}_{r_k v_{k+1}}{}^{\wedge}\right)\right)^{\vee}, \tag{D.13}$$

$$= \tilde{\boldsymbol{\phi}}_{c_k c_{k+1}} - \mathbf{C}_{v_k c_k}^{\top} \log\left(\exp\left(\bar{\boldsymbol{\phi}}_{r_k v_{k+1}}^{\wedge}\right) \exp\left(\delta\boldsymbol{\phi}_{r_k v_{k+1}}{}^{\wedge}\right)\right)^{\vee}. \tag{D.14}$$

Using the BCH approximation for a small rotation (Barfoot, 2017),

$$\log\left(\exp\left(\boldsymbol{\phi}_1{}^{\wedge}\right) \exp\left(\boldsymbol{\phi}_2{}^{\wedge}\right)\right)^{\vee} \approx \begin{cases} \mathbf{J}_{\ell}(\boldsymbol{\phi}_1)^{-1}\boldsymbol{\phi}_1 + \boldsymbol{\phi}_2 & \text{if } \boldsymbol{\phi}_1 \text{ small}, \\ \boldsymbol{\phi}_1 + \mathbf{J}_r(\boldsymbol{\phi}_1)^{-1}\boldsymbol{\phi}_2 & \text{if } \boldsymbol{\phi}_2 \text{ small}, \end{cases} \tag{D.15}$$

the second term becomes linear with respect to the rotation error:

$$\boldsymbol{\epsilon}_{\phi,k+1} \approx \tilde{\boldsymbol{\phi}}_{c_k c_{k+1}} - \mathbf{C}_{v_k c_k}^{\top} \left(\bar{\boldsymbol{\phi}}_{r_k v_{k+1}} + \mathbf{J}_r(\bar{\boldsymbol{\phi}}_{r_k v_{k+1}})^{-1} \delta\boldsymbol{\phi}_{r_k v_{k+1}}\right), \tag{D.16}$$

$$\approx \tilde{\boldsymbol{\phi}}_{c_k c_{k+1}} - \mathbf{C}_{v_k c_k}^{\top} \bar{\boldsymbol{\phi}}_{r_k v_{k+1}} - \mathbf{C}_{v_k c_k}^{\top} \mathbf{J}_r(\bar{\boldsymbol{\phi}}_{r_k v_{k+1}})^{-1} \delta\boldsymbol{\phi}_{r_k v_{k+1}}, \tag{D.17}$$

$$\approx \underbrace{\tilde{\boldsymbol{\phi}}_{c_k c_{k+1}} - \mathbf{C}_{v_k c_k}^{\top} \bar{\boldsymbol{\phi}}_{r_k v_{k+1}}}_{\bar{\boldsymbol{\epsilon}}_{\phi,k+1}} - \mathbf{C}_{v_k c_k}^{\top} \mathbf{J}_{\ell}(-\bar{\boldsymbol{\phi}}_{r_k v_{k+1}})^{-1} \delta\boldsymbol{\phi}_{r_k v_{k+1}}. \tag{D.18}$$

Note that we use $\mathbf{J}_{\ell}(-\boldsymbol{\phi}) = \mathbf{J}_r(\boldsymbol{\phi})$ (Barfoot, 2017) to produce the final expression. We now have an expression where the nominal terms, $\bar{\boldsymbol{\epsilon}}_{\phi,k+1}$, are separated from the error state terms, and the expression is linear with

respect to $\delta\boldsymbol{\phi}_{r_k v_{k+1}}$. The derivative of this expression is

$$\frac{\partial \boldsymbol{\epsilon}_{\phi,k+1}}{\partial \delta\boldsymbol{\phi}_{r_k v_{k+1}}} = -\mathbf{C}_{v_k c_k}^{\top} \mathbf{J}_{\ell}(-\bar{\boldsymbol{\phi}}_{r_k v_{k+1}})^{-1}. \tag{D.19}$$

Translation Measurement Jacobian. A similar process is followed to compute the Jacobian of $\boldsymbol{\epsilon}_{r,k+1}$ with respect to the error state. Substituting Equation (D.2) into Equation (D.1), $\boldsymbol{\epsilon}_{r,k+1}$ becomes

$$\begin{aligned}
\boldsymbol{\epsilon}_{r,k+1} &= \tilde{\boldsymbol{r}}_{c_k}^{c_{k+1}c_k} - \mathbf{r}_{c_k}^{c_{k+1}c_k}, && \text{(D.20)}\\
&= \tilde{\boldsymbol{r}}_{c_k}^{c_{k+1}c_k} - \left(\mathbf{C}_{v_k c_k}^{\top} \mathbf{C}_{r_k v_{k+1}} \mathbf{r}_{v_k}^{c_k v_k} + \mathbf{C}_{v_k c_k}^{\top} \left(\mathbf{r}_{r_k}^{v_{k+1} r_k} - \mathbf{r}_{v_k}^{c_k v_k}\right)\right). && \text{(D.21)}
\end{aligned}$$

Letting $\mathbf{C} = \bar{\mathbf{C}}_{r_k v_{k+1}} \exp\left(\delta\boldsymbol{\phi}_{r_k v_{k+1}}{}^{\wedge}\right)$, and $\mathbf{r}_{r_k}^{v_{k+1} r_k} = \bar{\mathbf{r}}_{r_k}^{v_{k+1} r_k} + \delta\mathbf{r}_{r_k}^{v_{k+1} r_k}$, we can isolate the nominal and error state terms:

