UNIVERSITY OF CALIFORNIA

Los Angeles

Uncertainty Calibration for Robotic Navigation and Vision

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Stephanie Tsuei

2023

ABSTRACT OF THE DISSERTATION

Uncertainty Calibration for Robotic Navigation and Vision

by

Stephanie Tsuei
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2023
Professor Stefano Soatto, Chair

We experimentally demonstrate that an uncertainty-aware framework for robotic navigation and vision is able to navigate a robotic platform around an obstacle without choosing an overly long and conservative path. The framework contains many interconnected experimental pieces, including monocular visual-inertial odometry (VIO) based on an Extended Kalman Filter (EKF), a recurrent neural network that predicts future covariance estimates from the EKF, a model predictive controller that uses uncertainty in its cost function, and an object detection network. Each interconnected piece is an algorithm that makes assumptions about its inputs, which are the outputs of another piece. The rest of the thesis is a systems validation exercise that examines several of these assumptions for validity and finds that they are largely not true. First, we learn that uncertainty estimates of the commonly used EKF are overconfident, but that the overconfidence is systematic and correctable. Next, we examine the distribution of feature track errors and find that not only are the errors not zero-mean Gaussian, the errors are dependent on motion type, speed, and the type of feature tracking algorithm used. We then quantify the effect of attribution errors, Gaussian noise, and drift on performance and uncertainty estimates of the VIO algorithm used in the framework. Finally, we attempt to characterize the uncertainty of image classification networks in a manner appropriate for online navigation. To our knowledge, the proposed architecture is new and this dissertation is the first time a systems validation exercise has focused on uncertainty estimation.

The dissertation of Stephanie Tsuei is approved.

Cho-Jui Hsieh

Quanquan Gu

Mark Milam

Paulo Tabuada

Stefano Soatto, Committee Chair

University of California, Los Angeles

2023

To my dad,

who never once doubted that

I would write this thesis someday

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Although I did not officially start this PhD program until the Fall 2018, it would be more accurate to say this work started while I was working in an autonomous systems research group at Northrop Grumman. It was there that Mark Milam, a member of my committee, told me something along the lines of *the most important thing about any kind of autonomy deployed is that we can verify and validate it.* The autonomy group consisted of engineers and scientists who not only made the systems they worked on function well, but took the time to understand why and check all the details over multiple times. Mark, Elaine, Michael, Robert, and Ken – thank you for being the role models I needed early in my career. I look forward to working with you again. Many thanks also go to Tom Pieronek for setting up my fellowship and giving me a rare opportunity to coexist in academia and industry.

During that time, I also met Paulo Tabuada, another member of my committee, when he consulted for us on issues in verification and validation during his sabbatical. As a consultant, and my secondary adviser at UCLA, he always listened very carefully to whatever issues I was running into, gave thoughtful feedback, and pointed me to pockets of scientific literature that I was unaware of. I also appreciate Paulo letting me use the equipment in his lab and his students as sounding boards during my PhD.

(Chronologically,) I next thank my adviser, Stefano Soatto. The environment Stefano created and encouraged – a motley group of people working towards a deep understanding of a myriad of computer vision tasks – complemented my past experience at Northrop Grumman surprisingly well. My research direction cemented quickly in my first year when he pointed me to the literature on uncertainty quantification. I appreciate his guidance in helping me transition from working in the more exact science of control systems to the empirical science that computer vision has become. I am grateful for his patience as I struggled to grasp concepts in visual-inertial odometry that proved to be more difficult and subtle than I originally thought.

Next, I thank my peers in the Stefano's Vision Lab, Paulo's CyPhy Lab, and a few others for their camaraderie and all their help over the years. In particular, I want to thank Xiaohan

Fei for taking on the daunting task of rewriting Corvis. The rewrite made the experiments in Chapters 2-5 much easier. Special thanks also go to Aditya Golatkar for many late nights working on Scene Uncertainty with me; to Alex Wong for making sure that I could build a lab machine, for helping me debug my research, and for an endless stream of research ideas; and to Alex Schperberg and Kenny Chen for bringing me the experiment in Chapter 2.

My last professional thanks go to Professors Quanquan Gu and Cho-Jui Hsieh for serving on my committee. I know that in the age of deep learning, my work is an anomaly in the computer science department and well out of your wheelhouse.

Last but not least, I would like to thank the friends and family that made sure I was sane, fed, and rested during these hectic years. Thank you for believing that I could finish this PhD through the times I wasn't sure I believed in myself. I couldn't have done it without you.

VITA

2008 - 2012	B.S. in Mechanical Engineering, California Institute of Technology
2012 - 2014	M.S. in Control and Dynamical Systems, California Institute of Technology
2014 - 2016	Guidance, Navigation, Controls Engineer, Northrop Grumman
2016 - present	Associate Researcher, Northrop Grumman
2018 - present	PhD. Student in Computer Science, University of California, Los Angeles

PUBLICATIONS

A. Schperberg, **S. Tsuei**, S. Soatto, and D. Hong, *SABER: Data-Driven Motion Planner for Autonomously Navigating Heterogeneous Robots*, IEEE Robotics and Automation Letters, vol. 6, no. 4, pp. 8086–8093, Oct. 2021, doi: 10.1109/LRA.2021.3103054.

S. Tsuei, S. Soatto, P. Tabuada, and M. B. Milam, *Learned Uncertainty Calibration for Visual Inertial Localization*, in 2021 IEEE International Conference on Robotics and Automation (ICRA), May 2021, pp. 5311–5317. doi: 10.1109/ICRA48506.2021.9561179.

A. Schperberg, K. Chen, **S. Tsuei**, M. Jewett, J. Hooks, S. Soatto, A. Mehta, and D. Hong, *Risk-Averse MPC via Visual-Inertial Input and Recurrent Networks for Online Collision Avoidance*, in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2020, pp. 5730–5737. doi: 10.1109/IROS45743.2020.9341070.

A. Wong, X. Fei, **S. Tsuei**, and S. Soatto, *Unsupervised Depth Completion From Visual Inertial Odometry*, IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 1899–1906, Apr. 2020, doi: 10.1109/LRA.2020.2969938.

Y. Shoukry, P. Tabuada, **S. Tsuei**, M. B. Milam, J. W. Grizzle, and A. D. Ames, *Closed-form controlled invariant sets for pedestrian avoidance*, in 2017 American Control Conference (ACC), May 2017, pp. 1622–1628. doi: 10.23919/ACC.2017.7963185.

S. Tsuei and M. B. Milam, *Trajectory generation for constrained differentially flat systems with time and frequency domain objectives*, in 2016 IEEE 55th Conference on Decision and Control (CDC), Dec. 2016, pp. 4172–4177. doi: 10.1109/CDC.2016.7798902.

CHAPTER 1

Introduction

1.1 Motivation

Algorithms and software to be deployed in safety-critical systems require extensive verification and validation. Verification is the process of checking that the algorithm behaves as intended. Validation is the process of checking whether or not the assumptions made by the algorithm are true. Industrial standards, such as DO-178C and ISO 26262, identify the tests that must be completed before any software may be legally deployed on an airplane or an automobile, respectively. Unsurprisingly, the difficulty of meeting those standards increases with complexity of the software. Software errors have already caused loss of life and major product recalls.

To reduce the amount of testing required while still being sure that the software can run, we would ideally turn to mathematical proof and formal methods. Mathematical proof typically comes from designing an algorithm to meet a specific specification or a clever search for a counterexample, a worst-case outcome. In computer systems domains where the world can be modeled as Boolean variables, these ideas have already been deployed in real-world applications. [Hol13, Lau10]. They have also seen some success in the verification and synthesis of controllers of hybrid systems, or systems with a combination of discrete and continuous states [ALF11, SNS18].

However, fundamental problems, or problems that must be solved before the usual curses of dimensionality and computational complexity become relevant, appear when applying these ideas to systems that use any kind of computer vision. The first issue arises from the use of statistical algorithms, especially deep learning, in computer vision tasks. Not only

are deep neural networks large and difficult to interpret, they are driven by imperfect data and are not designed for absolute guarantees on a specific specification. Nevertheless, their performance on vision tasks such as semantic segmentation, depth completion, and object detection, is far better than any method that does not use deep learning.

The second fundamental issue is that using mathematical proof and formal methods requires a mathematical description of the "environment". Such a description is possible for an industrial robot arm whose interaction of the world can be measured by force sensors and joint encoders.[1] However, the interaction of light and matter that generates images, and the processing of images into useful information is not easily described with a closed-form equation $y = h(x)$ because the visual world is too complex and diverse. Indeed, the formal methods community has begun to develop methods for analyzing whether or not the inputs and outputs of neural networks for control systems can satisfy a particular specification [KHI19, TYM20]. Their efforts to lessen the curse of dimensionality do not change the fact that we do not know how to write an appropriate specification for a vision network.

1.2 A Proposed Solution and Thesis Outline

Our proposed solution is to start from the premise that absolute guarantees are impossible when computer vision is part of a system and instead design an autonomous system to be uncertainty-aware. We start with a demonstration of a proposed architecture. The rest of this dissertation is an exercise in systems validation, for it is not enough that a systems architecture be comprised of uncertainty-aware algorithms. If the assumptions that each algorithm makes about its inputs are incorrect, then any successful experiment is successful not because of correctness, but because of luck. Uncertainty-awareness is not a new idea — Partially Observable Markov Decision Processes (POMDPs) were first introduced in 1965 [Ås65] — and neither is the examination of separate components integrated into a system.

[1]The description, however, will likely be conservative because possibly large, but low-probability disturbances will be deemed just as likely as smaller, but commonplace disturbances.

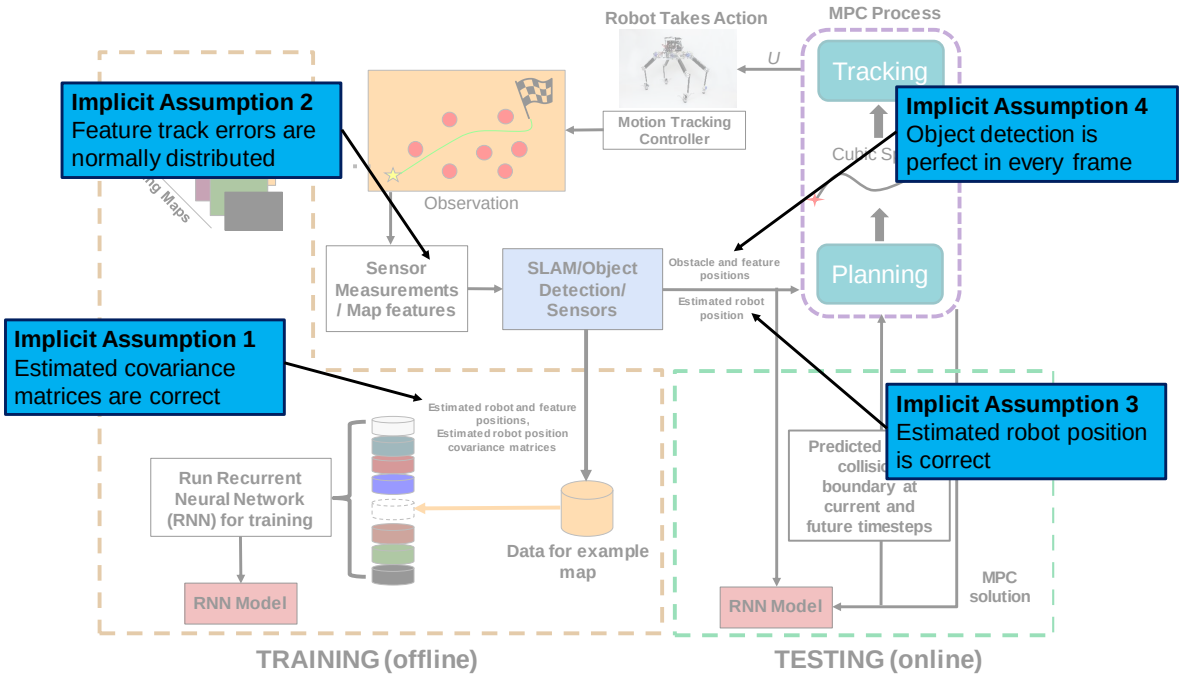


Figure 1.1: **An overview of this thesis.** Chapter 2. describes the uncertainty-aware systems architecture in the background. The algorithms in the systems architecture make four implicit assumptions, noted in blue boxes. Chapters 3, 4, 5, and 6 then examine the correctness of each assumption and explores possible solutions.

To our knowledge, the proposed architecture is new and this dissertation is the first time a systems validation exercise has focused on uncertainty estimation.

The proposed architecture, labeled with the four assumptions that we examine, is pictured in Figure 1.1. The heart of the architecture is an uncertainty-aware model predictive controller that depends on position and covariance estimates from a Simultaneous Localization and Mapping (SLAM) algorithm, and the outputs of an object detection neural network. Chapter 2 describes the model predictive controller and the architecture in more detail. Chapter 3 finds that the covariance matrices estimated by the SLAM algorithm are systematically inaccurate, but can be corrected. Chapter 4 finds the feature tracks used in the SLAM algorithm are not always Gaussian, as commonly assumed, and characterizes their distribution conditional on motion, speed, lighting, and the feature tracking algorithm extracting them. Chapter 5 then characterizes the effects of feature track errors on the estimated state, which the model predictive controller assumes is correct. Chapter 6 presents a preliminary study of uncertainty quantification of neural networks with semantic vision tasks. Finally, Chapter 7 ends with some concluding remarks. The SLAM software used throughout the experiments in this dissertation, XIVO, is described in detail in Appendix A.

1.3 Related Work

1.3.1 Active SLAM: Planning Under Uncertainty

Necessary requirements for robots to autonomously perform complex tasks such as search and rescue operations and unknown environment exploration include, but are not limited to, online low-level feedback controls, localization, vision, motion planning, high-level reasoning, and reasoning under uncertainty. Currently, all individual components are well-developed, but integrating multiple pieces together into a single system, especially for environments that are not well-known, has proven to be a daunting challenge because of issues related to robustness [ACH18]. For example, simultaneous planning, localization, and mapping

(SPLAM, or "Active SLAM") is an active area of research that attempts to satisfy some of these requirements. The main challenges to Active SLAM consist of planning under uncertainty in an acceptable amount of time, bridging the gap between sensor data ("semantic mapping"), and ensuring that it is robust enough for a complex platform. Because of these challenges, there are many works addressing a subset of Active SLAM, namely simultaneous localization and planning [AAC15], and SLAM [MMT15a,JS11,ESC14]. Other Active SLAM works use very simple research platforms [KIS19] or were only tested in simulation [LHD06].

There are two common frameworks to address the problem of planning under uncertainty, which are explicitly modeling the posterior distributions in a Bayesian setting [FBH18] or using a partially-observable Markov decision process (POMDP) [KLC95]. However, the Bayesian setting is only computationally tractable for the simplest cases (e.g. Gaussian prior and Gaussian observations); POMDPs suffer from the curses of history and dimensionality and do not sufficiently model an agent's future intent [ECH18]. Recent work addressing Active SLAM using POMDPs also either lacks mapping capability [AAC15], requires an inordinate amount of computational resources [Van14]. As a consequence, there is a need for new path planning architectures for unknown and uncertain environments that addresses the concerns of belief space planning or provides alternative methods that can be ubiquitously applied on most robotic systems.

1.3.2 Uncertainty Calibration of Extended Kalman Filters

While deep learning has dominated much of the computer vision literature in recent years, "traditional" filtering methods still perform better in localization problems that use one or more cameras, such as Visual Odometry (VO) and Visual Inertial Odometry (VIO). This is because the filters hard-code known nonlinear kinematic models that are difficult to learn from data, although deep neural networks are often used to learn feature representations of measured data. The Extended Kalman Filter (EKF) is the most common because it is so simple, even though complex non-linearities in the rotational dynamics and patent violation of the Gaussian assumption in the visual measurements remove all guarantees of

convergence and accuracy. In practice, the EKF provides accurate state estimates $\hat{x}$ and overconfident covariance estimates $\hat{P}$ [BNG06]. To improve the covariance estimates of VO problems, [VBB13] and [LOV18] learn time-dependent measurement noise covariances Q while [DL20] uses a deep convolutional network to correct $\hat{x}$ and $\hat{P}$ directly from images. On the other hand, [HMR09, KHB13, HP18, HKB14, BBB19, ZWS17, FCD17] improve VIO covariance estimation by using filters specific to VIO. The accuracy of covariance estimates are evaluated by tabulating the percentage of timesteps where the estimation error is within the 1,2,3-σ bounds dictated by the estimated covariance $\hat{P}$.

1.3.3 Characterization of Feature Track Uncertainty

Performance of feature detectors and descriptors conditional on nuisances. The main metric used to evaluate feature detectors is *repeatability* [MTS05], or the probability that a feature detector will detect the same feature across multiple images of the same scene under different illuminations and viewpoints. Other metrics are *entropy* [HDF12], the spread of detected features over an image, and *recall* [ADS12], the number of features that are likely "matchable" to features in another image of the same scene. On the other hand, the primary metrics used to evaluate feature descriptors are *precision* and *recall*, calculated using pairs of "matches" that are found using the descriptor [MS05]. The evaluation of feature detectors requires multiple images of the same scene. The evaluation of descriptors originally used the same datasets as the evaluation of detectors. To disentangle the problem of detecting features from the evaluation of feature description, two comprehensive datasets of image patches was released in 2017 [BLV17, MHZ17]. At around the same time, [SHS17b] evaluated both learned and handcrafted feature detectors and descriptors. Of most interest to us are [HDF12], which used a small dataset containing pure rotation, pure scaling, and illumination changes to evaluate the performance of various detector/descriptor combinations condition on each, [ZCY20], which extended the datasets used in [HDF12], and [ADS12], which evaluated the performance of feature detectors conditional on change in view angle and lighting condition. Tangentially interesting are [HS12], which released a dataset of image pairs that are geometrically consistent, but contain large changes in style (e.g. summer vs.

winter) and lighting; and [SMT18], which contains groups of image sequences with similar motions, but large outdoor illumination changes.

Learning or Fitting a Covariance Matrix to Feature Tracks. Early works sought to compute covariance of feature location using information in the RGB image. [KK01] approximated the covariance with the Hessian of the image centered at the feature point was the covariance of a detected feature – the idea is that the sharper the curvature given by the Hessian, the more likely a convolutional filter will find the correct location of the feature. [NH02] contains a sum-of-squared-differences formula for computing feature track covariance. [ZGS09] contains a formula for computing the covariance matrix of SIFT and SURF features. Later on, [SKY15] and [WM17] present two methods to model the mean and covariance of Lucas-Kanade feature tracks. With the exception of [SKY15], which assumes that the location of a feature track could be a Gaussian Mixture Model, all other models assume that uncertainty is zero-mean Gaussian.

1.3.4 Uncertainty Quantification of Deep Neural Networks

There is a large amount of literature on uncertainty quantification for neural networks, especially in recent years. Uncertainty, and the methods that estimate and measure it, is often sorted into one of two categories: *epistemic*, or a model's lack of knowledge, and *aleatoric*, uncertainty inherent in the data-generating process. Many works, including ours, do not fit cleanly into either category. Below, we sort a sample of methods for uncertainty quantification in image classification by answering the question *it estimates uncertainty with respect to what?*

To the weights. The data-fitting capacity of modern deep networks is so large that all epistemic uncertainty can be captured as uncertainties in the values of the weights. This is formalized in Bayesian neural nets [Nea12, GDS20, WVB18, RTS18]. At inference time, the output prediction and uncertainty is computed using Bayes rule, rather than a simple forward pass. This is computationally intractable for models the size of modern deep nets, but can

be approximated using Monte-Carlo test-time dropout [GG16], deep ensembles [LPB17], or even a mix of the two [DBB21].

To the pixels in the image. In image classification, the only aleatoric uncertainty considered is the noise caused by the cameras capturing the image. So far, there has been far less work estimating aleatoric uncertainty than epistemic uncertainty. Two methods that address this problem are Assumed Density Filtering [GR18] and test-time dropout [KG17]. [LSS20] implements both [GR18] and [GG16] to estimate both aleatoric and epistemic uncertainty.

To a test dataset. Works on calibration ignore the distinction between aleatoric and epistemic uncertainty and focus on achieving the frequentist notion of *calibration* — confidence scores are calibrated when exactly X% of all samples given a confidence of X% are correctly classified. The quality of a network's confidence scores is given by the Expected Calibration Error (ECE) metric and several other variants [NDZ19].

It was first noted in [GPS17] that modern deep neural networks trained with the cross-entropy loss are incredibly overconfident when the maximum value of the softmax vector is taken as a measure of confidence. [GPS17] compared and proposed several methods for post-hoc adjustment of the softmax vector. It is also common to use Monte-Carlo dropout [GG16] and deep ensembles [LPB17] to compute more calibrated softmax vectors as well. More recent work has focused on ensuring that calibrated confidences remain calibrated in the presence of OOD data [OFR19, TGE21, ZL21]. Other work has extended calibration to conformal prediction [ABJ20, BCR21], where a model predicts a set of classes whose probabilities sum to a desired confidence level rather than a single class.

Other categories. [MG18, SKK18] incorporate the likelihood that a test image is from the same distribution as the training images into the training and inference process. [HOZ21] uses GANs to improve upon [SKK18]. A common feature of these methods is interpreting the softmax vector as the parameters of a Dirichlet distribution rather than a categorical distribution. [JCS20] models uncertainty to specifically address noise and errors in ground-

truth labels.

To the scene. The counterfactual ideas we present in this work have appeared elsewhere, but have not been formalized. [HD18] and [SDR19] find that image classifiers are not able to correctly classify all the frames of short videos. [WLA19] use test-time data augmentation in the same way we do, except in the context of segmentation of medical images. Related in spirit to our approach is [ABA20], which adds a layer of counterfactual reasoning to quantify the sensitivity of uncertainty estimates to changes in the input.

1.4 Summary of Contributions

Uncertainty-aware motion planning, visual-inertial odometry, and estimation of uncertainty are not new ideas. However, their combined use in safety-critical systems is limited for two reasons: computational complexity and a lack of trust[2] in SLAM and uncertainty estimation.

Chapter 2 address computational complexity in a manner different from the papers noted above, which ignore mapping or planning. Instead, it uses a Recurrent Neural Network (RNN) to predict future belief states quickly so that a model predictive controller can use belief states in its planning problem. Although not included in this thesis, we have also extended the ideas in Chapter 2 to a cooperative multi-agent scenario [STS21].

The rest of this dissertation addresses the lack of trust in SLAM and uncertainty estimation. Lack of trust in SLAM stems from the fact that state estimation accuracy is not robust [CCC16]; the existence of recently released SLAM benchmarks [WZW20], [ZHF22] corroborates this. Lack of trust in uncertainty estimation stems from the fact that measuring the accuracy of an uncertainty estimate is difficult outside of simulation and that Monte-Carlo simulations have already shown that estimates are commonly overconfident

[2]There are many definitions of "trust" in the literature on psychology and human machine interaction. For the purposes of this dissertation, a system is *trusted* if its performance is predictable, reliable, and verifiable, i.e. it has to work "well" in "enough" operating conditions *and* human engineers must understand how and why it works well when it does. Examples of trustworthy technologies are GPS, modern operating systems, airplanes, and automobiles.

(see references in Section 1.3.2). Since the architecture of Chapter 2, and most real-world systems, are too complex for provable guarantees, we address the lack of trust using empirical approaches.

Chapter 3 makes two contributions. The first is a method for calculating ground-truth uncertainty estimates in real-world data without Monte-Carlo simulations by assuming ergodicity. An accompanying method to verify whether or not the ergodic assumption is true is also presented. This method is related to those in [HMR09, FCD17] to evaluate the calibration of estimates from a state estimator, except without the need for Monte-Carlo experiments. The second contribution of Chapter 3 is the use of deep learning to correct overconfident uncertainty estimates. While it is not the first work to use supervised machine learning in this manner, its scope encompasses all Extended Kalman Filters and it uses the smallest neural network possible – the only input to the network is the estimated covariance matrix and the network is a feedforward neural network. On the other hand, [DL20] uses a convolutional network whose input is the RGB image used for state estimation. This second contribution illustrates that the overconfidence in the Extended Kalman Filter's covariance estimates is systematic.

Chapters 4 and 5 addresses the last of robustness in visual and visual-inertial SLAM. Rather than trying to create a new system and experimentally demonstrating that the new system is better than all existing ones using a research benchmark, they search for the root causes of this lack of robustness. In Chapter 4, we show that the common assumption that measurement errors are zero-mean Gaussian is not true when the measurements are feature tracks. In particular, we show that measurement errors have nonzero mean and covariance that is dependent on time, the feature tracking algorithm, and the type of motion. We were unable to find any existing works that attempted to characterize the error distribution of feature tracks. Related experimental characterizations of the performance of feature trackers are listed in Section 1.3.3.

In Chapter 5, we then attached our in-house VIO system, XIVO, to a simulation of a pinhole camera in a point cloud world. This allowed us full control over measurement errors and attribution errors, and allowed us to characterize how each type of measurement

error and attribution error affected XIVO's performance and uncertainty. Most existing works on realistic SLAM systems benchmark performance on real-world datasets of motion sequences. On the other hand, works that use simulations, such as [ZWS17] often test a complex algorithm whose software is not written for deployment and a simulation that only adds Gaussian noise to simulated visual measurements. Using a simulation enables us to analyze the effects of drift, Gaussian noise, and attribution errors individually, as they are all entangled in real-world data. We find that performance degradation due to Gaussian noise and drift is graceful, while the degradation due to attribution errors is exponential. We find that even with small levels of drift and Gaussian noise, the probability of poor performance is greater than 1 in 100, unacceptably high for a safety-critical system. At the end of Chapters 4 and 5, we offer suggestions for building off of our work to improve SLAM.

Finally, in Chapter 6, we tackle the problem of integrating a neural network designed for a semantic task into the architecture of Chapter 2 in an uncertainty-aware fashion. In Chapter 2, an objection detection neural network was integrated like a black box sensor that would return perfect results on every frame. The object detection network successfully detected an obstacle in enough frames to ensure success of the experiment, but not all frames. This indicates that the object detection network needs to be integrated like a noisy sensor, not a perfect sensor. In order to integrate it like a noisy sensor, we need to be able to model its uncertainty. To simplify the problem, we focused on image classification instead of object detection, and formalize the idea of *uncertainty with respect to the scene*, or how much we expect an image classification to vary given multiple images of the same environment. There are many existing works on the uncertainty of an image classifier; we list a selection in Section 1.3.4. Most of them use a definition of uncertainty that does not capture *uncertainty with respect to the scene*. The few that do either are not focused on uncertainty [ABA20], or use a subset of our ideas on a different problem [WLA19].

CHAPTER 2

Risk-Averse MPC via Visual-Inertial Input and Recurrent Networks for Online Collision Avoidance

Our uncertainty-aware architecture is a multifaceted approach that uses model predictive control (MPC), SLAM[1], and recurrent neural network (RNN) algorithms to address the problem of Active SLAM and account for uncertainties in both current and future robot positions. Our architecture is based on MPC because MPC operates online, continually satisfies the dynamic state of the robot over a prediction horizon N, and naturally offsets estimation errors [XOT19]. The MPC is augmented to be "risk-averse" by considering uncertainty in position from timestep k to $k+N$. This uncertainty is inferred by an RNN, which has been demonstrated to handle time series data, account for temporal factors that directly affect predictions, have shown promise in modeling complex interactions between agents and their environment [ECH18, YWL19] and previously applied to MPC but for industrial processes [LLC19]. Our RNN model is trained on the positional covariance estimations of a visual-inertial odometry (VIO) system taking readings from an inertial measurement unit (IMU) and camera data as input. By considering the current and future positional uncertainties in the MPC optimization problem, our method can solve for more optimal control actions at each timestep. To facilitate object avoidance, we additionally incorporate an object detection pipeline that uses a deep convolutional neural network (CNN) to recognize obstacles and a feature detector with RGB and depth images to estimate the distance and size of nearby obstacles. We show that by using a trained RNN model to infer positional uncertainties at future timesteps, a robot can demonstrate more evasive behavior to better

[1]In this paper, "SLAM" includes visual-inertial odometry with sparse mapping in addition to algorithms that produce denser maps.

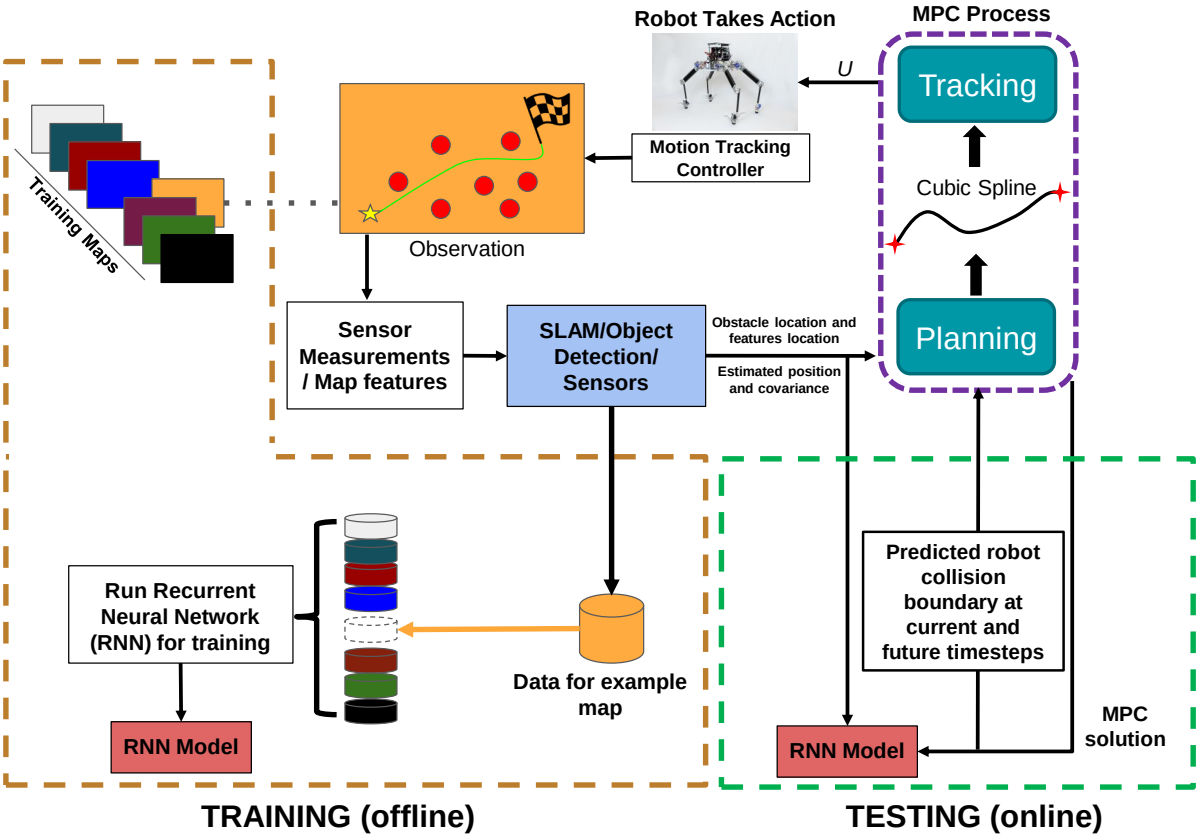


Figure 2.1: **Architecture Overview.** This figure demonstrates the training and testing procedures of our method. In training, we first select different maps, where obstacles in each map are randomly distributed. A simulation where the robot moves from an initial to a goal position is executed on this map. At each timestep an observation is taken (e.g., camera or on-board sensor data). These measurements are used as the input to our SLAM/Object Detection/Sensors system, which estimate the current position and uncertainty in position of the robot, and also location and size of obstacles. MPC accounts for this information and produces outputs entered into our motion tracking controller. For every map at every timestep, the current observations, state position, and positional uncertainty (among other variables outlined in Section 2.1.5) are entered into a large database to produce our RNN model. Lastly, in the testing phase, RNNs can predict the positional uncertainty (which provide our collision boundaries) of the robot at future timesteps of the MPC prediction horizon.

guarantee collision avoidance without becoming too conservative. Our linearized path planning framework is applied and tested on a complex quadruped robot, which demonstrates our algorithm's robustness and efficiency in computation, showing the feasibility of extending our work to a wide range of robotic platforms. In short,

1. We evaluate the feasibility of an online end-to-end path planner that unifies MPC, SLAM, RNN, and an object detector using CNNs to generate paths for unknown and uncertain environments using a non-linear programming solver.

2. We verify that our quadrupedal robot, ALPHRED [HAY20], avoids collisions and computes a shorter trajectory while maintaining safety using our method as compared to a more conservative and naive planner.

3. We use RNNs to estimate positional uncertainties at all future timesteps of the MPC's prediction horizon.

4. We integrate all components into a high-fidelity simulation using the quadruped dynamics of ALPHRED (Figure 2.4). Additionally, we test all components individually either online or offline using hardware (Figure 2.7).

In the following sections, we will explicitly refer to the simulation or hardware data. The main difference between the model of ALPHRED in simulation versus hardware is that in simulation the model is equipped with an idealistic RGB + dense depth Microsoft Kinect camera, while the actual hardware is equipped with Intel's Realsense D435i. The idealistic camera publishes both RGB and dense depth images at arbitrarily fast speeds while the RealSense publishes RGB images at 30Hz and dense depth images at only 2Hz.

2.1 Methods

In this section, we provide an overview of our architecture and how our risk-averse MPC propagates uncertainty through its finite time horizon trajectory. In Section 2.1.1, we provide a high-level overview of our path planning algorithm. In Section 2.1.2, we describe our

MPC's mathematical framework for planning and tracking. In Section 2.1.3, we describe the constraints used in our cost functions. In Section 2.1.4, we describe our object detection system using CNNs and keypoint detection on RGB and depth images, and finally in Section 2.1.5, we detail our RNN training and inference procedures (utilizing our SLAM algorithm) for predicting future positional uncertainties used to create collision boundaries.

2.1.1 Architecture Overview

Our path planner is formulated as an MPC optimization problem using a non-linear programming solver [AGH19]. We divide our MPC framework into a planning phase and a tracking phase, with different cost functions for each. In the planning phase, the goal of our MPC is to create waypoints that move a robot closer to a desired position while detecting and avoiding obstacles through measurement updates. Specifically, the object detection algorithm feeds the MPC with the position and size of surrounding obstacles, while the positional uncertainty of the robot at all future timesteps in the MPC prediction horizon is inferred by RNNs. In the tracking phase, we discretize the generated path into segments of fixed temporal length using a cubic polynomial to create a smooth reference trajectory. MPC is used to track this reference trajectory and outputs our desired planar velocity values (v_d and $\dot{\psi}_d$). These velocity values are used by our motion tracking controller to generate stable footstep trajectories. Note that dividing our MPC formulation into two phases facilitates lower computation time, and allows for separate control on waypoint generation and creation of custom reference trajectories if desired (see Algorithm 1, Fig. 2.1, or our accompanied video[2] for a general overview of our path planning architecture).

[2] `https://www.youtube.com/watch?v=faurQ1LpNVI`

Algorithm 1: Risk-Averse MPC

Initialize state X, control U, dt_{plan}, dt_{track}, horizon N, robot collision boundary $r_{\Sigma_{k:k+N}}$, and timestep k

`// Planning Phase (waypoints to goal)`

while $\|X_{curr} - X_{goal}\|_2 > 0$ **do**

  $X, U_{sols} \leftarrow \text{MPC}(X_{curr}, X_{goal}, r_{\Sigma_{k:k+N}})$

  $X_{ref}, U_{ref} \leftarrow \text{CubicSpline}(X, U_{sols})$

  $[pixel_x, pixel_y]_{1:f} \leftarrow \text{FeatureExtractor(RGB)}$

  $bboxes \leftarrow \text{CNN(RGB)}$

  $[x, y, z]_{1:l} \leftarrow \text{ObjectDetector(RGB-D}, [pixel_x, pixel_y]_{1:f}, bboxes)$

  $X_{curr} \leftarrow \text{RobotEstimator}(U, \text{IMU}, \text{joint encoders})$

  $r_{\Sigma_{k:k+N}} \leftarrow \text{RNN}(X_{sols}, [x, y, z]_{1:l})$

  `// Tracking Phase (follow` X_{ref} `and` U_{ref}`)`

  while $dt \leq dt_{plan}$ **do**

    $U \leftarrow \text{MPC}(X_{curr}, X_{ref(dt)}, U_{ref(dt)})$

    $X_{curr} \leftarrow \text{MotionTrackingController}(U)$

    $dt \mathrel{+}= dt_{track}$

  end

  $dt = 0$

end

2.1.2 General MPC Formulation

2.1.2.1 Planning Phase

MPC in the planning phase has the following time-invariant linear discretized model:

$$f(X_k, U_k) = X_{k+1} = AX_k + BU_k + w_k \tag{2.1}$$

where $X = \begin{bmatrix} x, & y \end{bmatrix}^\top$ represents our state variables (planar waypoint position), and $U =$

$\left[\ v_x, v_y\ \right]^\top$ represents our control variables (planar velocity). We also initialized our state and control variables to zero before run-time.

Because we have a motion tracking controller to incorporate robot dynamics (see Section 2.2.1), our A and B matrices can assume a simple point mass:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ B = \begin{bmatrix} dt_{plan} & 0 \\ 0 & dt_{plan} \end{bmatrix}$$

where dt_{plan} is the time between taking proprioceptive and exteroceptive sensor measurements (e.g., RGB-D images and odometer readings), and w_k represents a non-unit variance random Gaussian noise ($w_k \sim \mathcal{N}(0, \sigma^2)$, where σ represents the standard deviation of planar velocity).

The goal of our cost function in the planning phase (equation (2.2)) is to find the optimal control value that minimizes the distance from the current and predicted states ($X_{k=0\rightarrow N}$) to the goal state (X_{goal}) – where $X_{k=0}$ is given by the results of localization. Note, that we use $\hat{U_k}$ instead of U_k in our cost function to represent the inclusion of a slack decision variable, ϵ (the slack variable has no role in our discretized model equation, but does affect the cost function through R - see 2.1.3), so that $\hat{U} = \left[\ v_x, v_y, \epsilon\ \right]^\top$.

$$\min_{U_{k:k+N}} \sum_{k=0}^{N} \left(X_{k+1} - X^{goal}\right)^\top Q \left(X_{k+1} - X^{goal}\right) + \hat{U_k}^\top R \hat{U_k} \tag{2.2}$$

s.t. I, II, III, V (see Table 2.1)

2.1.2.2 Tracking Phase

MPC in the tracking phase has the following time-invariant linear discretized model:

$$f\left(X_k, U_k\right) = X_{k+1} = AX_k + BU_k \tag{2.3}$$

where $X = \left[\ x,\ y,\ \psi\ \right]^\top$ represents our state variables (desired planar position and yaw or heading angle), and $U = \left[\ v_x, v_y, \dot{\psi}\ \right]^\top$ represents our control variables (desired planar

velocity and yaw rate). Matrices A and B are the same as shown in (2.1), except for an additional row/column for yaw and yaw rate.

The goal of our cost function in the tracking phase (equation (2.4)) is to output desired planar velocity and yaw rate (v_d and $\dot{\psi}_d$) values that follow a reference trajectory.

$$\begin{aligned} \min_{U_{k:k+N}} \sum_{k=0}^{N} \left(X_k - X_k^{ref}\right)^\top Q \left(X_k - X_k^{ref}\right) \\ + \left(U_k - U_k^{ref}\right)^\top R \left(U_k - U_k^{ref}\right) \end{aligned} \tag{2.4}$$

$$\text{s.t. I, II, III, IV (see Table 2.1)}$$

X^{ref} and U^{ref} are obtained by cubic interpolation (equation (2.5)) with end points specified by the MPC planning phase from $X_k ... X_{k+2}$ and $U_k ... U_{k+2}$ (the reason we discretize to $k+2$ instead of $k+1$ is to ensure there are enough reference points for MPC to "look-ahead").

$$\begin{aligned} X^{ref}(t), U^{ref}(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3, \\ a_i = f\left(dt_{track}, X_k, U_k, X_{k+2}, U_{k+2}\right) \end{aligned} \tag{2.5}$$

2.1.3 MPC Constraints

2.1.3.1 Constraints I - IV

Constraint I represents multiple shooting constraints which facilitate solving non-linear programs [HHC15]. The limits on state variables (i.e., map constraints), and control variables (limits on velocity) are represented by Constraint II (note that the slack decision variable should be set as $0 \leq \epsilon$). If there is apparent jerk during path planning, it may be necessary to include Constraint III, where α^{limit} represents the limit on acceleration (a_x, a_y, and $\ddot{\psi}$) and U represents velocity (v_x, v_y and $\dot{\psi}$). Orienting the robot along the planned trajectory can be achieved using Constraint IV, and setting the limit on v_y to be much smaller than the limit on v_x (which points directly along the path) in the body frame. Because MPC outputs velocities in the inertial reference frame (irf), a rotation matrix is required to transform these velocities into the correct frame of reference.

2.1.3.2 Constraint V - Collision Boundary with Slack Variable

Our obstacle avoidance constraints are given by Constraint V, which ensure that the collision boundary of the robot does not collide with detected obstacles (note, that because these constraints are updated at every timestep dt_{plan}, moving obstacles can also be considered). x_{o_i} and y_{o_i} represent the x and y center positions of all obstacles detected by the robot ($i \to M$: where M is the number of obstacles currently in range). x_k and y_k represent the x and y positions of the robot from timestep k to timestep $k+N$ (future positions can be received from the MPC solution). r_{Σ_k} represents the radius of the collision boundary of the robot, and r_{o_i} represents the radius of the collision boundary of the obstacle. The collision boundary radius of the robot is calculated by using the major axis of the covariance uncertainty ellipse (Σ) estimated from RNN and is added to the radius or size of the robot itself (thus, we assume a more conservative collision boundary, which, combined with the slack variable—see below—provides some tolerance to ensure the planner does not fail while at the same time lowering the probability of collision). Note, from timestep k to timestep $k+N$, $\Sigma_{k \to N}$ is predicted by our RNN (see Section 2.1.5).

Without a slack variable and because of sensor measurement noise, the measured state of the robot may suddenly find itself very near or slightly inside the collision boundary of the obstacle and cause the solver to fail. To accommodate for this issue, Constraint V includes a slack variable to allow for some degree of constraint violation in our optimization problem. In other words, we effectively separate the covariance into a constraint and slack variable, where we tune the confidence attributed to the posterior estimate and break it into a nominal estimate, and a controllable slack parameter. Specifically, slack can be tuned through the R matrix, where a high cost for ϵ will ensure that the majority of solutions will not violate Constraint V, while lower values allow for greater violation (the cost on the slack variable is largely dependent on user-experience during implementation).

Table 2.1: Model Predictive Control Constraints

No.	Constraint
I	$X_{k+1} - f\left(X_k, U_k\right) = 0$
II	$X^{limit} \geq \lvert X_k \rvert,\ U^{limit} \geq \lvert U_k \rvert$
III	$\alpha^{limit} \geq \lvert\, [\mathrm{U_{k+1}} - \mathrm{U_k}]\, \rvert / \mathrm{dt}$
IV	$\begin{bmatrix} v_x^{limit} \\ v_y^{limit} \end{bmatrix}_{body} \geq \left\lvert \begin{bmatrix} \cos\theta_k & \sin\theta_k \\ -\sin\theta_k & \cos\theta_k \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}_{irf} \right\rvert$
V	$-\sqrt{\left(x_k - x_{o_i}\right)^2 + \left(y_k - y_{o_i}\right)^2} + r_{\Sigma_k} + r_{o_i} - \epsilon \leq 0$

2.1.4 Obstacle Detection

2.1.4.1 Convolutional Neural Networks

To support the avoidance of incoming obstacles as our robot traverses the environment, we use a custom-trained CNN model for real-time object detection. More specifically, using Redmon et al.'s YOLOv3 [RF18] fast CNN architecture because of its maturity (although other methods could be used, such as [FS18]), we trained two custom models. One localized brown boxes within an RGB camera frame using 1,500 hand-labeled images and the other localized black $1m$ x $1m$ boxes in a mostly empty Gazebo environment using 300 images. Weights were initialized using YOLOv3's default weights and trained for 5,200 epochs using stochastic gradient descent with a batch size of 64, momentum of 0.9, and learning rate of 0.001 for both models. We validated our model on labeled data withheld from the training data and we verified empirically that our object detector could successfully draw tight bounding boxes around our brown boxes (Fig. 2.2).

2.1.4.2 From Bounding Boxes to 3D Obstacles

For our end-to-end Gazebo simulation, we implemented simple classical feature detection over the simulated RGB and dense depth images to transform the bounding boxes from the object detector into useful 3D obstacles for the motion planner. The scheme described below assumes that features are cubes and that ALPHRED is directly facing all existing boxes. It is executed only once, at the beginning of the simulation. Note that instead of fully addressing the semantic mapping problem, we use simple placeholder computer vision components designed for our specific test scenarios; for now, we only utilize SLAM as training data for the RNN.

First, ORB features [RRK11] are extracted. Let x_p and y_p be the pixel coordinates of a single feature, and Z_c be its depth. Then, let $g_{sb} = (R_{sb}, T_{sb})$ be the body-to-spatial transformation, $g_{bc} = (R_{bc}, T_{bc})$ be the camera-to-body transformation, and K be the intrinsics matrix. Then, the position of the feature in the spatial frame, X_s (a 3x1 vector) can be

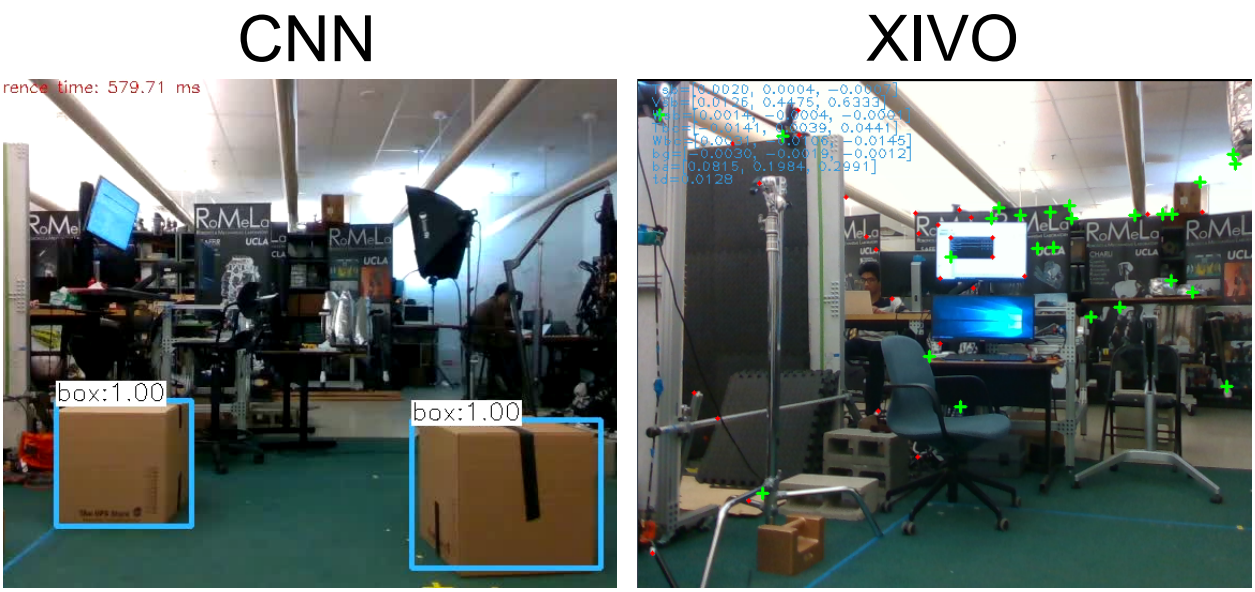


Figure 2.2: **Example Module Outputs.** *Left*: An example output image of our trained object detector using a custom-trained convolutional neural network model. We used the YOLOv3 [RF18] architecture with default initialized weights for fast training and inference. *Right*: Inlier (green +) and outlier tracks (red *) produced by XIVO on data collected from the Intel Realsense D435i.

calculated as:

$$\begin{aligned}
\begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix} &= K^{-1} \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} \\
X_c &= Z_c \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix} \\
X_b &= R_{bc} X_c + T_{bc} \\
X_s &= R_{sb} X_b + T_{sb}
\end{aligned} \tag{2.6}$$

Next, for each bounding box captured by the CNN object detection process, we determine which features are in each bounding box. The size of the box is the maximum distance (in meters) between any two points. Half of that size then becomes the "radius" of the obstacle's collision boundary (the MPC assumes that the collision boundary are circles).