$$\begin{aligned}
\boldsymbol{\epsilon}_{r,k+1} &= \tilde{\boldsymbol{r}}_{c_k}^{c_{k+1}c_k} - \left(\mathbf{C}_{v_k c_k}^{\top} \bar{\mathbf{C}}_{r_k v_{k+1}} \exp\left(\delta\boldsymbol{\phi}_{r_k v_{k+1}}{}^{\wedge}\right) \mathbf{r}_{v_k}^{c_k v_k} + \mathbf{C}_{v_k c_k}^{\top} \left(\bar{\mathbf{r}}_{r_k}^{v_{k+1} r_k} + \delta\mathbf{r}_{r_k}^{v_{k+1} r_k} - \mathbf{r}_{v_k}^{c_k v_k}\right)\right), && \text{(D.22)}\\
&\approx \tilde{\boldsymbol{r}}_{c_k}^{c_{k+1}c_k} - \mathbf{C}_{v_k c_k}^{\top} \bar{\mathbf{C}}_{r_k v_{k+1}} (\mathbf{I}_3 + \delta\boldsymbol{\phi}_{r_k v_{k+1}}^{\wedge}) \mathbf{r}_{v_k}^{c_k v_k} - \mathbf{C}_{v_k c_k}^{\top} \bar{\mathbf{r}}_{r_k}^{v_{k+1} r_k} - \mathbf{C}_{v_k c_k}^{\top} \delta\mathbf{r}_{r_k}^{v_{k+1} r_k} + \mathbf{C}_{v_k c_k}^{\top} \mathbf{r}_{v_k}^{c_k v_k}, && \text{(D.23)}\\
&\approx \tilde{\boldsymbol{r}}_{c_k}^{c_{k+1}c_k} - \mathbf{C}_{v_k c_k}^{\top} \bar{\mathbf{C}}_{r_k v_{k+1}} \mathbf{r}_{v_k}^{c_k v_k} - \mathbf{C}_{v_k c_k}^{\top} \bar{\mathbf{C}}_{r_k v_{k+1}} (\delta\boldsymbol{\phi}_{r_k v_{k+1}})^{\wedge} \mathbf{r}_{v_k}^{c_k v_k} \\
&\quad - \mathbf{C}_{v_k c_k}^{\top} \bar{\mathbf{r}}_{r_k}^{v_{k+1} r_k} - \mathbf{C}_{v_k c_k}^{\top} \delta\mathbf{r}_{r_k}^{v_{k+1} r_k} + \mathbf{C}_{v_k c_k}^{\top} \mathbf{r}_{v_k}^{c_k v_k}, && \text{(D.24)}\\
&\approx \bar{\boldsymbol{\epsilon}}_{r,k+1} - \mathbf{C}_{v_k c_k}^{\top} \bar{\mathbf{C}}_{r_k v_{k+1}} (\delta\boldsymbol{\phi}_{r_k v_{k+1}})^{\wedge} \mathbf{r}_{v_k}^{c_k v_k} - \mathbf{C}_{v_k c_k}^{\top} \delta\mathbf{r}_{r_k}^{v_{k+1} r_k}, && \text{(D.25)}
\end{aligned}$$

where $\bar{\boldsymbol{\epsilon}}_{r,k+1}$ contains all of the nominal state terms, independent of the error state. The other two terms, containing error states, can be differentiated with respect to the error state. The derivative with respect to the rotation error vector, $\delta\boldsymbol{\phi}_{r_k v_{k+1}}$, is

$$\frac{\partial \boldsymbol{\epsilon}_{r,k+1}}{\partial \delta\boldsymbol{\phi}_{r_k v_{k+1}}} = -\frac{\partial \left(\mathbf{C}_{v_k c_k}^{\top} \bar{\mathbf{C}}_{r_k v_{k+1}} (\delta\boldsymbol{\phi}_{r_k v_{k+1}})^{\wedge} \mathbf{r}_{v_k}^{c_k v_k}\right)}{\partial \delta\boldsymbol{\phi}_{r_k v_{k+1}}}. \tag{D.26}$$

Using $\mathbf{u}^{\wedge}\mathbf{v} = -\mathbf{v}^{\wedge}\mathbf{u}$ (Barfoot, 2017), this expression becomes linear with respect to $\delta\boldsymbol{\phi}_{r_k v_{k+1}}$:

$$\begin{aligned}
\frac{\partial \boldsymbol{\epsilon}_{r,k+1}}{\partial \delta\boldsymbol{\phi}_{r_k v_{k+1}}} &= \frac{\partial \left(\mathbf{C}_{v_k c_k}^{\top} \bar{\mathbf{C}}_{r_k v_{k+1}} (\mathbf{r}_{v_k}^{c_k v_k})^{\wedge} \delta\boldsymbol{\phi}_{r_k v_{k+1}}\right)}{\partial \delta\boldsymbol{\phi}_{r_k v_{k+1}}}, && \text{(D.27)}\\
&= \mathbf{C}_{v_k c_k}^{\top} \bar{\mathbf{C}}_{r_k v_{k+1}} (\mathbf{r}_{v_k}^{c_k v_k})^{\wedge}. && \text{(D.28)}
\end{aligned}$$

The derivative with respect to $\delta\mathbf{r}_{r_k}^{v_{k+1} r_k}$ is straightforward:

$$\begin{aligned}
\frac{\partial \boldsymbol{\epsilon}_{r,k+1}}{\partial \delta\mathbf{r}_{r_k}^{v_{k+1} r_k}} &= -\frac{\partial \left(\mathbf{C}_{v_k c_k}^{\top} \delta\mathbf{r}_{r_k}^{v_{k+1} r_k}\right)}{\partial \delta\mathbf{r}_{r_k}^{v_{k+1} r_k}}, && \text{(D.29)}\\
&= -\mathbf{C}_{v_k c_k}^{\top}. && \text{(D.30)}
\end{aligned}$$

Incorporating the Scale Parameter. With the addition of the scale parameter, λ_{k+1}, the translation measurement residual becomes

$$\boldsymbol{\epsilon}_{r,k+1} = \tilde{\boldsymbol{r}}_{c_k}^{c_{k+1}c_k} - \lambda_{k+1}\mathbf{r}_{c_k}^{c_{k+1}c_k}, \tag{D.31}$$

$$= \tilde{\boldsymbol{r}}_{c_k}^{c_{k+1}c_k} - \lambda_{k+1}\left(\mathbf{C}_{v_kc_k}^{\top}\mathbf{C}_{r_kv_{k+1}}\mathbf{r}_{v_k}^{c_kv_k} + \mathbf{C}_{v_kc_k}^{\top}\left(\mathbf{r}_{r_k}^{v_{k+1}r_k} - \mathbf{r}_{v_k}^{c_kv_k}\right)\right). \tag{D.32}$$