We note that our classical feature detection approach is computationally efficient but also simple (i.e., not as robust). For example, most features exist near corners, where rounding errors could lead to a very different depth value. For the purpose of our end-to-end Gazebo simulation, we discarded any obstacle detections that were more than 5 meters away.

2.1.5 Recurrent Neural Networks for Learning Uncertainties

The RNN, shown in Figure 2.3, uses a combination of feedforward layers and simple RNN layers. The hidden layers all use ReLU activations. The network's 18 inputs are the robot's x, y, and z positions. The next 15 inputs consist of the x, y, and z positions of the five closest tracked features at any given state. The four output layer neurons correspond to the four values of the robot's 2×2 x-y covariance matrix, which is then used in Constraint V of the motion planning MPC. Unlike the hidden layers, the output layer uses a linear activation function because the outputs themselves are not restricted. Note that even though our MPC plans in only two dimensions, the inputs to the neural network are three-dimensional because the state estimation in our experiment is three-dimensional.

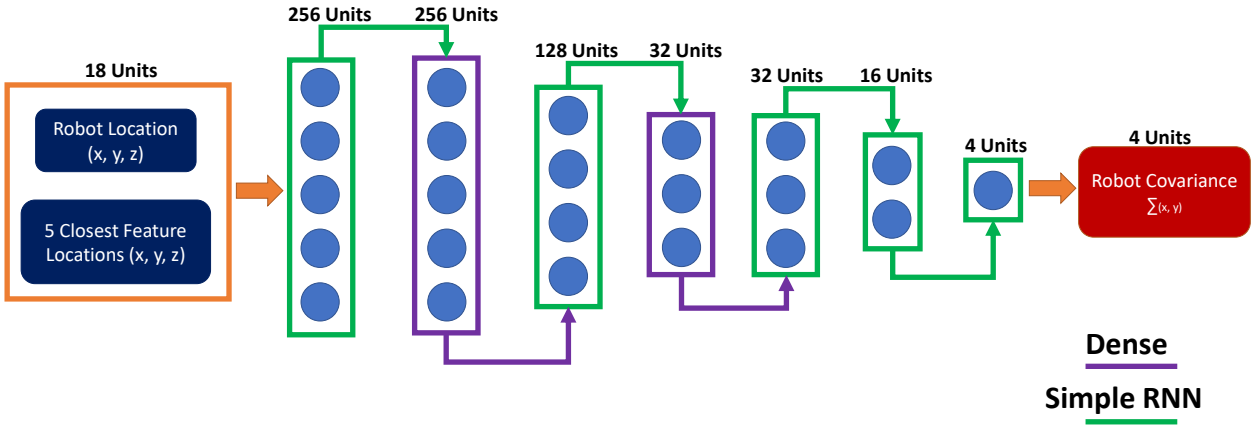


Figure 2.3: **Recurrent Neural Network Architecture.** Our RNN architecture predicts the covariances at robot poses $[x_{t+n}, y_{t+n}]$ at timesteps $t+n$ for $n = 1, ..., N$ (where N is the length of the MPC's prediction horizon). During training, we used inputs collected from the output of XIVO to parameterize the network towards the four output units, as indicated by the first 18 input units and last four units in the figure above. Seven hidden layers were used with ReLU activation functions, with five recurrent layers (green) and two fully connected layers (purple), to learn the temporal structure for covariance propagation.

We used the Mean Squared Error (MSE) as the loss function:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\Sigma_i - \hat{\Sigma_i})^2$$

Here N is the total number of timesteps, Σ_i is the covariance matrix of the planer position computed by a SLAM system at timestep i and $\hat{\Sigma_i}$ is the covariance matrix predicted by the RNN. Conceptually, the covariance matrix is a 2×2 matrix, but the implementation of the RNN treats it as a 4×1 flattened matrix when it makes predictions and propagates error. Lastly, we note that covariance matrices are positive semidefinite by definition. However, the above training procedure does not constrain the output of the RNN to positive semidefinite; the outputs were indeed arbitrary 2 x 2 matrices. To account for this, we zeroed out off-diagonal elements and negated any negative diagonal elements.

Training data (robot position, position of tracked features, and covariance matrices) for the RNN was collected from running XIVO,[3] a simplified and modernized implementation of the SLAM system described in [JS11], on time-synchronized RGB and IMU data collected from an Intel RealSense D435i mounted onto ALPHRED's head. We collected four ~40-second training sequences in total (using 100 epochs for training on the four sequences). The right side of Figure 2.2 displays tracked features and localization estimates from the collected data.

One key assumption that XIVO makes is that disturbances to the angular velocity and acceleration measurements (bias + noise) are a random walk (i.e. white, zero-mean, and Gaussian). This is not true for a walking robot, where each step produces a large periodic disturbance. Thus, XIVO's generic motion model is best suited to a flying robot. However, to adapt XIVO for our quadruped, we limited the acceleration and angular rate measurements to "realistic" values and then "de-tuned" the filter by setting large bounds on expected IMU measurement noise. This hampered accuracy, but ultimately enabled convergence.

Figure 2.4: **Gazebo Simulation.** Our high-fidelity simulation accurately models the dynamics of the ALPHRED quadruped robot.

2.2 Experimental Results

In this section, we provide an overview of our robot model and its motion tracking controller, describe the training and testing results of the CNN and RNN neural networks, and then summarize our end-to-end results using our Gazebo simulation environment.

2.2.1 Robot Model and Motion Tracking Controller

The robot used in this study is ALPHRED from Hooks *et al.* [HAY20], a full-sized quadruped robot that has unique kinematic configurations which enable several dynamic modes of operation as shown in Fig. 2.7 and Table 2.2. Our path planner is tested on a highly accurate simulation of ALPHRED using Gazebo software [KH] (Fig. 2.4). The robot is modeled as several interconnected rigid-bodies in PyBullet so that the state includes not only joint angular velocities, but sensor and actuator noise due to motor temperature. The camera model used is a standard perspective projection with the same intrinsics as the Intel RealSense camera used to collect RNN training data, but without distortions. ALPHRED uses an Extended Kalman Filter (EKF) that fuses kinematic encoder data with on-board IMU measurements to provide full state estimation [BHH13]. A Raibert-style controller [Rai86] is used to track desired trajectories, where the input to the controller is desired planar velocities (v_d) and a desired yaw rate ($\dot{\psi}_d$) in the body frame. The controller operates by planning footsteps using powerful heuristics based on velocity feedback and corrects velocity and orientation errors by adjusting the length of the limbs in support. Further details of the ALPHRED platform and its low-level motion tracking controller can be seen in [HAY20].

[3]Code available: `https://github.com/ucla-vision/xivo`

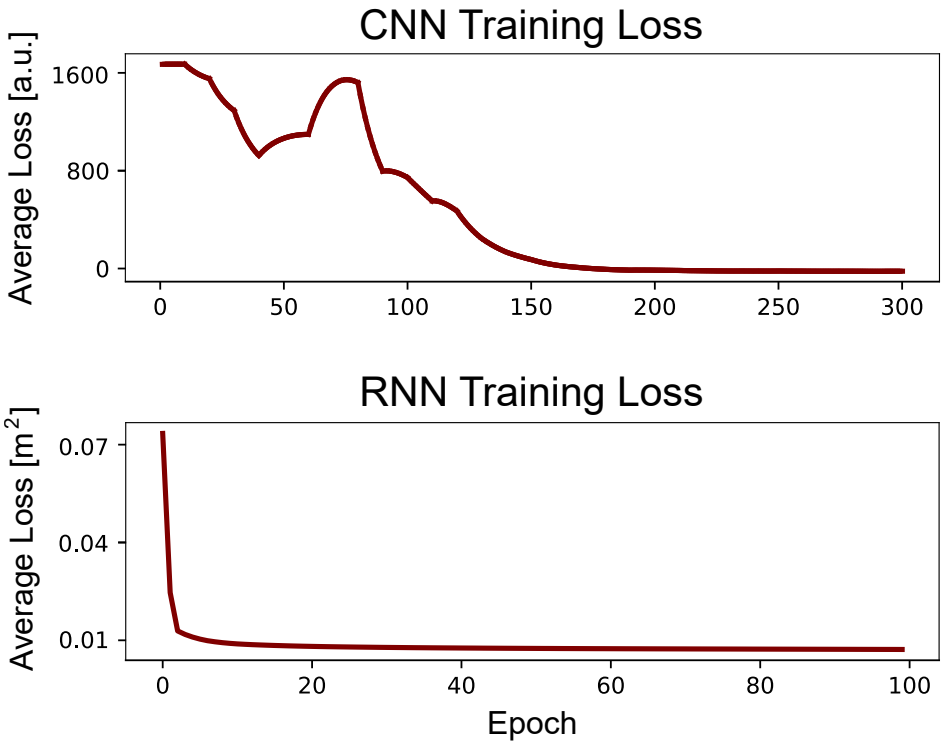


Figure 2.5: **Training Loss.** *Top:* Our CNN model's training loss, used in our object detection pipeline. We trained for 5,200 epochs but only display 300 in the figure above. Note that we verified avoidance of overfitting via a validaton set but did not plot the curve here. *Bottom:* Our RNN model's training loss, used to infer future localization uncertainty for the MPC. As with the CNN, we verified avoidance of overfitting using a validation set.

2.2.2 Analysis of Learning Components

Training loss for both the CNN and RNN are shown in Figure (Fig. 2.5). CNN and RNN networks were trained for 5,300 and 100 epochs, respectively, but only a limited range was plotted for visualization. To avoid overfitting, we used cross-validation and ensured that the validation loss was close to the training loss during the training process for both networks. Additionally, we observed that as ALPHRED tracked more features (i.e., the corners of an obstacle), the RNN's covariance estimates decreased. Conversely, as tracked features went out of view, estimates would increase. This is expected from the behavior of a visual-inertial odometry algorithm.

2.2.3 Gazebo Simulation

To test our proposed method, we used a custom Gazebo environment loaded with a high-fidelity model of our quadrupedal robot equipped with a Microsoft Kinect sensor. For localization, we used the motion tracking controller as described in Section 2.2.1. Our 3D environment consisted of a $1m^3$ box obstacle with the objective to command ALPHRED to move from its initial position at [0,0] to the goal position at [8,0]. We compared our method against a baseline approach, in which only an MPC was used for trajectory planning (with the obstacle explicitly hardcoded), and a naive approach for safer traversal, in which the robot's radius was artificially inflated to twice the original size (from 0.7m to 1.4m).

In the illustrative example shown in Fig. 2.6, we observed that when using a classic MPC controller, which assumes that the robot's state estimation is perfect, the resulting trajectory is too close to the obstacle and ALPHRED crashes (red). On the other hand, when using a conservative MPC controller, in which the assumed value of ALPHRED's radius is twice its actual size, the resulting trajectory over-avoids collisions and ALPHRED moves slowly towards the goal point (blue). However, when using our full risk-aware MPC in this scenario, we observed that ALPHRED not only avoids collision, but executes a tighter trajectory than the conservative approach and requires less time to move to the goal. Note that the simulation was run on a laptop with an Intel Core i7 6700 HQ CPU and a NVIDIA GeForce GTX 970M GPU in real time with $dt_{plan} = 0.1s$ and $dt_{track} = 0.005s$.

2.3 Discussion and Future Work

Collision-free path planning within unknown and unexplored environments requires the daunting integration of several components, such as sensor processing, control algorithms, and uncertainty resolution, into a fast and online end-to-end framework. To this end, we propose an architecture that unifies these modalities which attempts to address the fundamental problem of uncertainty in Active SLAM. By inferring the future positional uncertainty for an MPC using an RNN, we can substitute typical belief space planners with a more com-

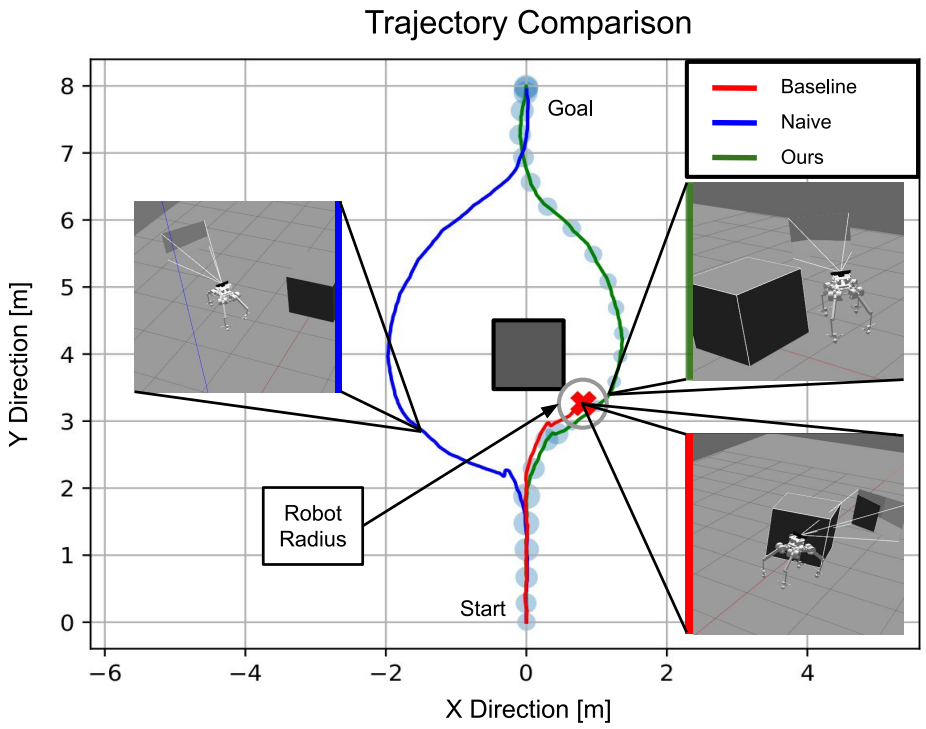


Figure 2.6: **Trajectory Comparison.** A comparison of the trajectories computed by three different approaches. The baseline method (red) is an MPC framework without our extensions to consider propagated future state uncertainty from an RNN, and we define the naive approach (blue) as artificially inflating a robot's boundary through all time. In comparison, our approach (green) can plan for a quick yet safe trajectory by predicting potential future collisions.

Table 2.2: ALPHRED Configuration

Parameter	**Value**
Degrees of Freedom	12 (3 per leg)
Weight	17.9 kg
Max Velocity	1.5 m/s
IMU	VectorNav 200
Camera	RealSense D435i

putationally efficient approach. Our work can also pave the way towards using RNNs to address problems with temporal structure which are difficult for classic robotic algorithms.

Overall, our architecture addresses Active SLAM by combining MPC, SLAM, RNN, and CNN algorithms. We demonstrate that by inferring future positional uncertainties of the robot using our RNN prediction model, the robot can reach a goal state faster than when assuming a fixed uncertainty while still safely avoiding obstacles. This is significant because modeling uncertainties within a neural network framework, rather than belief space planning (i.e., POMDP), sufficiently shortens the computation time, one of the major barriers to belief-space planning.

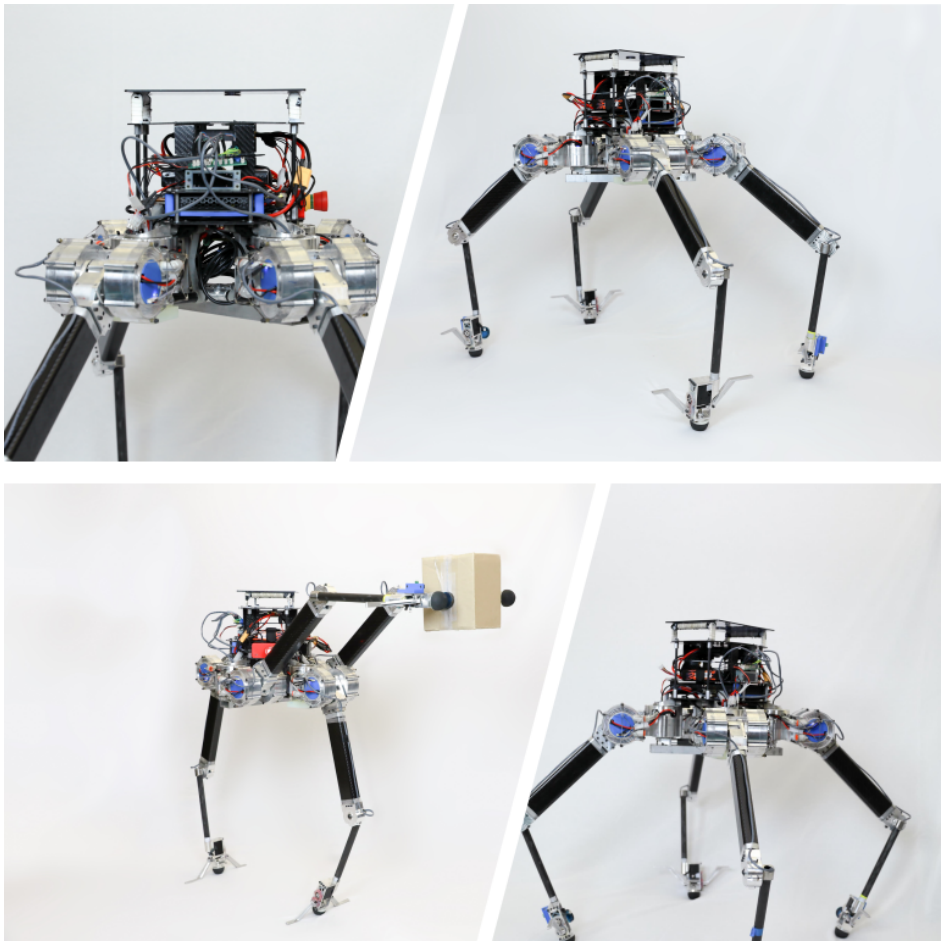


Figure 2.7: **ALPHRED Hardware.** The ALPHRED quadrupedal robot developed by Hooks *et al.* [HAY20] of the RoMeLa robotics laboratory at the University of California, Los Angeles. This complex platform is an ideal model to apply our methods, as showing success on this platform also demonstrates the potential of applying our methods to a wide selection of robotic systems. Table 2.2 describes some physical properties of the system.

CHAPTER 3

Learned Uncertainty Calibration for Visual Inertial Localization

We now examine Assumption 1 of Figure 1.1: that the covariance estimates of the SLAM algorithm within the framework are accurate. First, we create a statistical test similar to that used in [HMR09,FCD17] to evaluate the calibration of estimates from a state estimator, except without the need for Monte-Carlo experiments (Sect. 3.1). Using that statistical test, we show that ground-truth covariance can be computed assuming ergodicity when Monte-Carlo experiments are impractical (Section 3.1.4) and also count the number of timesteps that are within 1,2,3-σ bounds given by $\hat{P}$. Our approach for correcting inaccurate covariance estimates is most similar to [DL20] in that we use supervised machine learning to improve the covariance estimates, but the input to our models only consist of $\hat{P}$ and $\hat{x}$ instead of the entire input image; having only $\hat{P}$ and $\hat{x}$ as inputs allows us to use much smaller networks.

In Section 3.3, we test our method on two simple examples as a quick validation. Section 3.4 contains our main experiment, where we test our method on a VIO system processing real-world data. We achieved significant calibration using both a state-independent model $\phi(\hat{P})$ and a state-dependent model $\phi(\hat{x})$, which implies that the state $\hat{x}$ contains very little information about the miscalibration of $\hat{P}$ – most of the miscalibration of $\hat{P}$ can be computed from $\hat{P}$ itself.

3.1 Evaluating Calibration of Kalman Filters

3.1.1 Background: Sources of Error in EKFs

Consider a nonlinear discrete-time dynamical system and measurement model with state $x \in \mathbb{R}^n$ and measurement $y \in \mathbb{R}^m$:

$$\begin{aligned} x_k &= f(x_{k-1}, u_{k-1}) + \nu_k \\ y_k &= h(x_k) + w_k \end{aligned} \tag{3.1}$$

Assume that $\nu_k \sim \mathcal{N}(0, R)$ and $w_k \sim \mathcal{N}(0, Q)$ are white Gaussian noise processes, and the input u_k is known, along with the dynamics f and nominal measurement model h. An Extended Kalman Filter (EKF) recursively computes an estimate of x_k, $\hat{x}_k$, along with its covariance $\hat{P}_k = \mathbb{E}\left[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T\right]$ whenever it receives a new measurement y_k by computing the quantities:

$$\begin{aligned} \hat{x}_{k|k-1} &= f(\hat{x}_{k-1}, u_{k-1}) \\ \hat{P}_{k|k-1} &= A_k \hat{P}_{k-1} A_k^\top + R \\ K &= \hat{P}_{k|k-1} C_k^\top (C \hat{P}_{k|k-1} C^\top + Q)^{-1} \\ \hat{x}_k &= \hat{x}_{k|k-1} + K(y_k - h(\hat{x}_{k|k-1})) \\ \hat{P}_k &= (I - KC)\hat{P}_{k|k-1}(I - KC)^\top + KQK^\top. \end{aligned} \tag{3.2}$$

The matrices A_k and C_k are the Jacobians of $f(x, u)$ and $h(x)$ with respect to x evaluated at $\hat{x}_{k-1}$ and u_{k-1}. Estimates $\hat{x}_k$ and $\hat{P}_k$ represent a posterior Gaussian distribution. If $f(x, u)$ and $h(x)$ are both linear, then as k increases, $\hat{x}_k$ is guaranteed to converge to x_k and the computation of the $\hat{P}_k$ are completely separate from the computation of the $\hat{x}_k$. Moreover, the innovation $z_k = y_k - h(\hat{x}_{k|k-1})$ should be zero-mean and white in both components and time. These same guarantees do not apply when either $f(x, u)$ or $h(x)$ are nonlinear. In the nonlinear case, the mean and innovation are computed using the original nonlinear model f and the covariance is updated using linearized models. These unaccounted linearization errors mean that $\hat{P}_k$ is usually underestimated [BNG06]. Finally, most implementations of VO and VIO treat Q and R as constants, though they may be state and time-dependent.

3.1.2 1,2,3-σ Intervals in Multiple Dimensions

For a set of discrete samples ϕ_k, $k = 1, \ldots, N$ drawn from a 1-D Gaussian distribution $\mathcal{N}(\mu_k, \sigma_k)$, about 68% lie in the interval $\mu_k \pm \sigma_k$, 95% in $\mu_k \pm 2\sigma_k$, and 99.7% in $\mu_k \pm 3\sigma_k$. The same can be done for a set of points $v_k \in \mathbb{R}^d$, each from a potentially different multivariate Gaussian distribution $\mathcal{N}(u_k, \Sigma_k)$. First, diagonalize each Σ_k with eigenvalue decomposition: $\Sigma_k = X_k \Lambda_k X_k^\top$. Then, the columns of the matrix $X_k \Lambda_k^{1/2}$ form an orthogonal, but not orthonormal, basis and

$$\nu_k = (X_k \Lambda_k^{1/2})^{-1}(v_k - u_k) \tag{3.3}$$

contains the coordinates of v_k in the new coordinate system. Then, for each dimension $1, \ldots, d$ of ν_k, 68% of samples are in the interval $[-1, 1]$, 95% in $[-2, 2]$ and 99.7% in $[-3, 3]$. By counting the value of ν_k for each dimension at each timestep, we can evaluate the filter's calibration for each individual dimension, but not overall.

3.1.3 Overall Calibration with Monte-Carlo Simulations

Let $e_k = x_k - \hat{x}_k$ be the estimation error and $\hat{\rho}_k$ be the normalized estimation error squared (NEES), or square of the Mahalanobis distance at each timestep k:

$$\hat{\rho}_k = e_k^\top \hat{P}_k^{-1} e_k \tag{3.4}$$

$\hat{\rho}_k \sim \chi_n^2$, i.e. $\hat{\rho}_k$ is a χ^2 variable with n degrees of freedom. If we run M Monte-Carlo simulations and compute a value of $\rho_{k,i}$ for every timestep k in each run i, then their sum $\hat{\bar{\rho}}_k = \sum_{i=1}^{M} \hat{\rho}_{k,i}$ is a χ^2 variable with $M \times n$ degrees of freedom. Then, over the Monte-Carlo runs, we can compute confidence intervals for values of $\hat{\bar{\rho}}_k$. Values for $\hat{\bar{\rho}}_k$ should remain within the confidence interval for all k if the $\hat{P}_k$ are well-calibrated. If $\hat{\bar{\rho}}_k$ is consistently too high, then the covariance estimates $\hat{P}_k$ are too optimistic. If $\hat{\bar{\rho}}_k$ is consistently too low, then the $\hat{P}_k$ are conservative. This approach was used in [HMR09, FCD17] to measure the accuracy of their covariance estimates.

Unfortunately, Monte-Carlo approximations are not scalable, and not practical in a real-world scenario where one would have to carefully place the sensor platform in the same precise

position and orientation at every run to conduct repeated trials. Also, to perform repeated and sufficiently exciting motions required for VIO, one would need a precise actuation system, like a robot arm and not a quadcopter. Therefore, we need another approach to evaluate the covariance calibration.

3.1.4 Exploiting Residual Independence for Calibration

Here, we present a finer-grained method for evaluating calibration in a multiple dimensions with a goodness-of-fit test for unbiased estimators that does not require Monte-Carlo simulations. First, we assume that the e_k are approximately independent. While not strictly satisfied, this assumption enables a practical procedure, which we will then validate empirically. Next, let $p_{\hat{\rho}_k}$ be the approximate probability density function of $\hat{\rho}_k$, which can be computed with a normalized histogram of the $\hat{\rho}_k$. Next, since each $\hat{\rho}_k \sim \chi^2_n$ if the system is well-calibrated, we can then use $p_{\hat{\rho}_k}(x)$ in a goodness-of-fit test with the χ^2_n distribution. For this work, we use the L2 divergence [PXS11] between this approximate density and the density of χ^2_n as a comparison metric, $p_{\chi^2_n}(x)$:

$$\mathcal{D}_{L_2}(\hat{\rho}_k \| \chi^2_n) = \left(\int_0^\infty (p_{\hat{\rho}_k}(x) - p_{\chi^2_n}(x))^2 dx \right)^{1/2} \tag{3.5}$$

$\mathcal{D}_{L_2}$ is easy to compute and useful for comparing goodness-of-fit of multiple sets of $\hat{\rho}_k$ on the same dataset, but not as an absolute measure as one would use a p-value. We will use it to compare methods of calibration to ground truth covariance in the rest of the paper.

3.2 Computing a Calibrated Covariance

Our hypothesis is that the covariance estimates provided by an EKF present systematic errors and because of that, there exists a learnable map from the estimated value to a more calibrated value that can be executed in real-time. We consider maps of the following forms, in order of complexity:

1. A multiplicative scalar: All components of the estimated covariance are offset by a single scaling factor, $P_k = s\hat{P}_k$.

2. $P_k = A\hat{P}_k A^\top$, i.e. the map is a constant transformation of the covariance.

3. $Q_k = \phi(\hat{P}_k)$ and $P_k = Q_k Q_k^\top$, i.e. the map is an arbitrary function of the estimated covariance. This map, and the next, can be implemented by a feedforward neural network.

4. $Q_k = \phi(\hat{x}_k, \hat{P}_k)$ and $P_k = Q_k Q_k^\top$, i.e. the map is an arbitrary function of the estimated state and the estimated covariance.

An even more general model would be a map represented with a recurrent neural network. Our experiments show, however, that the memoryless maps of hypotheses 3 and 4 are sufficient for uncertainty calibration.

3.2.1 Finding Ground-Truth Covariance

Validating any of the hypotheses above requires a "ground-truth" value of covariance. In a Monte-Carlo experiment, we can use the unbiased sample covariance at a given timestep k given measurements $e_{k,i}$:

$$\tilde{P}_k = \frac{1}{M-1} \sum_{i=1}^{M} e_{k,i} e_{k,i}^\top \tag{3.6}$$

However, running many nearly identical tests on real-world equipment to measure ground-truth covariance is costly and time-consuming. For real-world experiments, we can compute a pseudo-ground-truth covariance with only one test if we additionally assume that that the motion state is approximately *ergodic*, i.e. that the population statistics match the temporal statistics. Practically, assuming ergodicity means assuming that errors in the motion state vary slowly over time, which is often true for converged filters. For an odd-sized time window K, we define pseudo-ground-truth as

$$\tilde{P}_k = \frac{1}{K-1} \sum_{k-\lfloor K/2 \rfloor}^{k+\lfloor K/2 \rfloor} e_k e_k^\top. \tag{3.7}$$

We can then use $\tilde{P}_k$ instead of $\hat{P}_k$ in (3.4) to compute a new set of $\tilde{\rho}_k$ and a new value for $\mathcal{D}_{L_2}$. For simplicity, we discard timesteps for which we cannot compute a sample covariance,

$k \leq \lfloor K/2 \rfloor$ and $k \geq N - \lfloor K/2 \rfloor$, where N is the total number of timesteps, from further analysis. Using these $\tilde{\rho}_k$, we can verify the ergodic assumption: if the ergodic assumption is true, then $\mathcal{D}_{L_2}(\tilde{\rho}_k \| \chi^2_n)$ should be small. In the experiments section, we visualize typical "small" and "large" values of $\mathcal{D}_{L_2}$.

The authors of [LOV18] and [DL20] trained neural networks to predict covariances by training them with negative log-likelihood (NLL) losses. Unlike the typical maximum likelihood problem in which a few parameters are estimated from many data, in their NLL loss every datum is potentially drawn from a different distribution. We believe that they were nevertheless able to train a neural network to predict covariances because their training data was approximately ergodic and therefore not drawn from many different distributions.

3.2.2 Hypothesis 1: Constant Multiplicative Scalar

We solve for a constant factor over all M sequences in a training dataset using the following optimization problem:

$$\begin{aligned} \underset{s}{\text{minimize}} \quad & \sum_{q=1}^{M} \sum_{k=0}^{N} \sum_{i=0}^{n} \sum_{j=i}^{n} (s \hat{P}^{i,j}_{q,k|k} - P^{i,j}_{q,k})^2 \\ \text{subject to} \quad & s \geq 0 \end{aligned} \tag{3.8}$$

where $P^{i,j}_{q,k}$ denotes the i,jth entry of the covariance matrix from the kth timestep of the qth sequence. This hypothesis is simplistic and not powerful enough, but it is an easily solvable quadratic program.

3.2.3 Hypothesis 2: Constant Linear Transformation

The second hypothesis is expressed as the optimization problem with decision variable $A \in \mathbb{R}^{n \times n}$

$$\underset{A}{\text{minimize}} \quad \sum_{q=1}^{M} \sum_{k=0}^{N} \sum_{i=0}^{n} \sum_{j=i}^{n} ((A \hat{P}_{q,k|k} A^\top - P_{q,k})^{i,j})^2 \tag{3.9}$$

This quartic and nonconvex optimization problem is the natural next step from Hypothesis 1. We implement Hypothesis 2 using IPOPT [WB06]. Theoretically, this adjustment should

be at least as effective as the constant multiplicative scalar, since $A = sI$, where s is the solution to (3.8), is a feasible solution to (3.9). However, because local nonconvex optimizers are sensitive to the initial guess and are not guaranteed to return the correct solution, the calibration of Hypothesis 2 is not uniformly better than that of Hypothesis 1. In Sections 3.3 and 3.4, it is always worse than Hypothesis 1.

3.2.4 Hypothesis 3 and 4: Fully-Connected Neural Networks

Let $Q \in \mathbb{R}^{n \times n}$ be the output of the neural network and let the adjusted covariance be $QQ^\top$. We use a weighted elementwise difference between the upper triangles of $QQ^\top$ and P as the loss function:

$$\mathcal{L} = \sum_{i=0}^{n} \sum_{j=i}^{n} w_{ij}((QQ^\top)^{i,j} - P^{i,j})^2 \tag{3.10}$$

Each timestep of each sequence serves as a training point. The inputs to the network are the upper triangle of a covariance matrix. In each of our experiments, we trained many simple feedforward neural networks using the Adam Optimizer in Tensorflow with varying architectures, L2 regularization weights, and epochs; very little thought was given to the architectures we tested. The outputs of these neural networks are well-calibrated and the best performing architecture for each experiment is detailed in Sections 3.3 and 3.4.

3.3 Two Contrasting Illustrations

Before presenting our main experiment in Section 3.4, we illustrate concepts from the previous two sections with a couple of easy-to-visualize examples.

3.3.1 Illustration 1: Linear Kalman Filter

A system that should have perfectly calibrated state estimates is a spring-mass damper system with mass $m = 1$, spring constant $k = 4$, and damping coefficient $c = 0.1$. After discretizing time into timesteps of length $\delta t = 0.01$s, the discrete-time dynamics and

measurement equations are:

$$\begin{aligned}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{k+1} &= \begin{bmatrix} 1 & \delta t \\ -\frac{k}{m}\delta t & 1 - \frac{c}{m}\delta t \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k + \begin{bmatrix} 0 \\ \delta t \end{bmatrix} u_k + \nu_k \\ y_k &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k + w_k \end{aligned} \tag{3.11}$$

where x_1 is the horizontal position, x_2 is the horizontal velocity, and u is a forcing input. In our experiment, $u(t)$ is the discretized version of $\sin(\frac{\pi}{2}t)$. In both the simulation and the Kalman Filter, $\nu_k \sim \mathcal{N}(0, 0.003^2)$ and each element of $w_k \sim \mathcal{N}(0, 0.005^2)$. Results for a single run from a set of 50 Monte-Carlo trials are shown in Figure 3.1. The overlay of the χ^2_2 density and the normalized histogram of the $\hat{\rho}_k$ are a near-perfect fit. Additionally, using the transformation in (3.3), we find that in the first dimension 68.20, 95.42, and 99.74% of points are within $\pm 1, 2, 3$, respectively. In the second dimension, the percentages are 68.12, 95.45, and 99.73%.

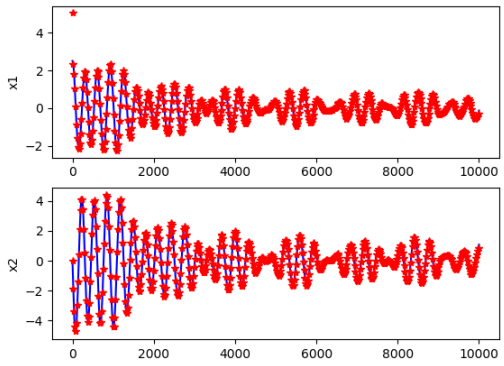
3.3.2 Illustration 2: EKF for 2D Localization

We repeat the same process above for an extended Kalman Filter for a 2D localization problem. The system is a Dubin's car with state $[x, y, v, \theta]$ and acceleration a and angular velocity ω inputs. It receives range and bearing measurements from a set of four known beacons located at $(3.5, -1.1), (10, 10), (-5, 15)$, and $(-10, -8.2)$. The discrete-time dynamics with discretization $\delta t = 0.1$s are:

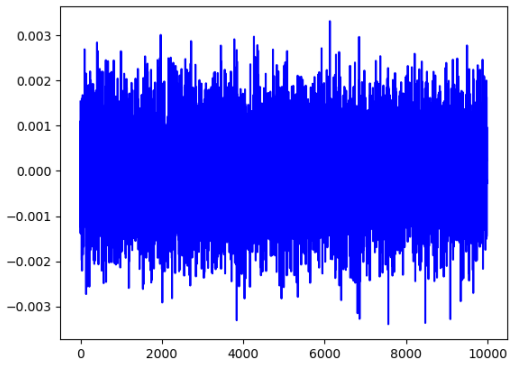
$$\begin{aligned} x_k &= x_{k-1} + \int_{t_{k-1}}^{t_k} v(\tau)\cos(\theta(\tau))d\tau \\ y_k &= y_{k-1} + \int_{t_{k-1}}^{t_k} v(\tau)\sin(\theta(\tau))d\tau \\ v_k &= a\delta t \\ \theta_k &= \omega\delta t \end{aligned} \tag{3.12}$$

and the measurement equations for beacon i located at $[x_i, y_i]$ are:

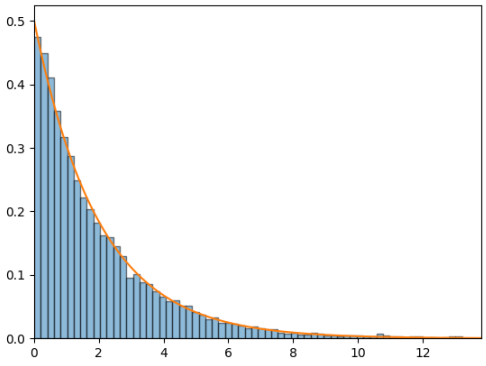
$$\begin{aligned} r_{i,k} &= ((x_i - x_k)^2 - (y_i - y_k)^2)^{1/2} \\ \phi_{i,k} &= \arctan(y_i - y_k, x_i - x_k) - \theta \end{aligned} \tag{3.13}$$



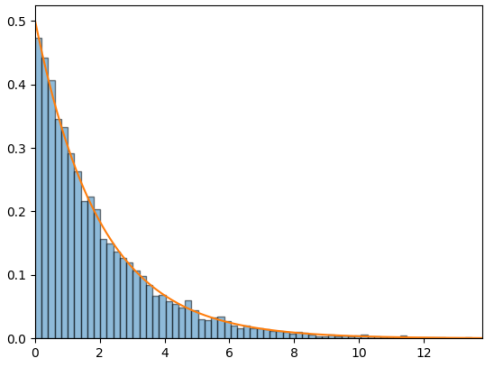
(a) State Errors



(b) Innovation



(c) χ^2_2 Overlay with $p_{\hat{\rho}_k}$



(d) χ^2_2 Overlay with p_{ρ_k}

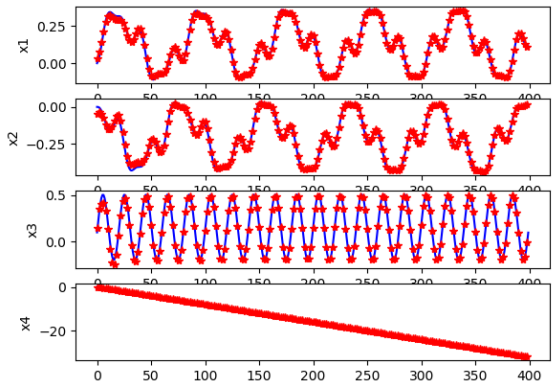
Figure 3.1: The state estimation and innovation of the linear Kalman Filter for a single run are shown in (a) and (b) - the state estimation is accurate and the innovation is essentially white noise. (c) plots $p_{\bar{\rho}_k}$ against the χ^2_2 density for a single run. Visually, the histogram and the χ^2_2 density are very close, showing that the independence assumption holds and that the covariance estimates are well-calibrated. This is further verified in (d), which is the same plot as (c), except that the normalized histogram is computed using a ground-truth covariance from Monte-Carlo trials. (3.6).

In these experiments, we generate 11 training sequences and one test sequence of a_k and ω_k. In all 12 sequences a_k is a sinusoid with frequency 0.5Hz and ω_k is a constant. Figure 3.2 is the corresponding figure to Figure 3.1 for this localization problem. It is clear that although the state estimation errors are small, the covariance estimates are inaccurate, and we test Hypotheses 1-4 on this problem. The neural network in Hypothesis 3 is a fully connected network with hidden layers that have 512, 512, 256, 256, 128, and 64 nodes. The L2 regularization weight was 1e-4 and it was trained for 50 epochs. The network for Hypothesis 4 is fully connected with five hidden layers that all have 128 nodes. Its L2 regularization weight was 1e-3 and was trained for 150 epochs. Both networks use ReLU activations, $w_{ij} = 5$ on the diagonals, and $w_{ij} = 1$ on the off-diagonals.

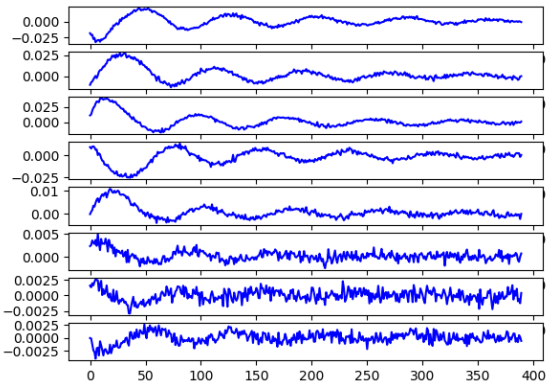
Divergences for the unadjusted covariances, ground-truth, and the four hypotheses are shown in Table 3.1 with corresponding overlays are in Figures 3.2 and 3.3. In Table 3.1, the third column contains the mean and standard deviation of the divergences of 50 Monte-Carlo runs. Ground-truth covariances are computed using (3.6) and the divergences in Table 3.1 are the mean and standard deviation of divergences of the 50 runs. The second column contains the reduction in divergence of the means in the second column as a percentage of the reduction from the unadjusted covariances to the ground-truth covariances. We observe that the calibration of the ground-truth covariances are within one standard deviations of the calibrations of both Hypothesis 3 and 4. The main conclusion of the results is that both Hypotheses 3 and 4 can achieve calibration, but that the state-dependence in Hypothesis 4 only yields a modest benefit. Note that in Figures 3.2 and 3.3, we did not plot the χ^2 density over the histogram when there was very little overlap between the two.

3.4 Calibration of Visual Inertial Odometry

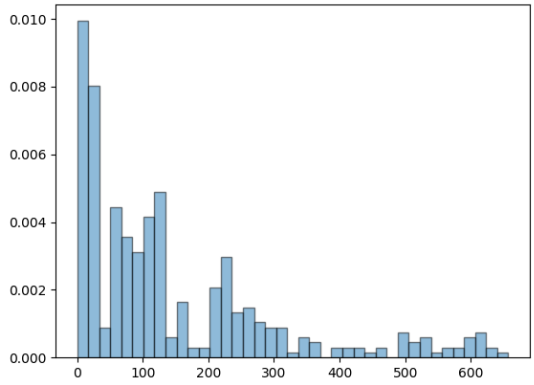
In the VIO problem, the state x consists of the orientation, position, velocity, the map states, and any autocalibration states. Since we often do not have ground truth for the map, alignment, and autocalibration states, we will only analyze the localization states. Orientation is represented as a rotation vector, so in this work, the state dimension is $n = 9$.



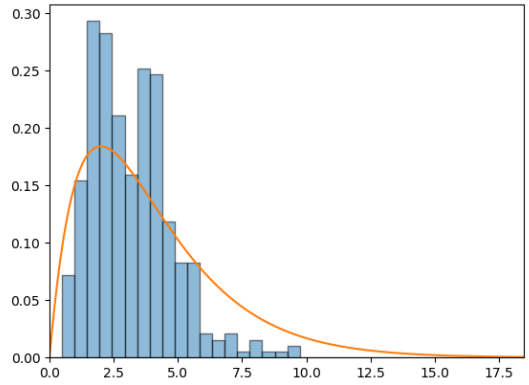
(a) State Errors



(b) Innovation

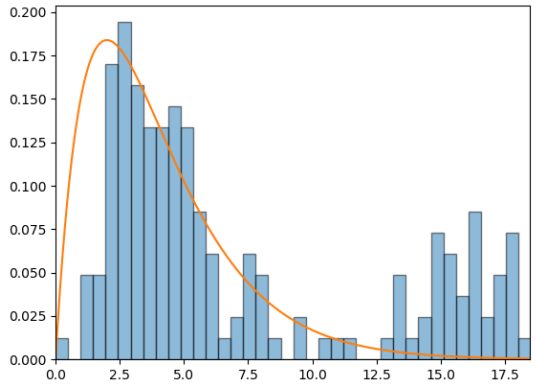


(c) χ^2_4 Overlay with $p_{\hat{\rho}_k}$

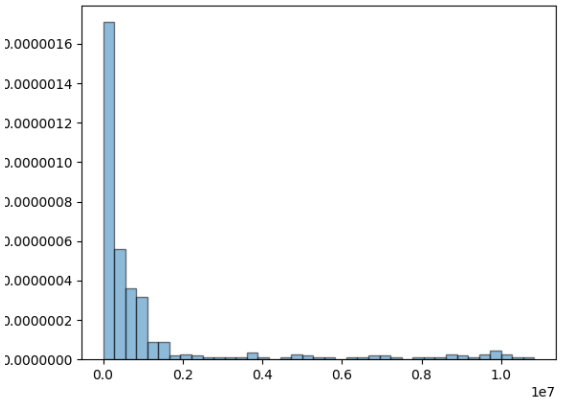


(d) χ^2_4 Overlay with p_{ρ_k}

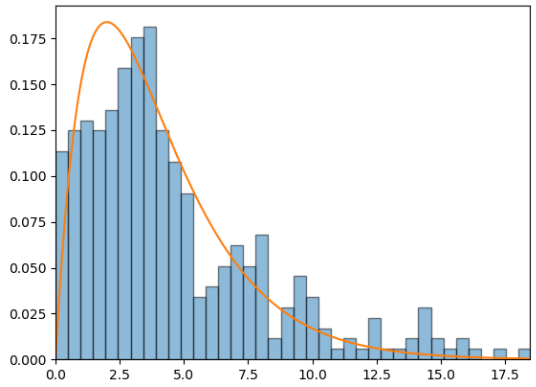
Figure 3.2: Calibration results for the EKF's test sequence. The EKF has small estimation error (a) and poor covariance calibration. The innovation for this 2D localization problem (b) is clearly not white. In (c), the approximate density of ρ_k is far from the χ^2_4 density that we did not plot the χ^2_4 pdf. Finally, in (d), the overlay is much closer to the ground-truth covariance computed using Monte-Carlo simulations, although still not a perfect fit because independence of the e_k is only an approximation.