We rederive the measurement Jacobian for this new measurement residual. To do so, we separate the scale factor (along with our other state variables) into the nominal and error state components through $\lambda_{k+1} = \bar{\lambda}_{k+1} + \delta\lambda_{k+1}$. The new measurement Jacobian becomes

$$\mathbf{H}_{k+1} = \begin{bmatrix} \mathbf{0}_{3\times9} & -\mathbf{C}_{v_kc_k}^{\top}\mathbf{J}_{\ell}(-\check{\boldsymbol{\phi}}_{r_kv_{k+1}})^{-1} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times9} & \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times9} & \check{\lambda}_{k+1}\mathbf{C}_{v_kc_k}^{\top}\check{\mathbf{C}}_{r_kv_{k+1}}\mathbf{r}_{v_k}^{c_kv_k\wedge} & -\check{\lambda}_{k+1}\mathbf{C}_{v_kc_k}^{\top} & \mathbf{0}_{3\times9} & -\mathbf{C}_{v_kc_k}^{\top}(\check{\boldsymbol{\phi}}_{r_kv_{k+1}}^{\wedge}\mathbf{r}_{v_k}^{c_kv_k} + \check{\mathbf{r}}_{r_k}^{v_{k+1}r_k}) \end{bmatrix}. \tag{D.33}$$

The upper row remains the same as the original measurement Jacobian in Equation (D.4), since the rotation residual is unchanged. The terms in the second row are all changed; we derive these terms next. Starting from Equation (D.32), we can separate the true state into the nominal and the error state:

$$\boldsymbol{\epsilon}_{r,k+1} = \tilde{\boldsymbol{r}}_{c_k}^{c_{k+1}c_k} - \lambda_{k+1}\left(\mathbf{C}_{v_kc_k}^{\top}\mathbf{C}_{r_kv_{k+1}}\mathbf{r}_{v_k}^{c_kv_k} + \mathbf{C}_{v_kc_k}^{\top}\left(\mathbf{r}_{r_k}^{v_{k+1}r_k} - \mathbf{r}_{v_k}^{c_kv_k}\right)\right), \tag{D.34}$$

$$\begin{aligned} = \tilde{\boldsymbol{r}}_{c_k}^{c_{k+1}c_k} - (\bar{\lambda}_{k+1} + \delta\lambda_{k+1})\Big(&\mathbf{C}_{v_kc_k}^{\top}\bar{\mathbf{C}}_{r_kv_{k+1}}(\mathbf{I}_3 + \delta\boldsymbol{\phi}_{r_kv_{k+1}}^{\wedge})\mathbf{r}_{v_k}^{c_kv_k} \\ &+\mathbf{C}_{v_kc_k}^{\top}\left(\bar{\mathbf{r}}_{r_k}^{v_{k+1}r_k} + \delta\mathbf{r}_{r_k}^{v_{k+1}r_k} - \mathbf{r}_{v_k}^{c_kv_k}\right)\Big). \end{aligned} \tag{D.35}$$

Expanding and rearranging these terms, while removing second-order quantities and above, we arrive at the following expression:

$$\begin{aligned} \boldsymbol{\epsilon}_{r,k+1} = \bar{\boldsymbol{\epsilon}}_r &- \bar{\lambda}_{k+1}\mathbf{C}_{v_kc_k}^{\top}\bar{\mathbf{C}}_{r_kv_{k+1}}\delta\boldsymbol{\phi}_{r_kv_{k+1}}^{\wedge}\mathbf{r}_{v_k}^{c_kv_k} - \delta\lambda_{k+1}\mathbf{C}_{v_kc_k}^{\top}\bar{\mathbf{C}}_{r_kv_{k+1}}\mathbf{r}_{v_k}^{c_kv_k} \\ &- \delta\lambda_{k+1}\mathbf{C}_{v_kc_k}^{\top}\bar{\mathbf{r}}_{r_k}^{v_{k+1}r_k} - \bar{\lambda}_{k+1}\mathbf{C}_{v_kc_k}^{\top}\delta\mathbf{r}_{r_k}^{v_{k+1}r_k} + \delta\lambda_{k+1}\mathbf{C}_{v_kc_k}^{\top}\mathbf{r}_{v_k}^{c_kv_k}. \end{aligned} \tag{D.36}$$

Note that $\bar{\boldsymbol{\epsilon}}_{r,k+1}$ contains all of the nominal terms and is independent of the error state. Differentiating with respect to $\delta\boldsymbol{\phi}_{r_kv_{k+1}}$ produces

$$\frac{\partial\boldsymbol{\epsilon}_{r,k+1}}{\partial\delta\boldsymbol{\phi}} = -\frac{\partial(\bar{\lambda}_{k+1}\mathbf{C}_{v_kc_k}^{\top}\bar{\mathbf{C}}_{r_kv_{k+1}}\delta\boldsymbol{\phi}_{r_kv_{k+1}}^{\wedge}\mathbf{r}_{v_k}^{c_kv_k})}{\partial\delta\boldsymbol{\phi}_{r_kv_{k+1}}}, \tag{D.37}$$

$$= \bar{\lambda}_{k+1}\mathbf{C}_{v_kc_k}^{\top}\bar{\mathbf{C}}_{r_kv_{k+1}}(\mathbf{r}_{v_k}^{c_kv_k})^{\wedge}. \tag{D.38}$$

Differentiating with respect to $\delta\mathbf{r}_{r_k}^{v_{k+1}r_k}$ gives us

$$\frac{\partial\boldsymbol{\epsilon}_{r,k+1}}{\partial\delta\mathbf{r}_{r_k}^{v_{k+1}r_k}} = -\frac{\partial\left(\bar{\lambda}_{k+1}\mathbf{C}_{v_kc_k}^{\top}\delta\mathbf{r}_{r_kv_{k+1}}\right)}{\partial\delta\mathbf{r}_{r_kv_{k+1}}}, \tag{D.39}$$

$$= -\bar{\lambda}_{k+1}\mathbf{C}_{v_kc_k}^{\top}. \tag{D.40}$$

Finally, the derivative with respect to $\delta\lambda_{k+1}$ is:

$$\frac{\partial \boldsymbol{\epsilon}_r}{\partial \delta\lambda_{k+1}} = -\frac{\partial \left(\delta\lambda_{k+1}\mathbf{C}_{v_k c_k}^{\top}\bar{\mathbf{C}}_{r_k v_{k+1}}\mathbf{r}_{v_k}^{c_k v_k} - \delta\lambda_{k+1}\mathbf{C}_{v_k c_k}^{\top}\bar{\mathbf{r}}_{r_k}^{v_{k+1} r_k} + \delta\lambda_{k+1}\mathbf{C}_{v_k c_k}^{\top}\mathbf{r}_{v_k}^{c_k v_k}\right)}{\partial \delta\lambda_{k+1}}, \tag{D.41}$$

$$= \mathbf{C}_{v_k c_k}^{\top}\bar{\mathbf{C}}_{r_k v_{k+1}}\mathbf{r}_{v_k}^{c_k v_k} - \mathbf{C}_{v_k c_k}^{\top}\bar{\mathbf{r}}_{r_k}^{v_{k+1} r_k} + \mathbf{C}_{v_k c_k}^{\top}\mathbf{r}_{v_k}^{c_k v_k}, \tag{D.42}$$

$$= \mathbf{C}_{v_k c_k}^{\top}\left((\bar{\mathbf{C}}_{r_k v_{k+1}} - \mathbf{I})\mathbf{r}_{v_k}^{c_k v_k} - \bar{\mathbf{r}}_{r_k}^{v_{k+1} r_k}\right), \tag{D.43}$$

$$\approx -\mathbf{C}_{v_k c_k}^{\top}\left(\bar{\boldsymbol{\phi}}_{r_k v_{k+1}}^{\wedge}\mathbf{r}_{v_k}^{c_k v_k} + \bar{\mathbf{r}}_{r_k}^{v_{k+1} r_k}\right). \tag{D.44}$$

Evaluating all of these Jacobian terms using the predicted state values produces Equation (D.33).

Bibliography

Almalioglu, Y., Turan, M., Saputra, M., de Gusmão, P., Markham, A., and Trigoni, N. (2022). SelfVIO: Self-supervised deep monocular visual-inertial odometry and depth estimation. *Neural Networks*, 150:119–136.

Ambrus, R., Guizilini, V., Li, J., Pillai, S., and Gaidon, A. (2020). Two stream networks for self-supervised ego-motion estimation. In *Conf. Robot Learn. (CoRL)*, pages 1052–1061.

Aqel, M., Marhaban, M., Saripan, M., and Ismail, N. (2016). Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, 5(1):1897.

Ashkar, R., Romanovas, M., Goridko, V., Schwaab, M., Traechtler, M., and Manoli, Y. (2013). A low-cost shoe-mounted inertial navigation system with magnetic disturbance compensation. In *Proc. IEEE Int. Conf. Indoor Position. Indoor Navig. (IPIN)*, pages 1–10.

Barfoot, T. (2017). *State Estimation for Robotics*. Cambridge University Press, New York, NY, USA, 1st edition.

Bian, J., Li, Z., Wang, N., Zhan, H., Shen, C., Cheng, M., and Reid, I. (2019). Unsupervised scale-consistent depth and ego-motion learning from monocular video. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, pages 35–45.

Bloesch, M., Burri, M., Omari, S., Hutter, M., and Siegwart, R. (2017). Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback. *Int. J. Robot. Res. (IJRR)*, 36(10):1053–1072.

Brown, D. C. (1966). Decentering distortion of lenses. *Photogrammetric Eng. Remote Sens.*

Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M., and Siegwart, R. (2016). The euroc micro aerial vehicle datasets. *Int. J. Robot. Res. (IJRR)*, 35(10):1157–1163.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Trans. Robot.*, 32(6):1309–1332.

Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L., and Lopez, A. (2020). A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, 408:189–215.

Chan, C. W. and Rudins, A. (1994). Foot biomechanics during walking and running. In *Mayo Clin. Proc.*, volume 69, pages 448–461.

Chen, C., Lu, C., Wang, B., Trigoni, N., and Markham, A. (2021a). DynaNet: Neural Kalman dynamical model for motion estimation and prediction. *IEEE Trans. Neural Networks Learning Syst.*, 32(12):5479–5491.

Chen, C., Lu, X., Markham, A., and Trigoni, N. (2018). IONet: Learning to cure the curse of drift in inertial odometry. In *Proc. AAAI Conf. Artificial Intell.*, volume 32.

Chen, C., Rosa, S., Miao, Y., Lu, C., Wu, W., Markham, A., and Trigoni, N. (2019a). Selective sensor fusion for neural visual-inertial odometry. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 10542–10551.

Chen, Y., Schmid, C., and Sminchisescu, C. (2019b). Self-supervised learning with geometric constraints in monocular video: Connecting flow, depth, and camera. In *Proc. IEEE/CVF Conf. Comput. Vision (ICCV)*, pages 7063–7072.

Chen, Z., Du, H., Liao, Y., Wang, Y., and Xiong, R. (2021b). Fully differentiable and interpretable model for VIO with 4 trainable parameters. *arXiv preprint arXiv:2109.12292*.

Clark, A. (2013). Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behav. Brain Sci.*, 36(3):181–204.

Clark, R., Bloesch, M., Czarnowski, J., Leutenegger, S., and Davison, A. (2018). Learning to solve nonlinear least squares for monocular stereo. In *Proc. Eur. Conf. Comput. Vision (ECCV)*, pages 284–299.

Clark, R., Wang, S., Wen, H., Markham, A., and Trigoni, N. (2017). VINet: Visual-inertial odometry as a sequence-to-sequence learning problem. In *Proc. AAAI Conf. on Artificial Intell.*, volume 31.

Clement, L. and Kelly, J. (2018). How to train a CAT: Learning canonical appearance transformations for direct visual localization under illumination change. *IEEE Robot. Autom. Lett. (RAL)*, 3(3):2447–2454.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (ELUs).

Cortés, S., Solin, A., and Kannala, J. (2018). Deep learning based speed estimation for constraining strapdown inertial navigation on smartphones. In *IEEE 28th Int. Workshop Mach. Learn. Signal. Process. (MLSP)*, pages 1–6.

Costante, G. and Ciarfuglia, T. (2018). LS-VO: Learning dense optical subspace for robust visual odometry estimation. *IEEE Robot. Autom. Lett. (RAL)*, 3(3):1735–1742.