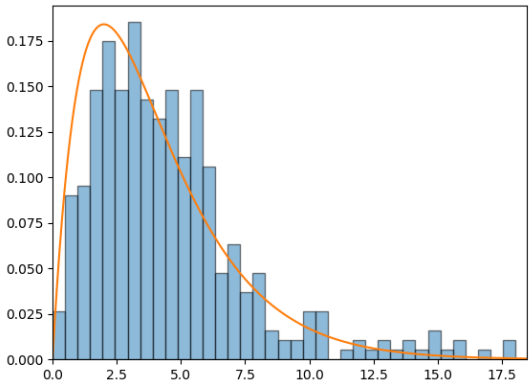
(a) Scalar Adj. Overlay



(b) Matrix Adj. Overlay



(c) NN Adj. Overlay

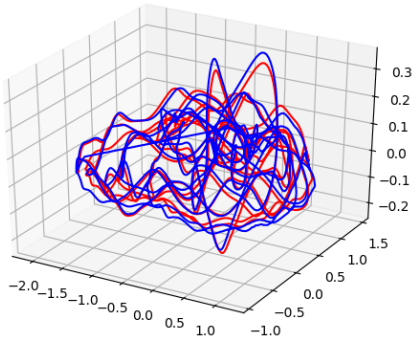


(d) NN w/State Overlay

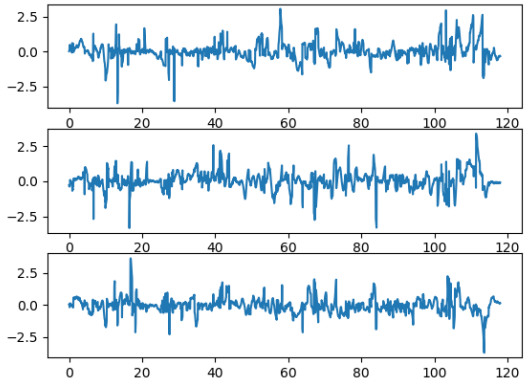
Figure 3.3: Overlays of χ^2_4 with $p_{\hat{\rho}_k}$ computed with adjusted covariances for the 2D localization problem. These overlays visualize the trends seen in Table 3.1.

Table 3.1: Calculated divergences for the 2D localization problem.

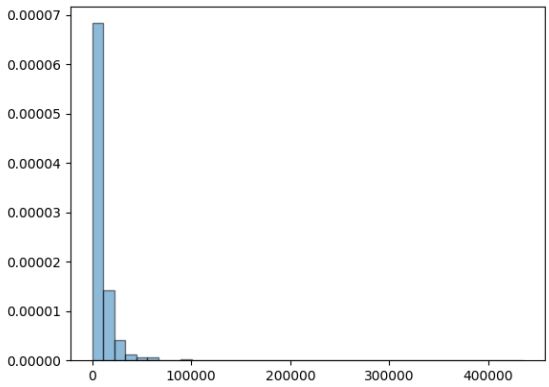
	% Dec.	**Test Set Sampling**	**% 1-σ**	**% 2-σ**	**% 3-σ**
Original Estimated	0%	0.3394 ± 0.0145	16.2, 17.6, 8.9, 58.5	30.0, 34.1, 17.1, 86.8	42.5, 48.3, 27.3, 94.8
MC Ground-Truth	100%	0.1839 ± 0.0547	60.8, 67.5, 68.5, 68.5	98.7, 96.1, 95.4, 95.9	99.7, 99.9, 99.5, 99.7
Global Scalar	31.6%	0.2902 ± 0.0510	30.2, 34.3, 17.3, 87.0	53.3, 60.3, 39.2, 96.7	69.3, 75.9, 60.8, 97.5
Global Matrix	-9.1%	0.3535 ± 2.4e-06	5.6, 2.1, 0.7, 0.4	11.2, 4.1, 1.0, 0.5	16.2, 6.1, 1.4, 0.5
Neural Network	75.4%	0.2222 ± 0.0884	89.6, 62.0, 64.1, 45.6	98.0, 88.3, 88.0, 75.2	99.8, 96.3, 94.4, 89.0
State-Dependent NN	90.9%	0.1981 ± 0.0553	95.7, 56.4, 51.1, 64.8	100.0, 95.3, 80.7, 91.4	100.0, 99.8, 92.7, 98.0



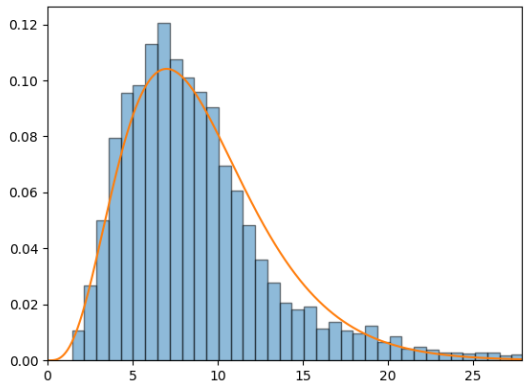
(a) 3D Trajectory



(b) Innovation



(c) χ^2_9 Overlay



(d) Ground-Truth Overlay

Figure 3.4: The 3D trajectory, innovation, and overlays for the VIO test sequence. As with the EKF experiments, the state estimation error is small, but the innovation is clearly not white noise. In (c), there is very little overlap between the histogram approximation of $\hat{\rho}_k$ the χ^2_9 distribution. (d) contains the same plot, except with $\hat{\rho}_k$ generated using ground-truth covariances computed using ergodicity from the test set. A visual comparison of (c) and (d) shows that the ergodic assumption and the independence assumption are both approximately true.

Table 3.2: Table of computed divergences for the VIO experiment.

	% Dec.	**Test Set Sampling**	**% 1-σ**	**% 2-σ**	**% 3-σ**
No Adjustment	0%	0.2697 ± 5.61e-08	11.4, 9.5, 10.2, 6.2, 5.1, 6.0, 3.3, 2.4, 2.1	21.8, 19.1, 20.5, 12.0, 9.6, 12.9, 6.2, 4.7, 4.7	32.6, 29.1, 30.4, 18.2, 14.5, 19.2, 9.2, 7.3, 7.0
"Ground-Truth"	100%	0.1103 ± 0.0223	60.4, 65.3, 67.4, 70.9, 71.7, 70.5, 67.1, 67.3, 70.3	98.2, 95.1, 96.1, 96.7, 96.9, 95.6, 95.8, 95.4, 95.1	99.9, 99.5, 99.5, 99.6, 99.7, 99.8, 99.7, 99.6, 99.4
Global Scalar	42.7%	0.1937 ± 0.0088	96.7, 93.4, 90.8, 74.7, 62.5, 79.7, 42.9, 30.1, 30.3	100.0, 97.8, 97.6, 95.5, 90.6, 96.8, 72.8, 50.3, 50.8	100.0, 98.6, 99.0, 98.8, 96.9, 99.0, 86.3, 65.1, 64.4
Global Matrix	16.1%	0.2433 ± 0.0037	98.1, 92.9, 78.1, 60.9, 66.5, 71.1, 60.0, 56.0, 74.2	99.7, 98.1, 83.0, 69.9, 77.0, 85.2, 71.1, 70.6, 90.3	99.9, 98.3, 83.9, 74.9, 82.8, 88.2, 75.3, 75.9, 94.9
Neural Network	97.8%	0.0987 ± 0.0159	76.4, 64.7, 72.8, 70.6, 70.6, 70.9, 69.8, 67.8, 65.7	96.5, 91.0, 93.3, 93.8, 94.0, 94.8, 93.2, 90.6, 90.1	99.6, 99.6, 98.5, 98.5, 98.8, 99.3, 98.4, 98.3, 98.1
NN with State	105.6%	0.1028 ± 0.0219	69.9, 65.4, 70.5, 70.2, 71.7, 71.3, 71.1, 70.6, 71.3	95.8, 91.2, 94.9, 95.0, 95.9, 95.6, 94.2, 94.2, 94.8	99.1, 99.2, 99.6, 99.3, 99.2, 99.4, 99.0, 99.2, 99.3

A detailed description and derivation of the equations of motion for a VIO system can be found in [JS11].

We evaluated XIVO, a reimplementation of the EKF SLAM system described in [JS11], on the TUM Visual Inertial Dataset [SGD18], a benchmark that features sequences of large, fast, and aggressive motions of a rig containing a stereo camera pair and an IMU. The dataset includes six sequences, named room1-room6, with "ground-truth" position and orientation collected using a motion capture system. Since XIVO's algorithm only uses monocular images, and not stereo images, we effectively have twelve sequences, totalling 32,470 timesteps. XIVO's estimate of the position and orientation is comparable with other state-of-the-art

VIO systems.[1]

Since the motion capture system did not measure velocity, we backdifferenced the position ground truth in order to compute a velocity ground truth as well. Next, we used the method of Horn [Hor87] to compute the transformation (a rotation and a translation) between the ground-truth points and the estimated points, since the two sets of points were not recorded in the same coordinate frame. The two room6 sequences were set aside as a test set while the other ten sequences were used for training. The trajectory of the test set, the innovation of the translation states, and overlay with the χ^2_9 distribution are shown in Figure 3.4.

3.4.1 Validating the Zero-Mean Assumption

The analysis in Section 3.1.3 assumes that the errors are zero-mean. We compute the errors for all timesteps of the twelve sequences in the TUM VI dataset and find the mean of all of them. After interpolation and alignment of the ground-truth data, the mean translation, rotation, and velocity errors are 5.18e-17m, 0.0064rad, and 0.0017m/s, respectively while the mean Euclidean norms of ground-truth translation, rotation, and velocity across all 12 sequences in the dataset are 1.15m, 1.50rad, and 0.902m/s. The translation error is zero because the method of Horn optimizes translation error when computing the alignment between the coordinate frames from the ground-truth measurements and the estimated states. Although technically nonzero, the rotation and velocity errors are small when compared to the motions in the dataset. Therefore, we will consider them negligible and consider the mean of the errors to be zero.

3.4.2 Validating the Ergodicity Assumption

The sample covariance in (3.7) is computed for each timestep k using a window of states centered around k. In order to find the best possible window size for each state of interest, we computed $\mathcal{D}_{L2}(\tilde{\rho}_k \| \chi^2)$ using odd-numbered window sizes between 27 and 601 for the ten

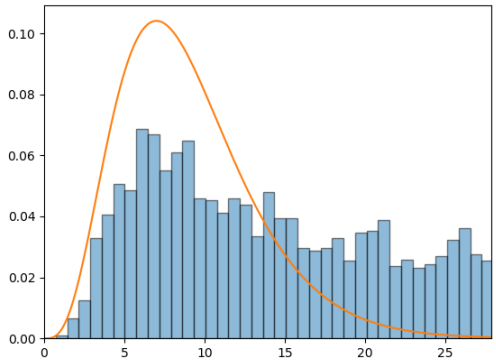
[1]See the table at `https://github.com/ucla-vision/xivo/blob/devel/wiki.md` for XIVO's absolute trajectory error (ATE) and relative pose error (RPE). Note that these errors are not the same as the *mean error* in Section 3.4.1.

training sequences. A window size of 275 produced low divergences for both the ten training sequences and the two test sequences. The divergence on the test set was 0.1105. The overlay with the χ^2_9 distribution is in Figure 3.4. Because these numbers are relatively small when compared to the divergences computed with the sampled covariances, and because of the relative visual fit compared to the overlay generated with the unadjusted covariance, we consider the ergodicity assumption validated.

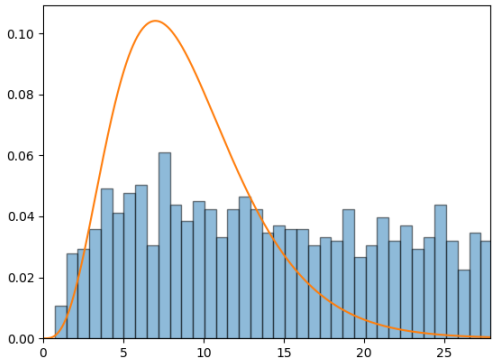
3.4.3 Experimental Results

We run our ten sequences of training data through Hypotheses 1-4. For Hypotheses 3 and 4, we use ReLU activations, a L2 regularization weight of 0.001 in the loss function, and set $w_{ij} = 10$ along the diagonals, 2.5 for off-diagonals in the same state's 3 x 3 block, and 0.5 otherwise. The neural network for Hypothesis 3 had hidden layers with widths 1024, 512, 256, 128, 64 and was trained for 25 epochs. The best network for Hypothesis 4 had hidden layers with widths 256, 256, 256, 128, 128 and was trained for 50 epochs.

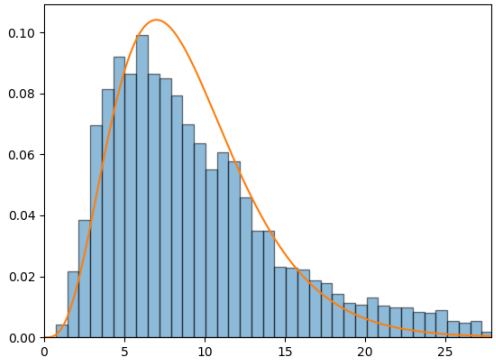
Divergences for the combined two test sequences are shown in Table 3.2 and visualized in Figure 3.5. The "% Dec." column in Table 3.2 displays the total decrease in divergence from the unadjusted covariances as a percentage of the reduction achieved using ergodic ground-truth. Numbers in the "Test Set Sampling" column are means and standard deviations of divergences computed from 50 groups of 200 points each from the test sequences. In the interest of space, the last three columns contain percentages for the position and orientations only. The trends are the same as those shown for the 2D localization experiment despite using ergodicity rather than Monte-Carlo simulations to compute ground-truth covariances: the calibration of the ground-truth covariance is within one standard deviation of the calibration of both neural networks and the calibrations of Hypotheses 1 and 2 are inadequate. Once again, when there was very little overlap between a histogram and the χ^2 density, we did not plot the χ^2 density.



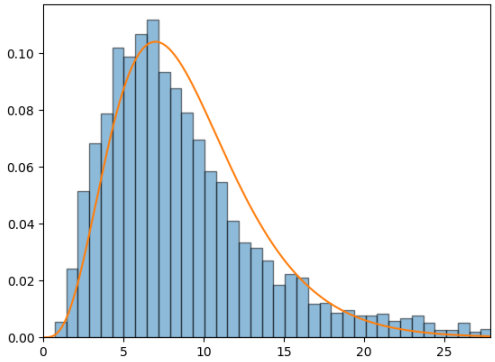
(a) Scalar Adjustment



(b) Matrix Adjustmnet



(c) Neural Network Adjustment



(d) State-Dependent NN Adjustment

Figure 3.5: Overlays of the test set's $\hat{\rho}_k$ computed with adjusted covariances - a visualization of the results in Table 3.2.

3.5 Summary

We have shown that there exists a learnable map between the uncalibrated estimates of a typical Extended Kalman Filter for VIO and the true, calibrated values. Another conclusion is that the ergodicity assumption is a reasonable way to compute a "ground-truth" value for covariance for suitable ground-truth motion when Monte-Carlo trials are not possible. Most interesting is the fact that the learnable map is almost entirely dependent on the covariance matrices alone for both XIVO and the smaller 2D navigation system.

CHAPTER 4

Feature Tracks are not Zero-Mean Gaussian

We now examine Assumption 2 of Figure 1.1, that feature track measurement errors are zero-mean Gaussian. That measurement errors, in general, are zero-mean Gaussian is an explicit assumption in the Kalman Filter and its nonlinear variants [TBF05, BB18] and implicitly built-into the optimization problem of bundle adjustment algorithms [MMT15b] and outlier-rejection algorithms [CGD09]. With extensive calibration with respect to temperature and mechanical alignment, the zero-mean Gaussian assumption is sufficient for the measurements of sensors such as inertial measurement units (IMUs) [Vec, TPM14], even if it is still not completely true: Extended Kalman Filters (EKFs) that rely on these IMUs are deployed on safety-critical systems actively in use.

Even though several well-known algorithms for Simultaneous Localization and Mapping (SLAM) rely on the often-deployed EKF (e.g. [JS11,GEL20,BBO17]), SLAM is still an active area of research. The existence of recently-released and actively used research benchmark datasets [ZHF22, WZW20] indicate that the robotics and computer vision communities still believe that performance of SLAM and an understanding of its failure cases are still insufficient, even after three decades of development [DB06]. This motivates an examination into the fundamental assumptions of SLAM.

This manuscript visits the assumption that feature tracks, the "measurements" of any indirect visual SLAM algorithm, contain only zero-mean Gaussian error. The covariance of the feature tracks is typically a tuning parameter to for all features at all times. We show that the feature track errors are not zero-mean Gaussian and furthermore, that the errors are conditional on the type of motion, the speed of motion, and the type of feature tracker used to extract the feature tracks. To our knowledge, this is the first study of the mean and

covariance of feature tracks *conditional* on the factors that affect them.

The organization of this chapter is as follows. Section 4.1 details the methods. Section 4.2 presents some key figures, and summarizes the error distribution of feature trackers. Section 4.3 ends with some concluding remarks. Additional figures from the experiment are given in Appendix.

4.1 Method

We wish to characterize the dependence of **mean error**, **mean absolute error**, **covariance**, **outlier ratio**, and **feature lifetime** on motion type, speed, tracker type, and when available, lighting. The types of motion investigated are:

- **Sideways motion** – Linear movement with no rotation.
- **Fixating motion** – Moving in a constant radius around a central object. The camera is always pointed directly at the central object, creating some rotation.
- **Forwards motion** – Driving-like motion. The primary change frame-to-frame is scale. Points near the center of an image will stay near the center in subsequent frames.

To vary speed, we skip frames at regular intervals from the image sequences. Nominal speed, or a speed of 1.00, means that all frames are used. A speed of 2.00 means that the feature tracker will only see every other frame, and a speed of 3.00 means that the feature tracker will only see one in every three frames. We do not test speeds below 1.00. The exact speeds tested depends on dataset. Finally, we also investigate the effect of two types of feature trackers:

- **Lucas-Kanade Sparse Optical Flow** [LK81]
- **Correspondence Tracker** using the SIFT descriptor [Low99]. Although computationally expensive, the SIFT descriptor was chosen because of its availability and its performance when used in state estimation tasks [SHS17b]. The descriptor of a feature track is set at the first frame it is detected and never updated.

We have chosen *not* to study lens distortion, since this would require multiple similar datasets collected with different cameras. All images in all datasets either have been preprocessed to remove lens distortions, or simulated without lens distortions. Since the Lucas-Kanade tracker is differential, we also choose not to study a differential correspondence tracker that updates the descriptor of a feature track at every frame.

4.1.1 Equations

Consider a feature i that was first detected at time t_0^i. If a depth image is available at time t_0^i and $g_{sc}(t_0^i)$ is known, we may fix the feature's position in the spatial frame, X_s^i:

$$\begin{aligned} X_c^i(t_0^i) &= \pi_K^{-1}(x_p(t_0^i), Z_0^i) \\ X_s^i &= g_{sc}(t_0^i) \circ X_c(t_0^i) \end{aligned} \tag{4.1}$$

In the above equation, $Z_c^i(t_0^i)$ is the third coordinate, or depth, of $X_c^i(t_0^i)$. Once, X_s^i is fixed, we can then calculate the **"ground-truth feature track"** $\bar{x}_p^i(t)$:

$$\bar{x}_p^i(t) = \pi_K(g_{sc}^{-1}(t) \circ X_s^i). \tag{4.2}$$

Some datasets provide a ground-truth point-cloud generated by a single lidar scan rather than a stream of depth images. A lidar scan is a point cloud with $M \sim 10^7$ points in the lidar frame L, which is defined as the camera frame at a particular time t_L: $\mathbf{P}_L = \{P_L^0, P_L^1, \ldots, P_L^M\}$. We can calculate the pixel coordinates of each point j in $\mathbf{P}_L$:

$$\pi_K(\mathbf{P}_L) = \{\pi_K(P_L^0), \pi_K(P_L^1), \ldots, \pi_K(P_L^M)\} \tag{4.3}$$

Feature tracks visible at time t_L can be associated with the nearest point in $\pi_K(\mathbf{P}_L)$. Suppose the nearest point in $\pi_K(\mathbf{P}_L)$ to feature i is P_L^j. Then, the ground-truth track of feature i is

$$\begin{aligned} X_s^i &= g_{sc}(t_L) \circ P_L^j \\ \bar{x}_p^i(t) &= \pi_K(g_{sc}^{-1}(t) \circ X_s^i). \end{aligned} \tag{4.4}$$

Once we have a ground-truth feature track for feature i, we can calculate the error signal for that feature:

$$e^i(t) = x_p^i(t) - \bar{x}_p^i(t) \tag{4.5}$$

where $x_p^i(t)$ is the observed track.

For datasets that provide a ground-truth point cloud at a single frame, the **mean error at timestep** t is

$$\mu(t) = \frac{1}{M(t)} \sum_{i=1}^{M(t)} e^i(t) \tag{4.6}$$

where $M(t)$ is the number of tracked features at time t. The **mean absolute error at timestep** t

$$\kappa(t) = \frac{1}{M(t)} \sum_{i=1}^{M(t)} |e^i(t)|. \tag{4.7}$$

Similarly, the **covariance at timestep** t is calculated by

$$\Sigma(t) = \frac{1}{M(t) - 1} \sum_{i=1}^{M(t)} e^i(t) e^i(t)^T. \tag{4.8}$$

It is only possible to compute $\mu(t)$, $\kappa(t)$, and $\Sigma(t)$ for features that are visible at time t_L, when the laser scan was acquired.

For datasets that provide a stream of depth images, we use different definitions of mean error, mean absolute error, and covariance. We can also use all features and not just those visible in a particular frame. The **mean error after** k **timesteps** is

$$\nu(k) = \frac{1}{\Psi(k)} \sum_{i=1}^{\Phi(k)} e^i(t_0^i + k\delta_t) \tag{4.9}$$

where $\Psi(k)$ is the number of features in the entire dataset tracked for at least k timesteps and δ_t is the length of each timestep. The **mean absolute error after** k **timesteps is:**

$$\eta(k) = \frac{1}{\Psi(k)} \sum_{i=1}^{\Psi(k)} |e^i(t_0^i + k\delta_t)| \tag{4.10}$$

where $\Phi(k)$ is the number of features tracked for at least k timesteps and δ_t is the length of each timestep. Finally, the **covariance after** k **timesteps** is given by

$$\Phi(k) = \frac{1}{\Psi(k) - 1} \sum_{i=1}^{\Psi(k)} e^i(t_0^i + k\delta_t) e^i(t_0^i + k\delta_t)^T. \tag{4.11}$$

When depth data is available at all frames, we define the feature's 3D location at the frame it is first detected and use equations (4.9), (4.10), (4.11).

At each frame, a feature tracker will attribute some features in one frame to the features in the previous frame. Let $F(t)$ be the total number of features in the frame at time t. The features in each frame will consist of $f_0(t)$ correct attributions, $f_1(t)$ incorrect attributions, and $f_2(t)$ new features, where $f_0(t) + f_1(t) + f_2(t) = F(t)$ and $f_0(t) + f_1(t) \leq F(t-1)$. Outlier rejection algorithms are used to determine $f_0(t)$ and $f_1(t)$ in real-time. The **outlier ratio** is defined as:

$$\frac{f_1(t)}{F(t-1)}. \tag{4.12}$$

Finally, the **feature lifetime** of a feature track is the total number of consecutive frames in which it found and successfully attributed. A feature is "born" at the frame it is first detected and "dies" if a feature is not found for a single frame.

4.2 Experiment Details

4.2.1 Feature Tracker Configuration

We used the feature tracker is the `Tracker` object integrated with XIVO, our in-house SLAM system. The tracker is configured to use the AGAST corner detector [MHB10], and to track between 1000 and 1200 features at a time. The AGAST corner detector was chosen for its speed and because it detects a large number of features in most scenes. The feature tracker was configured to track up to 1200 features per scene. We use RANSAC with $p = 0.995$ and an error threshold of 3 pixels to reject outliers. More details on the `Tracker` object and XIVO can be found in Appendix A.

Since the tracker software was programmed to be part of a larger system and not specifically for these experiments, the implementation of the Correspondence Tracker is not ideal. If a feature is visible in frames 0-5, but is not detected in frame 2, the tracker will drop the feature at frame 2 and initialize a new one in frame 3. This behavior is consistent with the definition of feature lifetime given in the previous section, but is not the ideal implementation for a Correspondence Tracker because there is always a possibility that a corner detector will not find the corner in one frame, or that a descriptor will be just a little too

different in one particular frame because of lighting. A more ideal implementation of the Correspondence Tracker would drop frames after a N_m missed frames, where $N_m > 1$ is an experimentally determined number. The definition of feature lifetime would also be changed to accommodate this more complex behavior. As a result of this choice, the distribution of feature lifetimes for the Correspondence Tracker are shorter than they otherwise would be. Furthermore, our experiments will fail to characterize trends that only appear at higher speeds.

4.2.2 Dataset-Specific Details

DTU Point Features Dataset. The DTU Point Features Dataset [ADS12] consists of sixty scenes of fixating motion. In the dataset, one or more objects is placed at the center of stage lit with up to 19 LEDs. A camera is mounted on a robot arm and moved in a precise manner at the stage. At each of 119 fixed locations, the camera acquires an image lit with one of the 19 LEDs, enabling lighting experiments using image-based relighting. The dataset contains a laser scan of the scene at a single frame, called the Key Frame. The original image size is 1600×1200. For speed, we use 800×600 px. grayscale versions of the images instead of the full resolution images.

We make use of the first 49 frames of each scene, or Arc 1 (see Figure 4.1). The Key Frame is Frame 25. We calculate mean error μ, mean absolute error κ, and covariance Σ using equations (4.6), (4.7), and (4.8). Since 3D data is only available at the Key Frame, calculation of errors and covariances only includes features that exist in Frame 25. Therefore, there is a bias towards longer tracks, as all short tracks that don't exist in Frame 25 are all tossed out. Since the "ground-truth" position of each feature in 3D is defined by its position in Frame 25, all results will therefore show that Frame 25 has zero covariance and the lowest errors. Statistics on feature lifetime and outlier rejection, however, do include features that do not exist in Frame 25.

To compute the ground-truth location of a feature track, we must associate a feature track to a point in a laser scan point cloud (eq. (4.4)). Since the point cloud does not cover

every pixel in the image, associations between features and laser scan points are only made if the pixel value of the laser scan point (eq. (4.3)) is less than 0.25 pixels from the feature. Associating a pixel to a laser scan point with the incorrect depth measurement will result in a very large calculated means in equation (4.6). Even with the low 0.25 pixel threshold, this bad association can still happen around edges and corners of objects. So that our analyses do not include very many of these poor depth associations, we throw out feature tracks whose maximum error is greater than the 90th percentile.

Since the DTU Point Features dataset was designed to enable image-based relighting, we also investigated the effects of directional light in addition to speed and the tracker used. We tested the same directional lights as [ADS12]. The position of each directional light is shown in Figure 4.1.

KITTI Vision Suite. The raw data [GLU12] in the KITTI Vision Suite consists RGB, GPS, IMU, and Lidar data captured from a moving vehicle. The motion captured in the images is predominantly forwards. The Lidar data was then processed into a separate benchmark dataset of depth images for single-image depth prediction and depth completion [USS17]. We make use stream `Image02`. Sequences containing "still frames" (e.g. significant amount of waiting at a traffic light), are excluded. Excluding sequences containing still frames leaves 28 scenes for our experiments. Although this is fewer scenes than the DTU dataset, it is still more frames because most sequences are longer than 49 frames.

Since 3D data is available at every frame, we define a feature's 3D position using the depth image from the very first frame where it was detected. Therefore, we use mean error ν (eq. (4.9)), absolute error η (eq. (4.10)), and covariance Φ (eq. (4.11)). To avoid errors due to bad depth measurements, we throw out the tracks whose maximum L2 error are above the 90th percentile and only calculate ν, η, and Φ at timesteps where there are at least 100 features (see Fig. B.56).

Simulated Supplementary Data. For sideways motions, we collected simulated RGB-D data in Gazebo. The simulation consisted of a Microsoft Kinect, modified so that RGB and

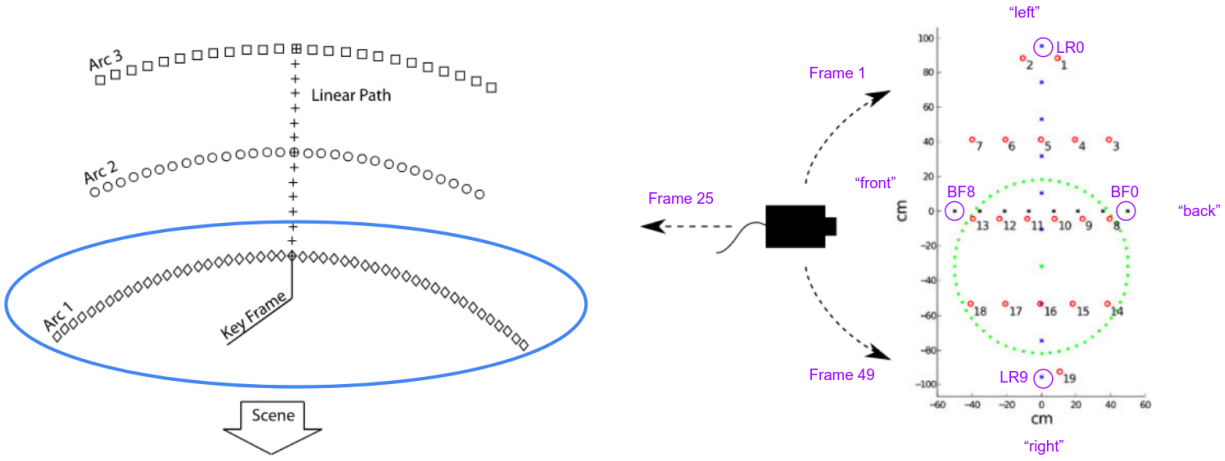


Figure 4.1: **An Illustration of the Light Stage Setup in the DTU Point Features Dataset. Left:** The locations at which images were acquired in the DTU Point Features dataset form three arcs and a linear path. Laser scans of the scenes were collected at the Key Frame (front and center). Frames from Arc 1 (circled in blue) are used for this experiment. **Right:** Red circles depict the location of 19 physical LEDs used to light the scene, which are spaced out over the scene. At each camera position in the left figure, the authors of the DTU Point Features dataset acquired 19 images. In each image, exactly one of the 19 LEDs is switched on. Acquiring 19 images in each location this way facilitates experiments in lighting using image-based relighting. Diffuse lighting can be simulated by using all 19 photographs from each position equally. More intense directional lighting can be simulated by weighting some LEDs more than others. In our experiments, we vary lighting from back-to-front (BF0-BF7) and left-to-right (LR0-LR9) as the camera follows the motion of Arc 1. Lights LR0 - LR9 and BF0 - BF7 are calculated by using Gaussian-weights on the 19 lights with $\sigma = 20$cm; Light LR6 is highlighted in green. Figures are reprinted and annotated with permission.

depth data would be co-located, mounted on a Hector quadrotor [MSK12] in ROS Melodic. The scene consisted of large objects from the Open Source Robotics Foundation's Gazebo Model Library. Images have a resolution of 800 $\times$ 600 pixels. In the subsequent sections, we refer to this dataset as "Gazebo Linear".In the Gazebo Linear dataset, we throw out tracks whose errors are above the 80-th percentile due to drift that naturally occurs when using the Lucas-Kanade Tracker in an environment containing straight and crisp edges parallel to the direction of motion. More details are given in Figure B.66.

4.2.3 Results

Mean error, mean absolute error, covariance, feature lifetime, and outlier ratio are all dependent on the type of motion, the tracker used, and the speed. For the DTU Point Features dataset, we found no dependence on the existence of directional light unless the directional light happened to cause tracking failure at high speeds. In Tables 4.1 - 4.3, we list the exact dependence of mean error, mean absolute error, feature lifetime, covariance, and outlier ratio on each independent variable. Differences in Tables 4.1 - 4.3 lead us to conclude that feature tracks are dependent on motion, tracker, and speed, but not the existence of directional light.

One notable difference between the Lucas-Kanade and Correspondence Trackers is that feature tracks produced by the Lucas-Kanade Tracker drift steadily while the Correspondence Tracker does not. This is because the Lucas-Kanade Tracker is differential, i.e. the characterization of a feature will slightly change frame to frame. For the Correspondence Tracker, this is not true. Therefore, the location of the feature track will drift, and the direction and magnitude of drift is dependent on the direction of motion. With left-to-right fixating motion, drift is positive (see Figure B.7). With left-to-right linear motion, drift is negative, and also larger (see Figure B.72). The flipside is that the Lucas-Kanade tracker generates features with a longer lifetime (see Figures B.2, B.55, B.67. When motion is fixating, the Correspondence Tracker also drifts about the direction of motion (see Figure B.21).

Finally, we note that the zero-mean Gaussian assumption holds when motion is predominantly forwards and we are using the Correspondence Tracker (see Figures B.63 and B.65). All figures supporting the assertions in this section are given in the Appendix.

	Tracker	**Lighting**	**Speed**
$\mu(t)$	No (fig. B.7)	No (figs. B.15, B.16)	No (figs. B.12, B.9)
$\kappa(t)$	Yes (fig. B.8)	No (fig. B.8)	Yes for Correspondence Tracker (fig. B.13), No for Lucas-Kanade Tracker (fig. B.10)
$\Sigma(t)$	Yes (fig. B.8)	No (fig. B.19, B.20)	Yes for Correspondence Tracker (fig. B.14), No for Lucas-Kanade Tracker (fig. B.11)
Feature Lifetime	Yes (fig. B.2)	No (fig. B.5)	Yes (fig. B.6)
Outlier Ratio	Yes (figs. B.4, B.3)	No (fig. B.4)	Yes (fig. B.3)

Table 4.1: **DTU Point Features Results Summary.** Cells contain whether or not the dependent variables in the left column are affected by the independent variables listed in the top row. Entries also contain figure numbers containing justification. The "Tracker" and "Lighting" columns contain references to figures containing plots at nominal speed. Although not indicated in the table, Figures B.26 - B.41 in the Appendix show that the existence of directional lighting continues to not affect outlier ratio, mean error, mean absolute error, and covariance at higher speeds for both the Lucas-Kanade and Correspondence Trackers.

4.3 Summary

Other than the caveat about the Correspondence Tracker noted in Section 4.2.1, the main limitation of this work is that there are more variables we could have tested, but chose

	Tracker	**Speed**
$\nu(t)$	No (fig. B.58)	Yes (figs. B.60, B.63)
$\eta(t)$	Yes (fig. B.59)	No for Correspondence Tracker (fig. B.64), Yes for Lucas-Kanade Tracker (figs. B.61)
$\Phi(t)$	Yes (fig. B.59)	No for Correspondence Tracker (fig. B.65), Yes for Lucas-Kanade Tracker (fig. B.62)
Feature Lifetime	Yes (fig. B.55)	Yes (fig. B.56)
Outlier Ratio	Yes (fig. B.57)	Yes (fig. B.57)

Table 4.2: **KITTI Results Summary.** Cells contain whether or not the dependent variables in the left column are affected by the independent variables listed in the top row. Entries also contain figure numbers containing justification.

	Tracker	**Speed**
$\nu(t)$	Yes (fig. B.70)	No for Correspondence Tracker (fig. B.75), Yes for Lucas-Kanade Tracker (fig. B.72)
$\eta(t)$	Yes (fig. B.71)	Yes (figs. B.73, B.76)
$\Phi(t)$	Yes (fig. B.71)	Yes (figs. B.77, B.74)
Feature Lifetime	Yes (fig. B.67)	Yes (fig. B.68)
Outlier Ratio	Yes (fig. B.69)	No for Correspondence Tracker, Yes for Lucas-Kanade Tracker (fig. B.69)

Table 4.3: **Gazebo Linear Results Summary.** Cells contain whether or not the dependent variables in the left column are affected by the independent variables listed in the top row. Entries also contain figure numbers containing justification.

not to. Examples of variables we chose not to test are the choice of feature detector and descriptor, and characteristics in the scene. For example, would the Correspondence Tracker have as little drift when moving forwards in an indoor environment and comparing BRIEF descriptors? Testing for conditionality on more variables inevitably leads to an unmanageable experiment, so we chose to lock in the feature detector and descriptor to well-performing available options and let the dataset dictate available scenes. Nevertheless, our work is a first step in characterizing the dependence of mean error, mean absolute error, covariance, feature lifetime, and outlier ratio on motion, tracker, speed, and the existence of directional lighting. The main conclusion is that the common zero-mean Gaussian assumption is rarely true.

CHAPTER 5

Quantifying VIO Uncertainty

This chapter examines Assumption 3 of Figure 1.1: that the estimated robot position is reliably correct. The previous chapter observed that the feature tracks commonly used in visual inertial odometry algorithms are affected by drift, noise, and attribution errors. This chapter follows up on the previous chapter and investigates the effect of drift, noise, and attribution errors on state estimation performance and uncertainty of XIVO in simulation.[1] Using a simulation rather than a benchmark dataset of real-world data allows us to quantify the individual effects of each rather than a mixture of all three. It also enables Monte-Carlo trials to calculate uncertainty without using the ergodicity assumption in Chapter 3. Since our previous work has shown that covariance matrices estimated by XIVO are not accurate, all mentions of "uncertainty" or "covariance" refers to sample covariances calculated using Monte-Carlo trials.

Most existing works on visual-inertial odometry benchmark performance on a real-world dataset of motion sequences consisting of IMU data and RGB images. When using current real-world datasets, there is little opportunity to benchmark uncertainty, as each motion sequence is only collected once.[2] Simulated cause-and-effect on our in-house monocular visual-inertial system allows us to study the effects of Gaussian noise, drift, and attribution errors individually on a general implementation. In real-world data, all these effects are tangled together.

Section 5.1 details the exact definitions and equations used in this chapter. Section 5.2

[1] Assumption 3 of Figure 1.1 is that the state estimate is "correct".

[2] If the ergodicity assumption holds, the methods in Chapter 3 can be used to compute uncertainty using real data.

gives details about the experiment. This chapter then ends with some concluding remarks.

5.1 Preliminaries

Our notation is consistent with [ML94] and past works describing the algorithm implemented in XIVO [JS11], [HTS15]. A description of the notation is given in Appendix A.

Let T denote the length of a trajectory in timesteps. Timestamps are denoted with the variable t. The state $\mathbf{x}(t) \in \mathbb{R}^9$ contains a rotation vector, translation vector, and velocity. Let $\hat{\mathbf{x}}(t)$ denote the estimated state and $\mathbf{e}(t) = \hat{\mathbf{x}}(t) - \mathbf{x}(t)$ denote the error state.

Our Monte-Carlo trials contain N runs each, indexed by n. $\mathbf{x}_n(t), \mathbf{e}_n(t)$ are the state and error state at time t in run n. The **sample covariance** at each timestep is given by

$$\Sigma(t) = \frac{1}{N-1} \sum_{n=1}^{N} \mathbf{e}_n(t) \mathbf{e}_n(t)^T. \tag{5.1}$$

The **mean sample covariance** over an entire trajectory is

$$\bar{\Sigma} = \frac{1}{T} \sum_{t=0}^{T} \Sigma(t) \tag{5.2}$$

Although not a part of standard metrics for evaluating SLAM systems [SEE12], errors in *scale* are the most sensitive to small perturbations in the input data. In other words, the error in translation and linear velocity are state-dependent and well-aligned with the ground-truth translation vector. This magnitude of these errors will vary widely depending on the exact imperfections in the input data. The **scale factor** ρ of an estimated trajectory is the mean value of the ratio of the norms of the estimated translation $\hat{T}_{sb}(t)$ and ground-truth translation $T_{sb}(t)$, after both trajectories are centered around their centroids:

$$\rho = \frac{1}{T} \sum_{t=0}^{T} \frac{\|\hat{\tilde{T}}_{sb}(t)\|}{\|\tilde{T}_{sb}(t)\|} \tag{5.3}$$

where

$$\begin{aligned} \tilde{T}_{sb}(t) &= T_{sb}(t) - \frac{1}{T} \sum_{\tau=0}^{T} T_{sb}(\tau) \\ \hat{\tilde{T}}_{sb}(t) &= \hat{T}_{sb}(t) - \frac{1}{T} \sum_{\tau=0}^{T} \hat{T}_{sb}(\tau). \end{aligned} \tag{5.4}$$

When $\rho < 1$, the estimated trajectory is "smaller" than the ground-truth trajectory. When $\rho > 1$, the estimated trajectory is "larger" than the ground-truth trajectory. To avoid division-by-zero errors when T_{sb} is close to the origin, we only include timesteps when $\|T_{sb}(t)\| > 0.1$ when computing ρ. For a set of N Monte-Carlo trials indexed by n, we can compute a set of scale factors ρ_n using equation (5.3) and plot their distribution.

5.1.1 On Observability and Identifiability for Monocular VIO

Observability is a property of a model that is assumed to contain a dynamic state, inputs, and outputs. Informally, a model is *observable* if given the inputs and outputs, the trajectory of the dynamic state can be exactly determined. A model is *unknown-input observable* if the state of the model can be exactly determined even if a subset of the inputs are not known. If the model contains calibration parameters to be estimated online, then the model is *identifiable* if the value of the calibration parameters and the trajectory of the dynamic state can be determined from the inputs and outputs. Parameter identifiability requires a *sufficiently exciting* input, i.e. if the input is not varied enough then it is possible that multiple values of the calibration parameters can satisfy the model dyanmics and output equations. What makes an input sufficiently exciting depends on the model.

For linear time-invariant systems, the definition of a sufficiently exciting input is well-known. The condition is called *persistent excitation*. Various equivalent definitions can be found in [AT65], [SLG76], and [WRM05].

The exact conditions for sufficient excitation of nonlinear systems is not well understood and is still an active area of study. An early result states that nonlinear systems are locally identifiable if their linearization about an equilibrium point is identifiable [GG76] — then the requirement for parameter identifiability is the same as for a linear time-invariant system, persistent excitation. Examples of recent papers on the topic are [PSA17, VT20, TM22]. In texts on systems identification and adaptive control, the requirements for sufficient excitation can be derived from LaSalle's invariance theorem and Barbalat's Lemma [LW12]. Texts on system identification and adaptive control typically focus on *linear-in-parameters* nonlinear

systems, or other assumed relationships between the dynamics and the parameters.

VIO models are not covered by the literature on systems identification and adaptive control. There are many previous works about the observability and identifiability of VIO models; each paper uses a slightly different model. The definition of sufficient excitation therefore varies from work to work. For example in [JS11], IMU measurements are outputs and the inputs to the model, linear jerk and angular acceleration, are modeled as noise independent of the state. IMU biases are Brownian motion. In [YGE19], the IMU measurements are modeled as inputs, rather than outputs, and the state contains an extra time-calibration parameter. IMU input noise is independent of the state. In [HTS15], IMU measurements are inputs to the model, IMU input noise is not independent of the state, there is no extra time-calibration parameter, and IMU biases are slowly-varying unknown inputs rather than noise. The model implemented in XIVO and used in these experiments is the one examined in [HTS15].

[HTS15] proves that the VIO model under examination is identifiable if and only if IMU bias rates are zero or exactly known. Otherwise, there exist multiple trajectories that can satisfy the output equations. The multiple trajectories are, however, a bounded set; the size of the bounded set is proportional to the IMU bias rates and inversely proportional to the *minimum excitation* of the angular velocity, angular acceleration, angular jerk, and linear jerk. The minimum excitation of a 3D signal $f(t)$ over a time interval $t \in \mathcal{I}$ is defined as:

$$m(f : \mathcal{I}) = \inf_{\|x\|=1} \left(\sup_{t \in \mathcal{I}} \|f(t) \times x\| \right). \tag{5.5}$$

In other words, the minimum excitation of $f(t)$ is determined by the (arbitrary) direction in 3D space with the smallest maximum value. If there is a direction that is not covered at all (e.g. no angular jerk in the z-direction), then the minimum excitation is zero, and the bounded set of possible trajectories in [HTS15] is actually unbounded. Sufficient excitation therefore means that angular velocity, angular acceleration, angular jerk, and linear jerk each cover 3-DOF. A randomly generated 3D trajectory will almost surely meet the requirement.

Of course, the model and the state estimation algorithm are different. A state estimation algorithm may fail to find the correct state of an observable model with a sufficiently exciting

input. Conversely, it is also possible for a state estimation algorithm to produce the original state even if the model is not observable or the input is not sufficiently exciting.

5.2 Experiment

We simulate IMU and visual inputs to our in-house VIO system, XIVO. Inputs are perturbed with Gaussian noise, drift, and random attribution errors so that we may measure the effect of each on Absolute Trajectory Error (ATE), Relative Pose Error (RPE), sample covariance $\Sigma(t)$ (5.1), and scale factor ρ (5.3).

5.2.1 The Trajectory Studied

Our experiment focuses on a randomly generated trajectory that is guaranteed to be sufficiently exciting and pictured in Figure 5.1. In the randomly generated trajectory, linear acceleration α^s_{sb} and angular velocity ω^s_{sb} in the spatial frame are modeled as Brownian motion with acceleration drift $\sigma_\alpha = 0.1$ m and angular velocity drift $\sigma_\omega = 0.001$ rad/s. A random amount of drift is added to α^s_{sb} and ω^s_{sb} with every IMU measurement, at 400Hz. α^s_{sb} and ω^s_{sb} are then transformed into the body-frame inputs, α^b_{sb} and ω^b_{sb}. Boundary conditions on linear velocity v_{sb} are set to $[-3, -1, -1]$ m/s and $[3, 1, 1]$ m/s. Boundary conditions on translation T_{sb} are set to $[-6, -3, -3]$ m and $[6, 3, 3]$ m. If the simulated ground-truth state in the spatial frame hits either boundary, the sign of that component of linear acceleration is flipped. Boundary conditions on all components of rotation vector w_{sb} are set to $-\pi$ and π. The total length of the trajectory is 127.76m over 80 seconds.

Although randomly generated Brownian motion curves, such as the trajectory in Figure 5.1, are guaranteed to be sufficiently exciting, the frequent and non-smooth changes in rotation and acceleration make convergence of an EKF and feature depth initialization (see Appendix A) difficult due to lack of parallax for triangulation.

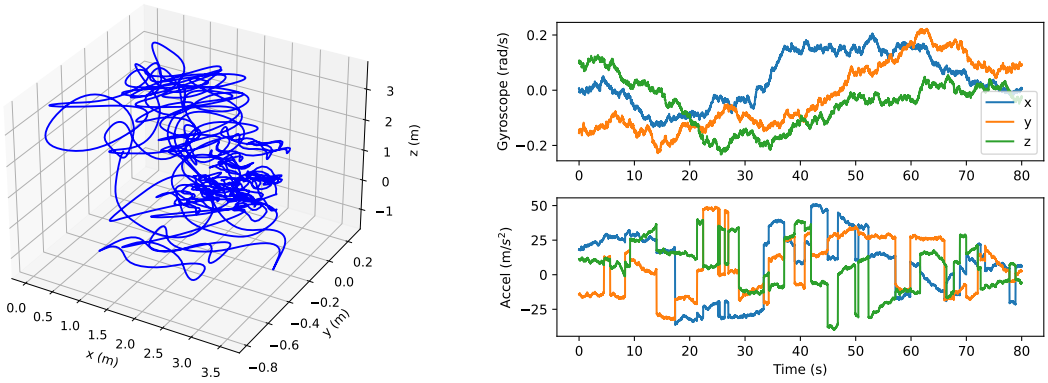


Figure 5.1: **The Brownian motion trajectory.** Linear acceleration and angular velocity are modeled as Brownian motion. Translation is plotted in the left figure in 3D. The linear acceleration and angular velocity inputs, in the body frame, are plotted in the right figure. Sudden jumps in the acceleration input correspond to instances when the trajectory hits a boundary condition in the spatial frame.