Costante, G., Mancini, M., Valigi, P., and Ciarfuglia, T. A. (2015). Exploring representation learning with CNNs for frame-to-frame ego-motion estimation. *IEEE Robot. Autom. Lett. (RAL)*, 1(1):18–25.

Dai, A., Chang, A., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. (2017a). ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 5828–5839.

Dai, A., Nießner, M., Zollhöfer, M., Izadi, S., and Theobalt, C. (2017b). BundleFusion: Real-time globally consistent 3D reconstruction using on-the-fly surface reintegration. *ACM Trans. Graph. (ToG)*, 36(4):1.

Delmerico, J. and Scaramuzza, D. (2018). A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2502–2509.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 248–255.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: Part I. *IEEE Robot. Autom. Mag.*, 13(2):99–110.

Eigen, D. and Fergus, R. (2015). Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proc. IEEE/CVF Int. Conf. Comput. Vision (ICCV)*, pages 2650–2658.

Eigen, D., Puhrsch, C., and Fergus, R. (2014). Depth map prediction from a single image using a multi-scale deep network. *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 27.

Engel, J., Koltun, V., and Cremers, D. (2017). Direct sparse odometry. *IEEE Trans. Pattern Anal. Mach. Intell. (PAMI)*, 40(3):611–625.

Engel, J., Schöps, T., and Cremers, D. (2014). LSD-SLAM: Large-scale direct monocular SLAM. In *Proc. Eur. Conf. Comput. Vision (ECCV)*, pages 834–849.

Engel, J., Sturm, J., and Cremers, D. (2013). Semi-dense visual odometry for a monocular camera. In *Proc. IEEE/CVF Int. Conf. Comput. Vision (ICCV)*, pages 1449–1456.

Fanani, N., Stürck, A., Barnada, M., and Mester, R. (2017). Multimodal scale estimation for monocular visual odometry. In *Proc. IEEE Intell. Veh. Symp. (IV)*, pages 1714–1721.

Farnebäck, G. (2003). Two-frame motion estimation based on polynomial expansion. In *Proc. Scandinavian Conf. Image Analysis*, pages 363–370.

Farrell, J. (2008). *Aided navigation: GPS with high rate sensors*. McGraw-Hill, Inc.

Fischer, C. and Gellersen, H. (2010). Location and navigation support for emergency responders: A survey. *IEEE Pervasive Comput.*, 9(1):38–47.

Fischler, M. and Bolles, R. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *J. Commun. ACM*, 24(6):381–395.

Forster, C., Carlone, L., Dellaert, F., and Scaramuzza, D. (2015). IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *Proc. Robot. Sci. Syst. (RSS)*.

Forster, C., Pizzoli, M., and Scaramuzza, D. (2014). SVO: Fast semi-direct monocular visual odometry. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 15–22.

Foxlin, E. (2005). Pedestrian tracking with shoe-mounted inertial sensors. *IEEE Comput. Graph. Appl.*, 25(6):38–46.

Frost, D., Kähler, O., and Murray, D. (2016). Object-aware bundle adjustment for correcting monocular scale drift. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 4770–4776.

Frost, D., Prisacariu, V., and Murray, D. (2018). Recovering stable scale in monocular slam using object-supplemented bundle adjustment. *IEEE Trans. on Robots.*, 34(3):736–747.

Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets robotics: The KITTI dataset. *Int. J. Robot. Res. (IJRR)*, 32(11):1231–1237.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 3354–3361.

Geiger, A., Ziegler, J., and Stiller, C. (2011). StereoScan: Dense 3D reconstruction in real-time. In *Proc. IEEE Intell. Veh. Symp. (IV)*, pages 963–968.

Godard, C., Aodha, O., and Brostow, G. (2017). Unsupervised monocular depth estimation with left-right consistency. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 270–279.

Godard, C., Aodha, O., Firman, M., and Brostow, G. (2019). Digging into self-supervised monocular depth estimation. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 3828–3838.

Godha, S. and Lachapelle, G. (2008). Foot mounted inertial system for pedestrian navigation. *Meas. Sci. Technol.*, 19(7).

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Gu, X., Yuan, W., Dai, Z., Zhu, S., Tang, C., and Tan, P. (2021). DRO: Deep recurrent optimizer for structure-from-motion. *arXiv preprint arXiv:2103.13201*.

Gui, J., Gu, D., Wang, S., and Hu, H. (2015). A review of visual inertial odometry from filtering and optimisation perspectives. *Adv. Robot.*, 29(20):1289–1301.

Guizilini, V., R., Pillai, S., Raventos, A., and Gaidon, A. (2020a). 3D packing for self-supervised monocular depth estimation. In *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 2485–2494.

Guizilini, V., Vitor, L., Jie, A., Ambrus, R., Pillai, S., and Gaidon, A. (2020b). Robust semi-supervised monocular depth estimation with reprojected distances. In *Conf. Robot Learn. (CoRL)*, pages 503–512.

Gurturk, M., Yusefi, A., Aslan, M., Soycan, M., Durdu, A., and Masiero, A. (2021). The YTU dataset and recurrent neural network based visual-inertial odometry. *Measurement*, 184:109878.

Haarnoja, T., Ajay, A., Levine, S., and Abbeel, P. (2016). Backprop KF: Learning discriminative deterministic state estimators. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, pages 4376–4384.

Hannink, J., Kautz, T., Pasluosta, C., Barth, J., Schülein, S., Gabmann, K.-G., Klucken, J., and Eskofier, B. (2018). Mobile stride length estimation with deep convolutional neural networks. *IEEE J. Biomed. Health Inform.*, 22(2):354–362.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 770–778.

Hou, X. and Bergmann, J. (2020). Pedestrian dead reckoning with wearable sensors: A systematic review. *IEEE Sens. J.*, 21(1):143–152.

Huai, Z. and Huang, G. (2018). Robocentric visual-inertial odometry. In *Proc. Conf. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, pages 6319–6326.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. Int. Conf. Mach. Learn. (ICML)*, pages 448–456.

Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. (2015). Spatial transformer networks. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, pages 2017–2025.

Jatavallabhula, K. M., Iyer, G., and Paull, L. (2020). gradSLAM: Dense SLAM meets automatic differentiation. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2130–2137.

Jiménez, A., Seco, F., Prieto, J., and Guevara, J. (2010). Indoor pedestrian navigation using an INS/EKF framework for yaw drift reduction and a foot-mounted IMU. In *2010 7th Workshop on Positioning, Navigation. and Commun.*, pages 135–143.

Ju, H., Lee, J. H., and Park, C. G. (2018). Pedestrian dead reckoning system using dual IMU to consider heel strike impact. In *Proc. IEEE Int. Conf. Control Autom. Syst. (ICCAS)*, pages 1307–1309.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *Trans. ASME, J. Basic Eng.*, 82(Series D):35–45.

Kendall, A., Grimes, M., and Cipolla, R. (2015). PoseNet: A convolutional network for real-time 6-DOF camera relocalization. In *Proc. IEEE/CVF Int. Conf. Comput. Vision (ICCV)*, pages 2938–2946.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kitt, B., Rehder, J., Chambers, A., Schonbein, M., Lategahn, H., and Singh, S. (2011). Monocular visual odometry using a planar road model to solve scale ambiguity. In *Proc. Eur. Conf. Mobile Robot.*

Kloss, A., Martius, G., and Bohg, J. (2021). How to train your differentiable filter. *Auton. Robots*, 45(4):561–578.

Konda, K. R. and Memisevic, R. (2015). Learning visual odometry with a convolutional network. In *Proc. Int. Conf. Comp. Vis. Theory App. (VISAPP)*, pages 486–490.

Kuznietsov, Y., Proesmans, M., and Gool, L. V. (2021). CoMoDA: Continuous monocular depth adaptation using past experiences. In *Proc. IEEE Winter Conf. Appl. Comput. Vision (WACV)*, pages 2907–2917.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lee, B., Daniilidis, K., and Lee, D. (2015). Online self-supervised monocular visual odometry for ground vehicles. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 5232–5238.

Lee, S. J., Choi, H., and Hwang, S. S. (2020). Real-time depth estimation using recurrent CNN with sparse depth cues for SLAM system. *Int. J. Control Autom. Syst.*, 18(1):206–216.

Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., and Furgale, P. (2015). Keyframe-based visual–inertial odometry using nonlinear optimization. *Int. J. Robot. Res. (IJJR)*, 34(3):314–334.

Li, C. (2020). Towards end-to-end learning of monocular visual-inertial odometry with an extended Kalman filter. Master's thesis, University of Toronto (Canada).

Li, C. and Waslander, S. (2020). Towards end-to-end learning of visual inertial odometry with an EKF. In *Proc. Conf. Computer Robot Vision (CRV)*, pages 190–197.

Li, M. and Mourikis, A. (2013). High-precision, consistent EKF-based visual-inertial odometry. *Int. J. Robot. Res. (IJJR)*, 32(6):690–711.

Li, R., Wang, S., Long, Z., and Gu, D. (2018). UnDeepVO: Monocular visual odometry through unsupervised deep learning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 7286–7291.

Li, S. (2017). A review of feature detection and match algorithms for localization and mapping. In *IOP Conf. Series: Mat. Sci. Eng.*, volume 231.

Li, S., Wang, X., Cao, Y., Xue, F., Yan, Z., and Zha, H. (2020). Self-supervised deep visual odometry with online adaptation. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 6339–6348.

Lipton, Z. C., Berkowitz, J., and Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *ArXiv e-prints arXiv:1506.00019 [cs.LG]*.

Longuet-Higgins, H. C. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828):133–135.

Ma, M., Song, Q., Li, Y., and Zhou, Z. (2017). A zero velocity intervals detection algorithm based on sensor fusion for indoor pedestrian navigation. In *2017 IEEE Conf. Inf. Technol. Netw. Electron. Autom. Control (ITNEC)*, pages 418–423.

Maas, A. L., Hannun, A. Y., Ng, A. Y., et al. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. Int. Conf. Mach. Learn. (ICML)*, volume 30, page 3.

Maddern, W., Pascoe, G., Linegar, C., and Newman, P. (2017). 1 Year, 1000km: The Oxford RobotCar Dataset. *Int. J. Robot. Res. (IJRR)*, 36(1):3–15.

Mahjourian, R., Wicke, M., and Angelova, A. (2018). Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 5667–5675.

Maybeck, P. S. (1982). *Stochastic models, estimation, and control*. Academic press.

McCraith, R., Neumann, L., Zisserman, A., and Vedaldi, A. (2020). Monocular depth estimation with self-supervised instance adaptation. *arXiv preprint arXiv:2004.05821*.

McGee, L. A. (1985). *Discovery of the Kalman filter as a practical tool for aerospace and industry*, volume 86847. National Aeronautics and Space Administration.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). NeRF: Representing scenes as neural radiance fields for view synthesis. In *Proc. Eur. Conf. Comput. Vision (ECCV)*, pages 405–421.

Montiel, J. M. M., Civera, J., and Davison, A. J. (2006). Unified inverse depth parametrization for monocular SLAM. Proc. Robot. Sci. Syst. (RSS).

Moravec, H. P. (1980). *Obstacle avoidance and navigation in the real world by a seeing robot rover*. PhD thesis, Stanford University.

Mourikis, A. and Roumeliotis, S. (2007). A multi-state constraint Kalman filter for vision-aided inertial navigation. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, page 6.

Mur-Artal, R. and Tardós, J. (2017). ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras. *IEEE Trans. Robot.*, 33(5):1255–1262.

Nabavi, S., Hosseinzadeh, M., Fahimi, R., and Wang, Y. (2020). Unsupervised learning of camera pose with compositional re-estimation. In *Proc. IEEE/CVF Winter Conf. Appl. Comput. Vision*, pages 11–20.

Nilsson, J.-O. and Handel, P. (2014). Foot-mounted inertial navigation made easy. In *Proc. IEEE Int. Conf. Indoor Position. Indoor Navig. (IPIN)*.

Nilsson, J.-O., Skog, I., and Händel, P. (2012). A note on the limitations of ZUPTs and the implications on sensor error modeling. In *Proc. IEEE Int. Conf. Indoor Position. Indoor Navig. (IPIN)*.