5.2.2 Configuration

The configuration of XIVO in Monte-Carlo experiments is given in Figure 5.2. More details about the configuration are given in the paragraphs below.

IMU Simulation. We simulate an IMU with similar specifications to the Intel RealSense d435i, a low-cost device. The accelerometer and gyroscope produce measurements at 400 Hz. IMU noise levels are held constant with accelerometer noise $\sigma_a = 1e^{-4}$ m/s$^2/\sqrt{\text{Hz}}$ and gyroscope noise $\sigma_g = 1e^{-5}$ radians/s$/\sqrt{\text{Hz}}$. IMU bias is initially zero, but drifts with parameters $\sigma_{b_a} = 3e^{-4}$ m/s$^2/\sqrt{\text{Hz}}$ and $\sigma_{b_g} = 5e^{-6}$ radians/s$/\sqrt{\text{Hz}}$.

Vision Simulation. So that we may have complete control over feature position errors and attribution, we skip the typical image processing step and instead directly feed pixel measurements of an attributed point cloud into XIVO at a rate of 25Hz. The point cloud consists of 1000 randomly generated points uniformly located in a box; all Monte-Carlo trials use the same set of generated features. At each vision timestep Visibility is calculated using

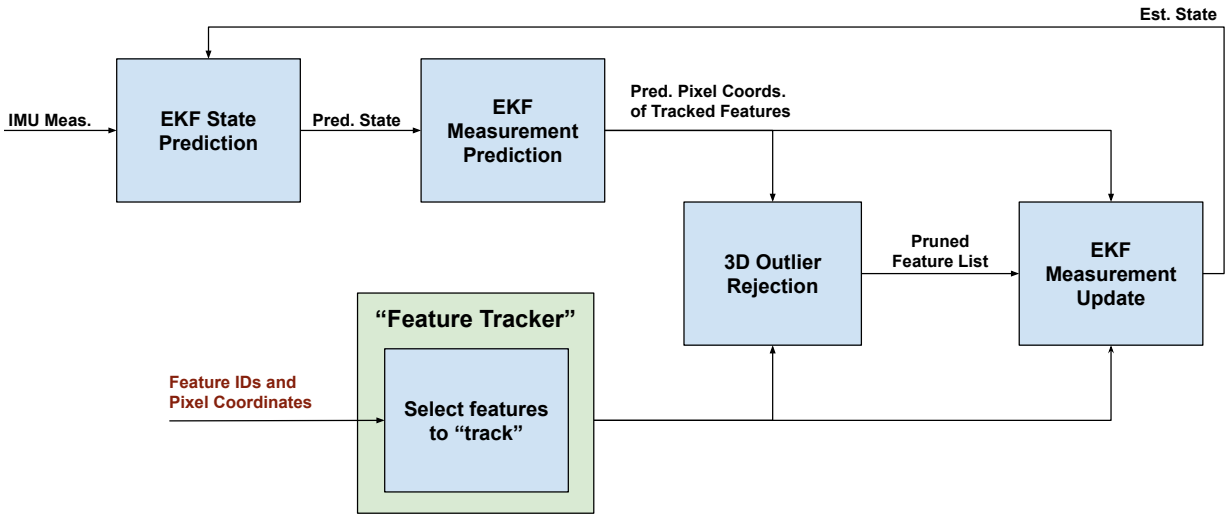


Figure 5.2: **XIVO's Configuration for Monte-Carlo Experiments.** For Monte-Carlo experiments, XIVO's typical feature tracker was replaced with simple bookkeeping software. Loop closure, an optional component, was not used. The typical configuration of XIVO is given in Figure A.1.

a camera with no distortion and the following intrinsics:

$$K = \begin{bmatrix} 275 & 0 & 320 \\ 0 & 275 & 240 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.6}$$

Observed features fall in and out of view during motion.

Extrinsics. Due to the development of specialized visual-inertial calibration algorithms and software, such as Kalibr, we assume that camera intrinsics, camera-IMU timestamp alignment, camera-IMU extrinsics, and accelerometer-gyroscope alignment are known. As in [JS11], we "remove" camera-IMU extrinsics from the state by setting the initial covariance of that portion of the state to a value $\sim 1e^{-10}$. All other calibration quantities are removed from the state through compile-time switches.

XIVO Configuration. A bare-bones "feature tracker" is configured to "track" and initialize depth estimates of 250 - 500 visible features. If there are more than 500 features, they

will be ignored. Feature depths are initialized with a combination of triangulation minimizing angular reprojection errors [LC19a] and subfiltering. There is no loop closure in these experiments. The EKF in XIVO uses 60 tracked features in its state. The features selected for state estimation are those with the most confident estimate of depth (see Appendix A for more details).

5.2.3 Experiment Parameters

Gaussian Noise. Let σ_p be the standard deviation of noise added to feature tracks and let $\bar{\sigma}_p$ be the standard deviation of noise used by the EKF when creating state estimates. We test values of $\sigma_p \in \{0.25, 0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 2.00, 2.25, 2.50\}$ pixels and set $\bar{\sigma}_p = \sigma_p$.

Drift. To simulate drift, we associate with each feature j a bias $b_p^j(t) \in \mathbb{R}^2$, with unit of pixels. When feature j is first detected, $b_p^j(t)$ is initialized to $[0, 0]^T$. With each frame, $b_p^j(t)$ evolves as $b_p^j(t+1) = b_p^j(t) + B$ where $B \sim \mathcal{N}(0, \sigma_b)$. At time t, the XIVO receives a simulated tracker measurement of $\pi(X_c(t)) + b_p^j(t)$, where $X_c^j(t) \in \mathbb{R}^3$ is the location of feature j in the camera frame at time t. We test $\sigma_b \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$ and two different values for the assumed Gaussian noise uncertainty in the EKF, $\bar{\sigma}_p = 0.50$ and $\bar{\sigma}_p = 1.00$. Over 25 frames (one simulated second), this corresponds to an average drift of zero pixels for all values of σ_b, with standard deviations of $\{0.005, 0.025, 0.05, 0.25, 0.5, 2.5\}$ pixels.

Attribution Errors. To create attribution errors, we swap the measurements of η percent of uniformly randomly selected visible features each frame before passing feature tracks to XIVO. The features assigned attribution errors are always those that are "tracked" by the bare-bones feature tracker, but may or may not be used in state estimation by the EKF. XIVO is configured to use Mahalanobis Gating to reject outlier measurements, such as those caused by misattributions. Since measurement errors caused by randomly swapping visible features are catastrophic, Mahalanobis Gating prevents the outlier measurements

from becoming part of the EKF's measurement update. Therefore, the real effect of our random misattributions is that the lifetime of feature tracks are prematurely cut short. We test with $\eta = 0, 1, 2.5, 5, 7.5, 10, 20, 30, 40$ percent.

5.2.4 Results

The outputs of our Monte-Carlo experiments are the box-and-whisker plots of distributions of ATE, RPE, mean sample covariance, and scale factor, shown in Figures 5.3 - 5.14. As expected, increasing Gaussian noise, drift, and attribution errors increase ATE, RPE, mean sample covariance, and scale errors. The degradation with respect to Gaussian noise and drift is graceful. However, the degradation with respect to attribution errors is exponential. Cutting the lifetime of feature tracks causes the largest performance errors, mean sample covariance, scale bias, and scale variance. Next, Gaussian noise causes larger performance errors, mean sample covariance, and scale uncertainty than drift. Drift, however, causes larger scale bias than Gaussian noise.

More particular details are noted in the paragraphs below. All plots for the same quantity (e.g. distribution of ρ) use the same vertical-axis scale for the Gaussian noise and drift experiments. The plots containing results of the attribution experiments require a different vertical-axis scale because of the exponential degradation.

Gaussian Noise. Results are shown in Figures 5.3, 5.4, 5.5. The general trend is that performance errors and state uncertainty, and variations of performance error and state uncertainty, increase with σ_p. The variation in ρ generally increases with σ_p, but mean and median of ρ hover around 2.1.

Next, we note that for performance error and scale errors for $\sigma_p = 0.25$ are larger than performance error and scale errors for $\sigma_p = 0.50$. We hypothesize that the combination of frequent changes in motion that make the dynamics less linear and poor feature initialization add measurement errors that are not adequately captured when we set $\bar{\sigma}_p = \sigma_p$. When we instead set $\bar{\sigma}_p$ a little bit higher, to $\bar{\sigma}_p = \sigma_p + 0.25$, performance error and scale errors when

$\sigma_p = 0.25$ become lower than when $\sigma_p = 0.50$. Results when $\bar{\sigma}_p = \sigma_p + 0.25$ are shown in Figures 5.6, 5.7, and 5.8.

Drift. Distributions of performance error, state covariance, and scale error in the drift experiment are shown in Figures 5.9, 5.10, 5.11. Performance error, state covariance, and scale errors increase monotonically with σ_b for both $\bar{\sigma}_p = 0.50$ and $\bar{\sigma}_p = 1.00$; their variations also widen as σ_b increases. Values of performance error, state covariance, and ρ are closer to their ideal values when $\bar{\sigma}_p = 0.50$ than when $\bar{\sigma}_p = 1.00$ for smaller quantities of drift. Once σ_b is high enough, values of performance error, state covariance, and ρ are closer to their ideal values with $\bar{\sigma}_p = 1.00$.

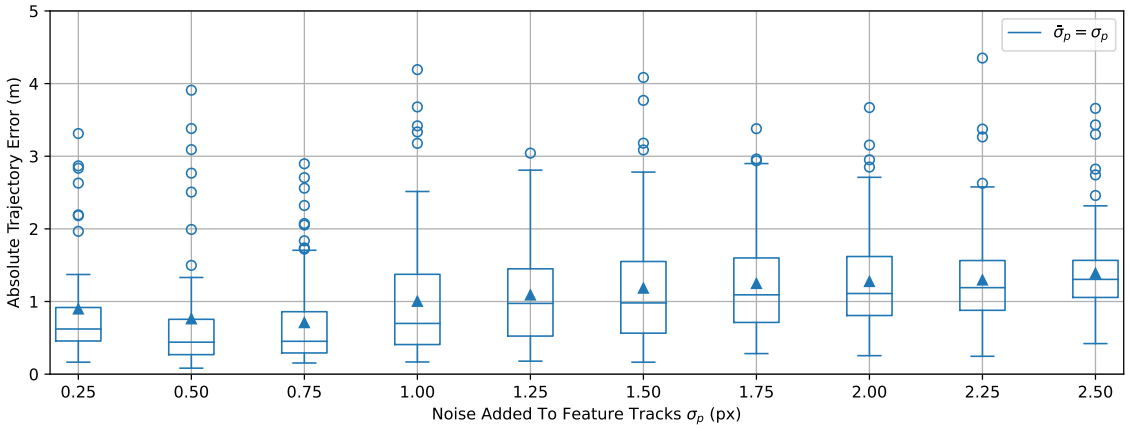
Attribution Errors. Distributions of performance error, state covariance, and scale error are shown in Figures 5.12, 5.13, 5.14. Results are simple: performance error, state covariance, and scale factor increase exponentially as η is increased. Variation in all three increase uniformly with η. Figures 5.12, 5.13, and 5.14 only display values of η up to $\eta = 0.1$, so that distributions of performance error, state covariance, and scale error at lower values are not dwarfed. Mean values of performance error, state covariance, and scale factor are shown for all tested values of η in Figure 5.15.

We note that $\eta = 0.1$ is a rather low percentage of outliers and that SLAM systems commonly encounter higher outlier ratios, as illustrated by the previous chapter. This indicates that outliers in real-world data are not random, like our simulated attribution errors, and that algorithms used to select features for state estimation are functioning as intended.

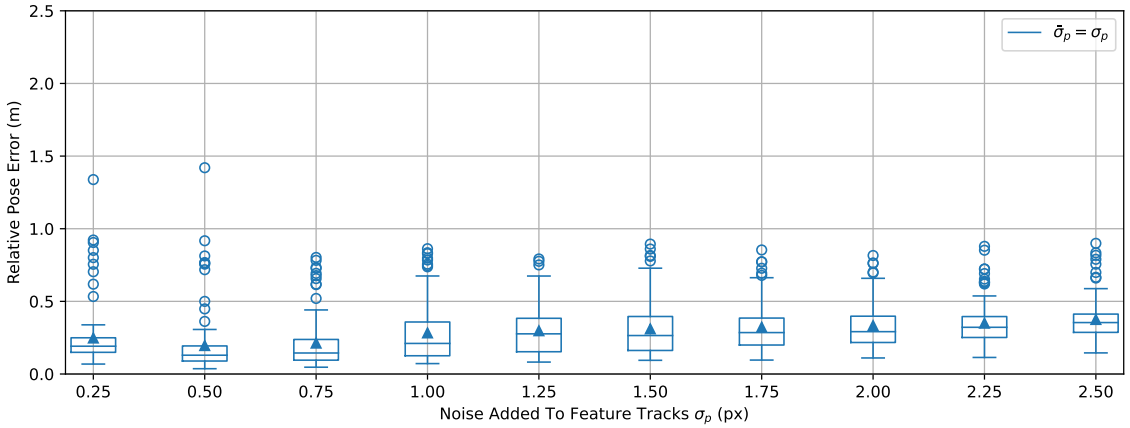
5.3 Summary and Discussion

There are at least two limitations to our results. The first is that we used theoretically correct values of $\bar{\sigma}_p$ in the Gaussian noise experiment rather than treating $\bar{\sigma}_p$ as a tuning parameter. An ideal experiment with infinite resources would find the very best[3] value of $\bar{\sigma}_p$ for every

[3]The notion of "best" also needs to be defined.



(a) Absolute Trajectory Error



(b) Relative Pose Error

Figure 5.3: $\bar{\sigma}_p = \sigma_p$**: Performance decreases with Gaussian Noise.** Each box-and-whisker illustrates the distribution of Absolute Trajectory Error (top) and Relative Pose Error (bottom) over 100 Monte-Carlo trials. Boxes extend from the first to the third quartile. Medians are lines in the boxes, means are triangles. Whiskers extend the box by 1.5x the inter-quartile range. All other points are plotted as "fliers". Mean and median error increase with noise for all $\bar{\sigma}_p = \sigma_p \geq 0.50$. The performance is lower for $\sigma_p = 0.25$ than for $\sigma_p = 0.50$ because $\bar{\sigma}_p = 0.25$ is too small to capture uncertainties due to poor feature initialization in Brownian motion in addition to Gaussian noise.

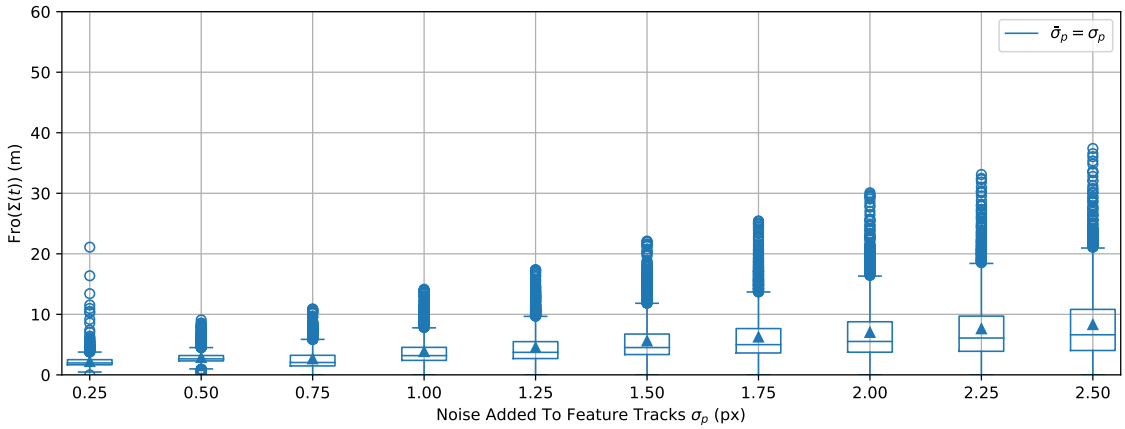


Figure 5.4: $\bar{\sigma}_p = \sigma_p$**: Gaussian Noise leads to larger sample covariances.** Each box-and-whisker illustrates the distribution of sample covariance (eq. (5.1)) computed using 100 Monte-Carlo trials. Boxes extend from the first to the third quartile. Medians are lines in the boxes, means are triangles. Whiskers extend the box by 1.5x the inter-quartile range. All other points are plotted as "fliers". As the amount of noise increases, so does the sample covariance and the variation in sample covariance.

possible value of every disturbance and then compare the distribution of performance errors, mean sample covariance, and scales. The second limitation is that we studied a randomly generated trajectory to eliminate the possibility that the number of trajectories that could produce the same measurements could be unbounded. Many realistic trajectories, such as driving a car on a flat road or flying a drone at a constant velocity, are not sufficiently exciting.

Nevertheless, we have systematically quantified and characterized the performance and uncertainty of an well-known monocular-VIO state estimation algorithm based on the Extended Kalman Filter. Our results largely confirm what is anecdotally known by practitioners: monocular VIO is "finicky". Whenever possible, a practitioner will always choose RGB-D SLAM, Lidar-Inertial Odometry, or Stereo Visual-Inertial Odometry if the platform allows it. The fliers in Figures 5.3, 5.5, 5.9, and 5.11 indicate that the right combination of noise and/or drift in the IMU and visual measurements has more than a 1 in 100 chance of creating a state estimate with a performance error more than twice the mean error and far

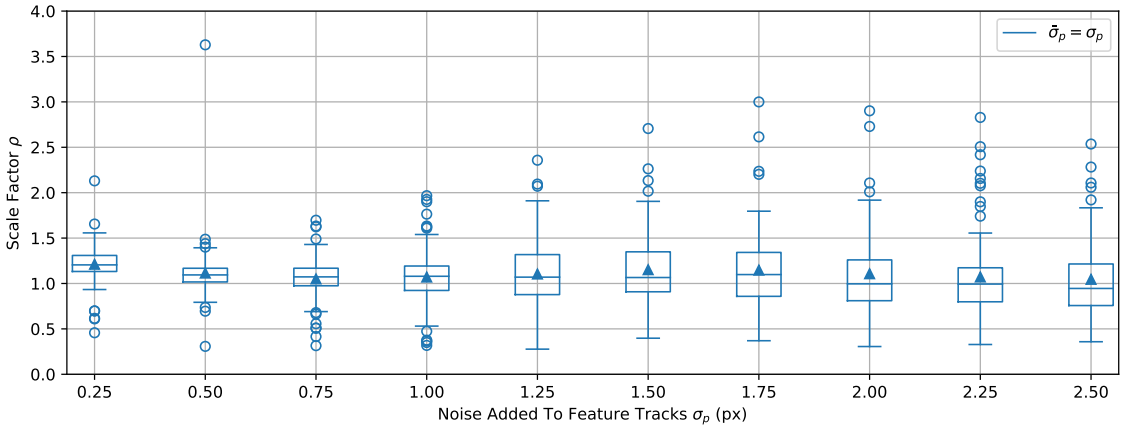
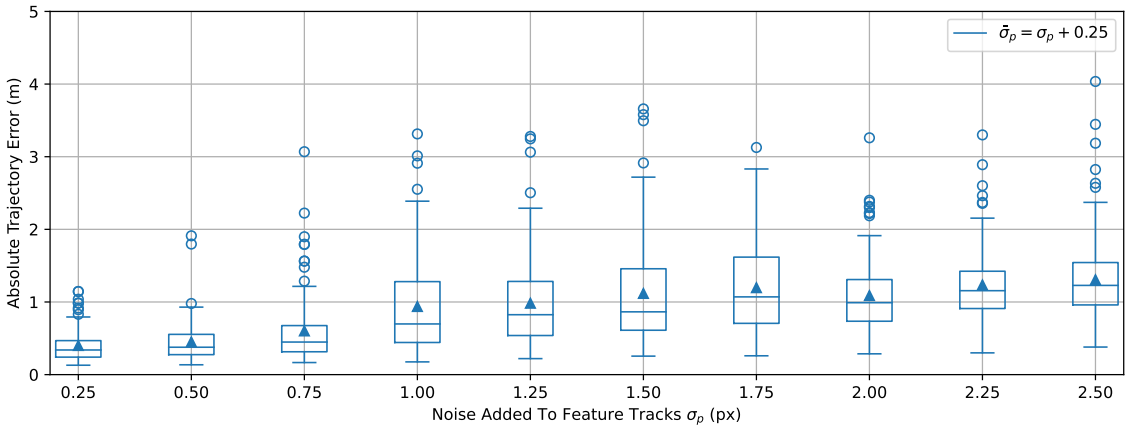
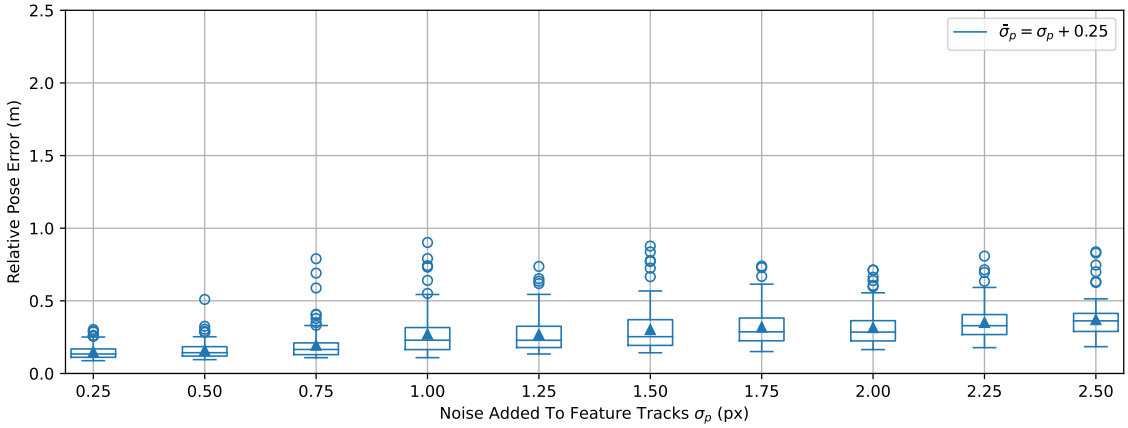


Figure 5.5: $\bar{\sigma}_p = \sigma_p$**: Mean and variation of scale factor** ρ **is a nonlinear function of** σ_p**.** Each box-and-whisker illustrates the distribution of ρ computed using 100 Monte-Carlo trials. Boxes extend from the first to the third quartile. Medians are lines in the boxes, means are triangles. Whiskers extend the box by 1.5x the inter-quartile range. All other points are plotted as "fliers". Generally, we see that although there is no trend in the mean or median scale, the variation in scale generally increases with σ_p. Scale estimates are relatively poor for $\sigma_p = 0.25$ because $\bar{\sigma}_p = 0.25$ is too small to capture uncertainties due to poor feature initialization in addition to Gaussian noise.