Nistér, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognition (CVPR)*, volume 1, pages I–I.

Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Olivares, A., Ramírez, J., Górriz, J. M., Olivares, G., and Damas, M. (2012). Detection of (in)activity periods in human body motion using inertial sensors: A comparative study. *Sensors*, 12(5):5791–5814.

Olson, E. (2011). AprilTag: A robust and flexible visual fiducial system. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 3400–3407.

Park, S. Y., Ju, H., and Park, C. G. (2016). Stance phase detection of multiple actions for military drill using foot-mounted IMU. In *Proc. IEEE Int. Conf. Indoor Position. Indoor Navig. (IPIN)*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch. In *Workshop on Automatic Differentiation, Conf. Neural Inf. Process. Syst. (NeurIPS)*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830.

Peretroukhin, V., Clement, L., and Kelly, J. (2017). Reducing drift in visual odometry by inferring sun direction using a Bayesian convolutional neural network. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2035–2042.

Peretroukhin, V. and Kelly, J. (2018). DPC-Net: Deep pose correction for visual localization. *IEEE Robot. Autom. Lett. (RAL)*, 3(3):2424–2431.

Qiao, S., Wang, H., Liu, C., Shen, W., and Yuille, A. (2019). Micro-batch training with batch-channel normalization and weight standardization. *arXiv preprint arXiv:1903.10520*.

Qin, T., Li, P., and Shen, S. (2018). VINS-Mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Trans. Robot.*, 34(4):1004–1020.

Rantanen, J., Mäkelä, M., Ruotsalainen, L., and Kirkko-Jaakkola, M. (2018). Motion context adaptive fusion of inertial and visual pedestrian navigation. In *Proc. IEEE Int. Conf. Indoor Position. Indoor Navig. (IPIN)*, pages 206–212.

Ren, M., Pan, K., Liu, Y., Guo, H., Zhang, X., and Wang, P. (2016). A novel pedestrian navigation algorithm for a foot-mounted inertial-sensor-based system. In *Sensors*, volume 16, page 139.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In *Proc. Int. Conf. Medical Image Computing Computer-Assisted Intervention*, pages 234–241.

Roumeliotis, S. I. and Burdick, J. W. (2002). Stochastic cloning: A generalized framework for processing relative state measurements. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, volume 2, pages 1788–1795.

Scaramuzza, D. and Fraundorfer, F. (2011). Visual odometry [tutorial]. *IEEE Robot. Autom. Mag.*, 18:80–92.

Shu, C., Yu, K., Duan, Z., and Yang, K. (2020). Feature-metric loss for self-supervised learning of depth and egomotion. In *Proc. Eur. Conf. Comput. Vision (ECCV)*, pages 572–588.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. In *Conf. Learning Representations (ICLR)*.

Skog, I., Nilsson, J.-O., and Händel, P. (2010a). Evaluation of zero-velocity detectors for foot-mounted inertial navigation systems. In *Proc. IEEE Int. Conf. Indoor Position. Indoor Navig. (IPIN)*.

Skog, I., Nilsson, J.-O., Händel, P., and Rantakokko, J. (2010b). Zero-velocity detection—an algorithm evaluation. *IEEE Trans. Biomed. Eng.*, 57(11):2657–2666.

Solà, J. (2015). Quaternion kinematics for the error-state KF. Technical Report hal-01122406v5, Institut de Robòtica i Informàtica Industrial.

Solà, J., Deray, J., and Atchuthan, D. (2018). A micro lie theory for state estimation in robotics. Technical Report IRI-TR-18-01, Institut de Robótica i Informática Industrial.

Song, S., Chandraker, M., and Guest, C. (2015). High accuracy monocular SFM and scale correction for autonomous driving. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(4):730–743.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res. (JMLR)*, 15(1):1929–1958.

Stevens, B. L., Lewis, F. L., and Johnson, E. N. (2015). *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons.

Tang, C. and Tan, P. (2019). BA-Net: Dense bundle adjustment network. In *Conf. Learning Representations (ICLR)*.

Teed, Z. and Deng, J. (2020). DeepV2D: Video to depth with differentiable structure from motion. In *Conf. Learning Representations (ICLR)*.

Teed, Z. and Deng, J. (2021). DROID-SLAM: Deep visual SLAM for monocular, stereo, and RGB-D cameras. *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 34.

Teichmann, M., Weber, M., Zoellner, M., Cipolla, R., and Urtasun, R. (2018). MultiNet: Real-time joint semantic reasoning for autonomous driving. In *Proc. IEEE Intell. Veh. Symp. (IV)*, pages 1013–1020.

Tian, X., Chen, J., Han, Y., Shang, J., and Li, N. (2016). A novel zero velocity interval detection algorithm for self-contained pedestrian navigation system with inertial sensors. *Sensors*, 16(10):1578.

Tomasi, J., Wagstaff, B., Waslander, S. L., and Kelly, J. (2021). Learned camera gain and exposure control for improved visual feature detection and matching. *IEEE Robot. Autom. Lett. (RAL)*, 6(2):2028–2035.

Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. (1999). Bundle adjustment—a modern synthesis. In *Int. Workshop Vis. Algorithms*, pages 298–372.

Ummenhofer, B., Zhou, H., Uhrig, J., Mayer, N., Ilg, E., Dosovitskiy, A., and Brox, T. (2017). DeMoN: Depth and motion network for learning monocular stereo. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 5038–5047.

Vaishakh, P., Van Gansbeke, W., Dai, D., and Van Gool, L. (2020). Don't forget the past: Recurrent depth estimation from monocular video. In *Proc. Conf. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*.

Vijayanarasimhan, S., Ricco, S., Schmid, C., Sukthankar, R., and Fragkiadaki, K. (2017). SfM-Net: Learning of structure and motion from video. *arXiv preprint arXiv:1704.07804*.

Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Comput. Intell. Neurosci.*, 2018.

Wagstaff, B. and Kelly, J. (2018). LSTM-based zero-velocity detection for robust inertial navigation. In *Proc. IEEE Int. Conf. Indoor Position. Indoor Navig. (IPIN)*.

Wagstaff, B. and Kelly, J. (2021). Self-supervised scale recovery for monocular depth and egomotion estimation. In *Proc. Conf. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, pages 2620–2627.

Wagstaff, B., Peretroukhin, V., and Kelly, J. (2017). Improving foot-mounted inertial navigation through real-time motion classification. In *Proc. IEEE Int. Conf. Indoor Position. Indoor Navig. (IPIN)*.

Wagstaff, B., Peretroukhin, V., and Kelly, J. (2019). Robust data-driven zero-velocity detection for foot-mounted inertial navigation. *IEEE Sens. J.*, 20(2):957–967.

Wagstaff, B., Peretroukhin, V., and Kelly, J. (2020). Self-supervised deep pose corrections for robust visual odometry. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2331–2337.

Wagstaff, B., Peretroukhin, V., and Kelly, J. (2022a). On the coupling of depth and egomotion networks for self-supervised structure from motion. *IEEE Robot. Autom. Lett. (RAL)*, 7(3):6766 – 6773.

Wagstaff, B., Wise, E., and Kelly, J. (2022b). A self-supervised, differentiable Kalman filter for uncertainty-aware visual-inertial odometry. In *Proc. IEEE 2022 Conf. Adv. Intell. Mechatron. (AIM)*.

Wahlstrom, J., Skog, I., Gustafsson, F., Markham, A., and Trigoni, N. (2019). Zero-velocity detection—a Bayesian approach to adaptive thresholding. *IEEE Sens. Lett.*

Walder, U. and Bernoulli, T. (2010). Context-adaptive algorithms to improve indoor positioning with inertial sensors. In *Proc. IEEE Int. Conf. Indoor Position. Indoor Navig. (IPIN)*.

Wang, C., Buenaposada, J., Zhu, R., and Lucey, S. (2018a). Learning depth from monocular videos using direct methods. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 2022–2030.

Wang, J., Zhang, G., Wu, Z., Li, X., and Liu, L. (2020). Self-supervised joint learning framework of depth estimation via implicit cues. *arXiv preprint arXiv:2006.09876*.

Wang, L., Wang, Y., Wang, L., Zhan, Y., Wang, Y., and Lu, H. (2021). Can scale-consistent monocular depth be learned in a self-supervised scale-invariant manner? In *Proc. IEEE/CVF Conf. Comput. Vision (ICCV)*, pages 12727–12736.

Wang, R., Pizer, S., and Frahm, J. (2019). Recurrent neural network for (un-) supervised learning of monocular video visual odometry and depth. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 5555–5564.

Wang, S., Clark, R., Wen, H., and Trigoni, N. (2017). DeepVO: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2043–2050.

Wang, X., Zhang, H., Yin, X., Du, M., and Chen, Q. (2018b). Monocular visual odometry scale recovery using geometrical constraint. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 988–995.

Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4):600–612.

Watson, J., Mac Aodha, O., Prisacariu, V., Brostow, G., and Firman, M. (2021). The temporal opportunist: Self-supervised multi-frame monocular depth. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 1164–1174.

Wei, P., Hua, G., Huang, W., Meng, F., and Liu, H. (2021). Unsupervised monocular visual-inertial odometry network. In *Proc. Int. Joint Conf. Artificial Intell. (IJCAI)*, pages 2347–2354.

Wei, X., Zhang, Y., Li, Z., Fu, Y., and Xue, X. (2020). DeepSFM: Structure from motion via deep bundle adjustment. In *Proc. Eur. Conf. Comput. Vision (ECCV)*, pages 230–247.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proc. IEEE*, 78(10):1550–1560.

Woodman, O. (2007). An introduction to inertial navigation. Technical Report 696, University of Cambridge.

Wu, Y. and He, K. (2018). Group normalization. In *Proc. Eur. Conf. Comput. Vision (ECCV)*, pages 3–19.

Xue, F., Wang, X., Li, S., Wang, Q., Wang, J., and Zha, H. (2019). Beyond tracking: Selecting memory and refining poses for deep visual odometry. In *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 8575–8583.

Xue, F., Zhuo, G., Huang, Z., Fu, W., Wu, Z., and Jr, M. A. (2020). Toward hierarchical self-supervised monocular absolute depth estimation for autonomous driving applications. In *Proc. Conf. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, pages 2330–2337.

Yang, N., Stumberg, L., Wang, R., and Cremers, D. (2020). D3VO: Deep depth, deep pose and deep uncertainty for monocular visual odometry. In *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 1281–1292.

Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. (2010). Deconvolutional networks. In *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 2528–2535.

Zhang, J., Sui, W., Wang, X., Meng, W., Zhu, H., and Zhang, Q. (2021). Deep online correction for monocular visual odometry. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 14396–14402.

Zhang, Y. and Yang, Q. (2021). A survey on multi-task learning. *IEEE Trans. Knowl. Data Eng.*

Zhao, C., Yen, G., Sun, Q., Zhang, C., and Tang, Y. (2020). Masked GANs for unsupervised depth and pose prediction with scale consistency. *IEEE Trans. Neural Netw. Learn. Syst.*

Zhou, B., Krähenbühl, P., and Koltun, V. (2019a). Does computer vision matter for action? *Sci. Robot.*, 4(30).

Zhou, D., Dai, Y., and Li, H. (2019b). Ground-plane-based absolute scale estimation for monocular visual odometry. *IEEE Trans. Intell. Transp. Syst.*, 21(2):791–802.

Zhou, L., Luo, Z., Shen, T., Zhang, J., Zhen, M., Yao, Y., Fang, T., and Quan, L. (2020). KFNet: Learning temporal camera relocalization using Kalman filtering. In *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 4919–4928.

Zhou, T., Brown, M., Snavely, N., and Lowe, D. (2017). Unsupervised learning of depth and ego-motion from video. In *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 1851–1858.

Zhou, Y., Barnes, C., Lu, J., Yang, J., and Li, H. (2019c). On the continuity of rotation representations in neural networks. In *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognition (CVPR)*, pages 5745–5753.

Zou, Y., Ji, P., Tran, Q.-H., Huang, J.-B., and Chandraker, M. (2020). Learning monocular visual odometry via self-supervised long-term modeling. In *Proc. Eur. Conf. Comput. Vision (ECCV)*, pages 710–727.