(a) Absolute Trajectory Error



(b) Relative Pose Error

Figure 5.6: $\bar{\sigma}_p = \sigma_p + 0.25$**: Performance decreases with Gaussian Noise.** Each box-and-whisker illustrates the distribution of Absolute Trajectory Error (top) and Relative Pose Error (bottom) over 100 Monte-Carlo trials. Boxes extend from the first to the third quartile. Medians are lines in the boxes, means are triangles. Whiskers extend the box by 1.5x the inter-quartile range. All other points are plotted as "fliers". Mean and median error increase with noise for all $\bar{\sigma}_p = \sigma_p \geq 0.50$.

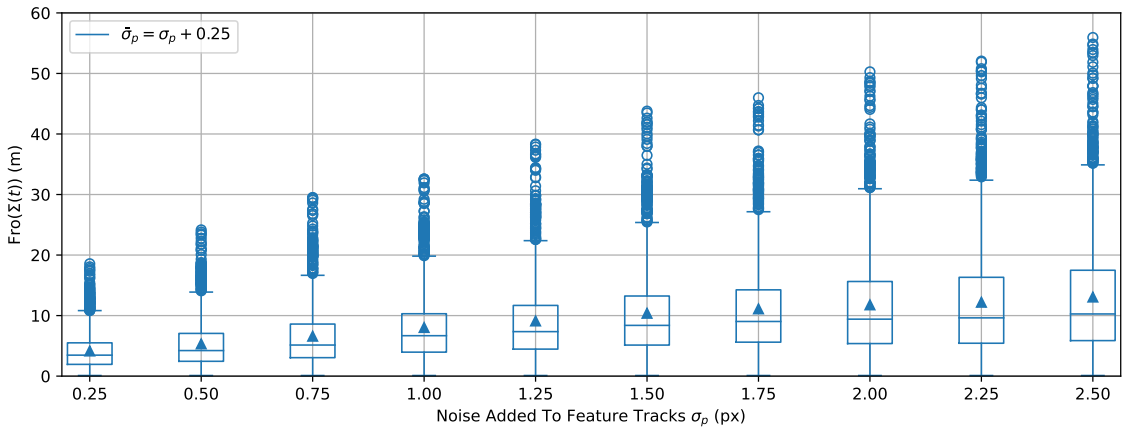


Figure 5.7: $\bar{\sigma}_p = \sigma_p + 0.25$**: Gaussian Noise leads to larger sample covariances.** Each box-and-whisker illustrates the distribution of mean sample covariance (eq. (5.1)) computed using 100 Monte-Carlo trials. Boxes extend from the first to the third quartile. Medians are lines in the boxes, means are triangles. Whiskers extend the box by 1.5x the inter-quartile range. All other points are plotted as "fliers". As the amount of noise increases, so does the sample covariance and the variation in sample covariance.

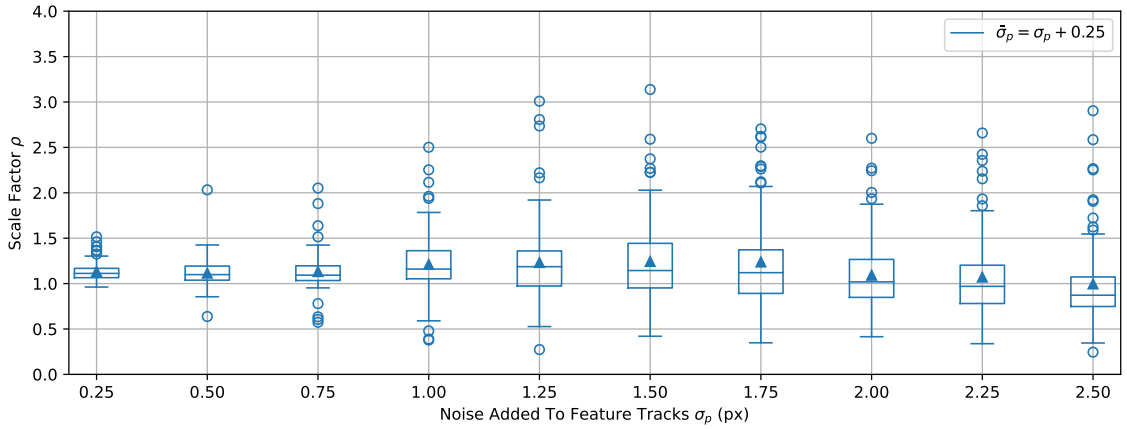
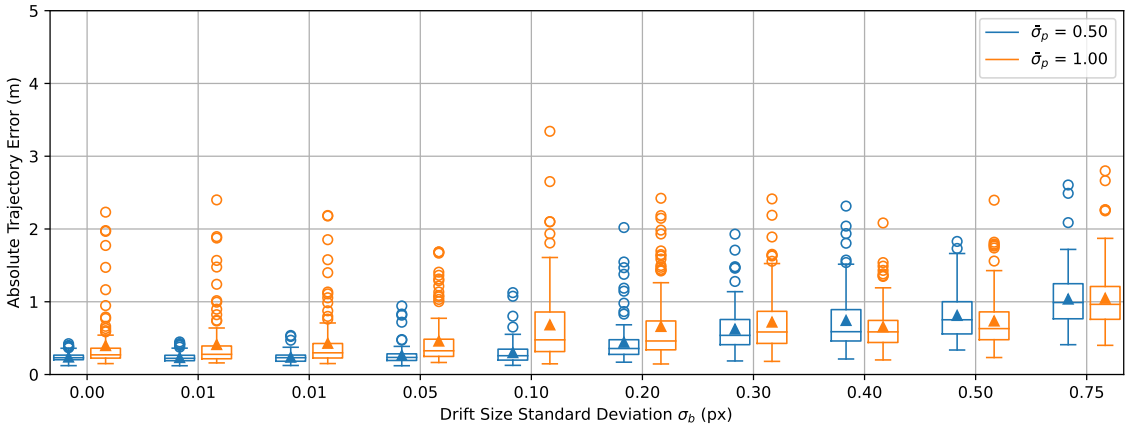
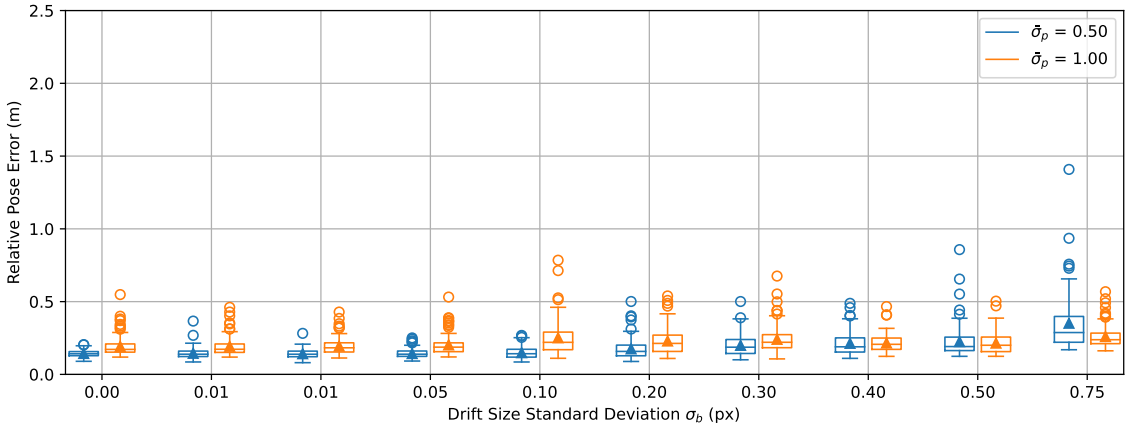


Figure 5.8: $\bar{\sigma}_p = \sigma_p + 0.25$**: Mean and variation of scale factor ρ is a nonlinear function of σ_p.** Each box-and-whisker illustrates the distribution of ρ computed using 100 Monte-Carlo trials. Boxes extend from the first to the third quartile. Medians are lines in the boxes, means are triangles. Whiskers extend the box by 1.5x the inter-quartile range. All other points are plotted as "fliers". Generally, we see that although there is no trend in the mean or median scale, the variation in scale generally increases with σ_p.



(a) Absolute Trajectory Error



(b) Relative Pose Error

Figure 5.9: **Drift increases estimation error and variation in estimation error.** Each box-and-whisker illustrates the distribution of ATE and RPE computed using 100 Monte-Carlo trials. Boxes extend from the first to the third quartile. Medians are lines in the boxes, means are triangles. Whiskers extend the box by 1.5x the inter-quartile range. All other points are plotted as "fliers". The mean and median performance error creeps upwards with the drift σ_b for both $\bar{\sigma}_p = 0.50$ and $\bar{\sigma}_p = 1.00$. Performance is better when $\bar{\sigma}_p = 0.50$ for smaller amounts of drift; for values of $\sigma_b \geq 0.40$, $\bar{\sigma}_p = 1.00$ is better.

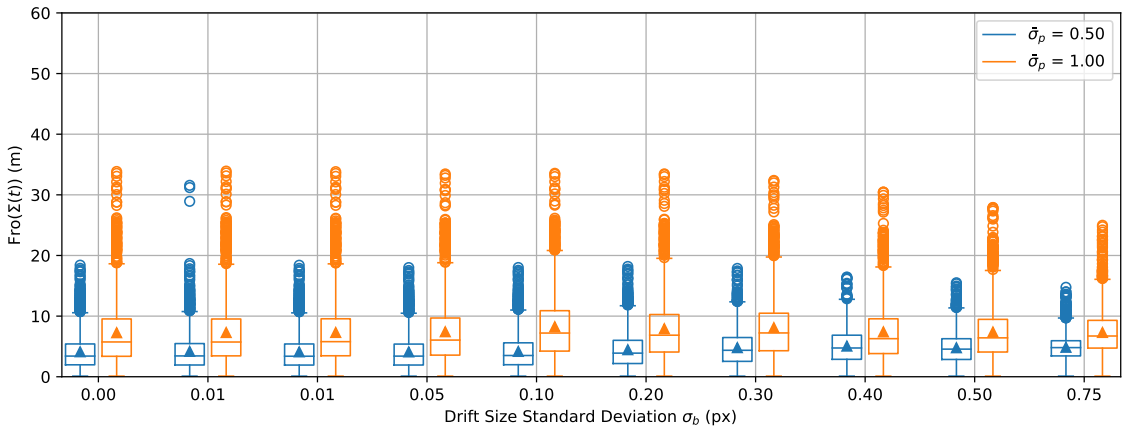


Figure 5.10: **Drift only slightly increases state uncertainty.** Each box-and-whisker illustrates the distribution of sample covariance (eq. (5.1)) computed using 100 Monte-Carlo trials. Boxes extend from the first to the third quartile. Medians are lines in the boxes, means are triangles. Whiskers extend the box by 1.5x the inter-quartile range. All other points are plotted as "fliers". Mean and median covariance size increases with drift. For both $\bar{\sigma}_p = 0.50$ and $\bar{\sigma}_p = 1.00$, the mean and median values of drift creep slightly upwards with increasing values of $\bar{\sigma}_b$.

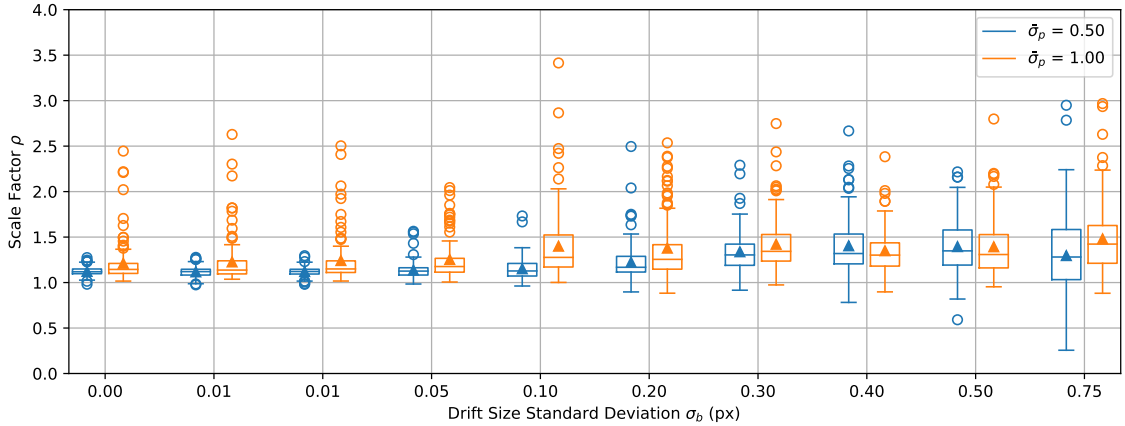
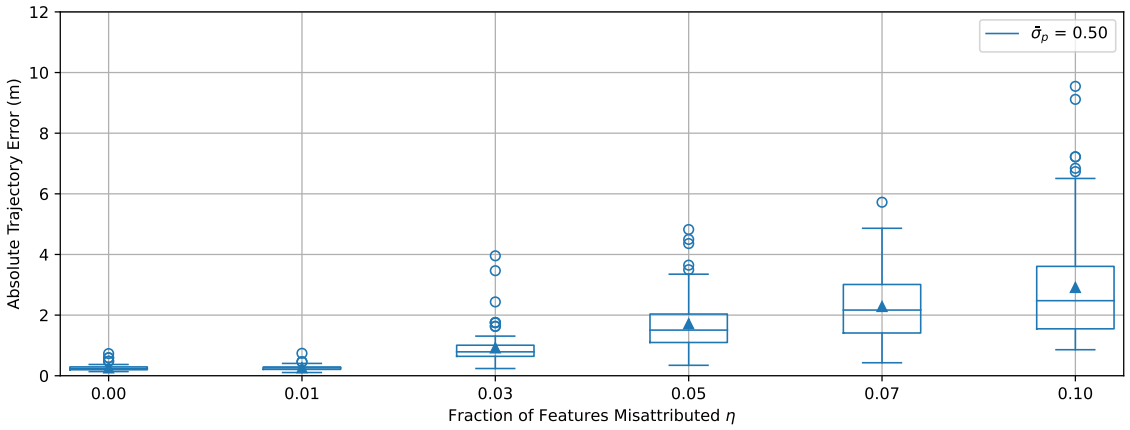
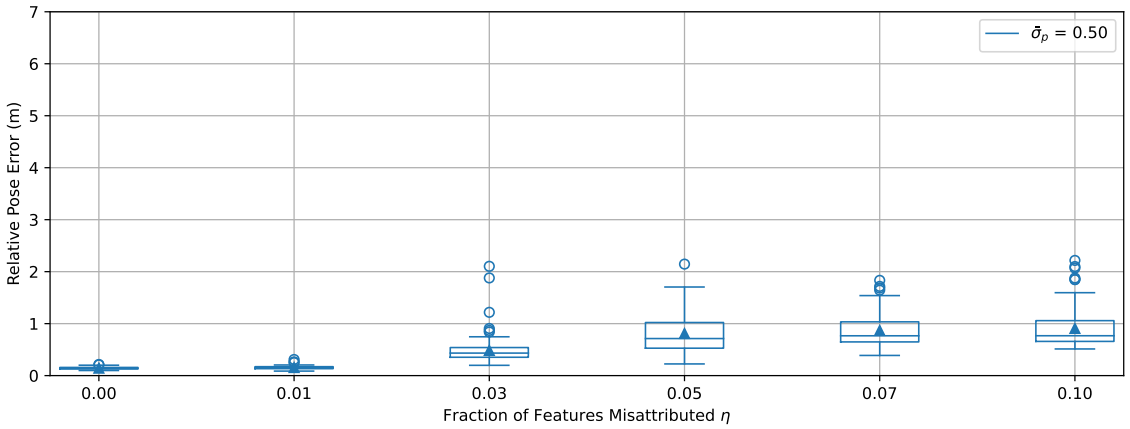


Figure 5.11: **Drift increases both bias and uncertainty in scale.** Each box-and-whisker illustrates the distribution of scale factor (eq. (5.3)) computed using 100 Monte-Carlo trials. Boxes extend from the first to the third quartile. Medians are lines in the boxes, means are triangles. Whiskers extend the box by 1.5x the inter-quartile range. All other points are plotted as "fliers".



(a) Absolute Trajectory Error



(b) Relative Pose Error

Figure 5.12: **Attribution errors increase bias and variance in performance.** Each box-and-whisker illustrates the distribution of ATE and RPE computed using 100 Monte-Carlo trials. Boxes extend from the first to the third quartile. Medians are lines in the boxes, means are triangles. Whiskers extend the box by 1.5x the inter-quartile range.

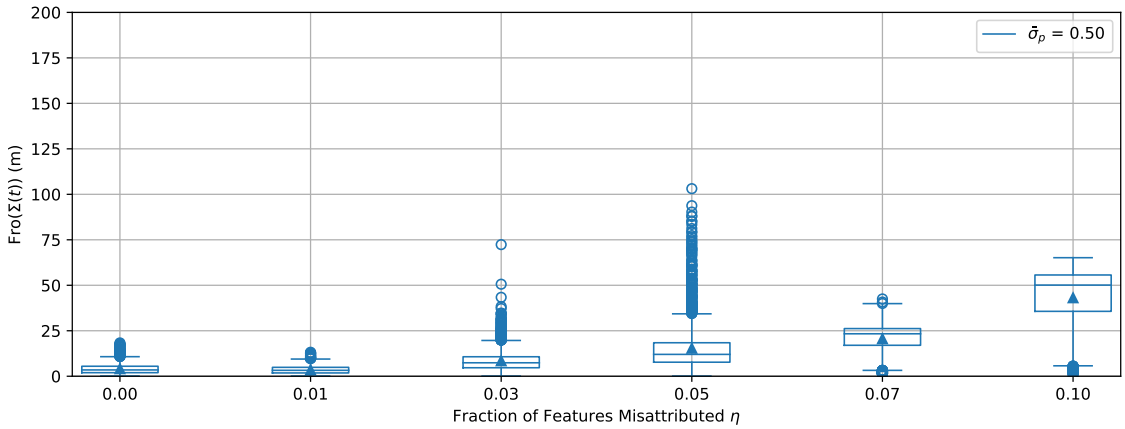


Figure 5.13: **Attribution errors produce more uncertainty.** Each box-and-whisker illustrates the distribution of mean sample covariance computed using 100 Monte-Carlo trials. Boxes extend from the first to the third quartile. Medians are lines in the boxes, means are triangles. Whiskers extend the box by 1.5x the inter-quartile range.

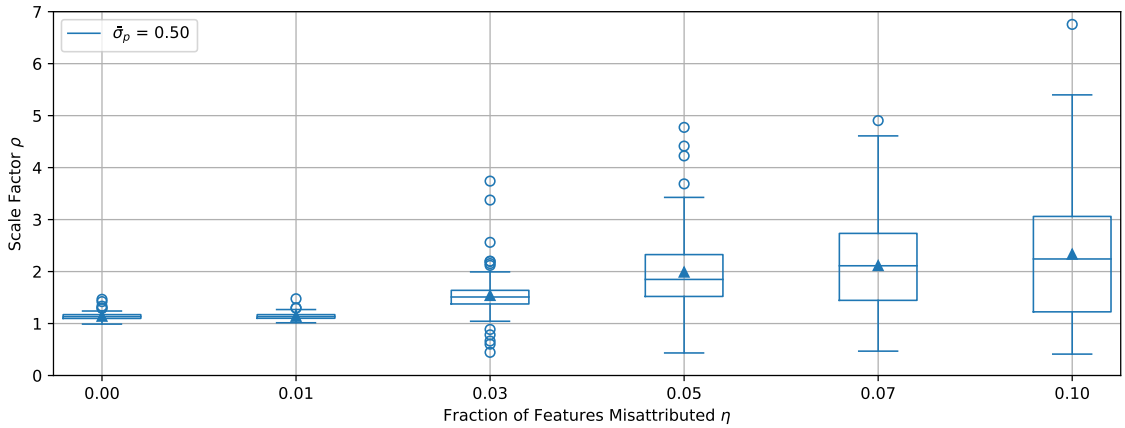


Figure 5.14: **Attribution errors increase bias and variance in scale.** Each box-and-whisker illustrates the distribution of scale factor computed using 100 Monte-Carlo trials. Boxes extend from the first to the third quartile. Medians are lines in the boxes, means are triangles. Whiskers extend the box by 1.5x the inter-quartile range.

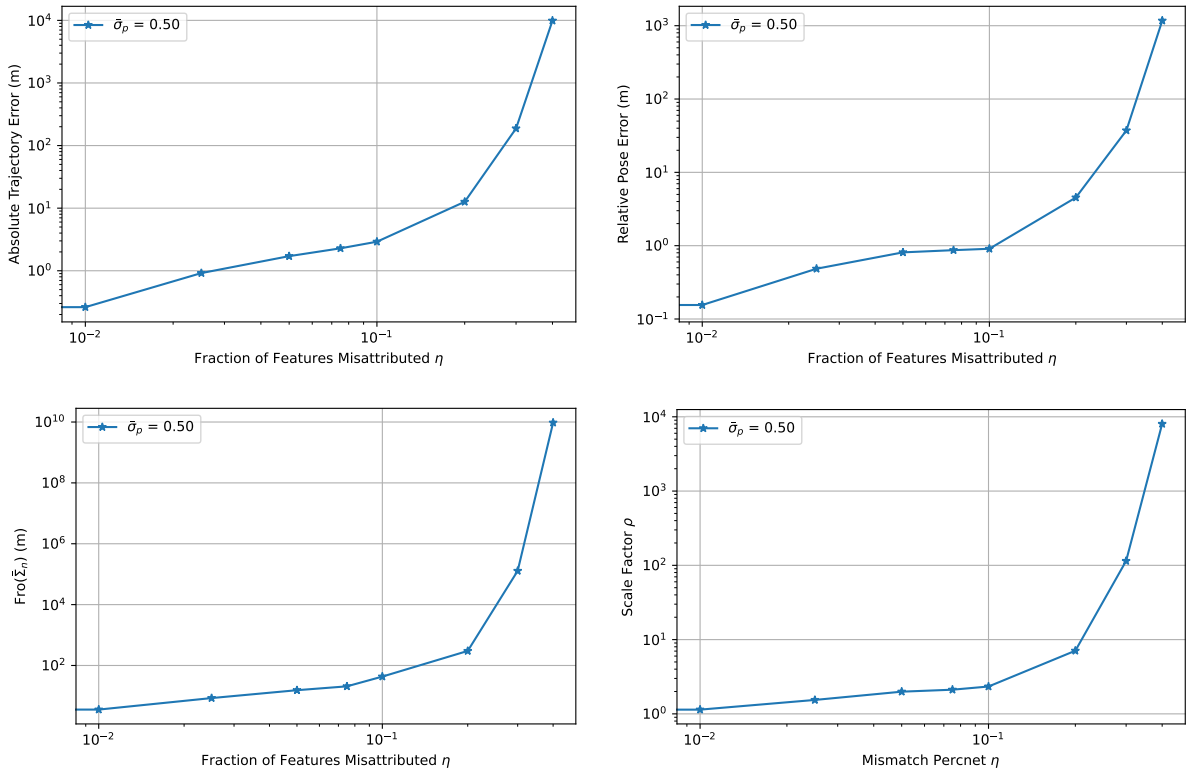


Figure 5.15: **The increases in performance errors, state uncertainty, and scale uncertainty due to attribution errors are exponential.** The four figures plot mean values of ATE, RPE, state uncertainty, and scale factor as a function of η, for $\eta \in [0.01, 0.025, 0.05, 0.075, 0.1, 0.2, 0.3, 0.4]$. Both the horizontal and vertical axes are in log scale. All curves are exponential functions of η.

above the lowest possible error. This highlights the need for more Monte-Carlo studies of SLAM, rather than reliance on benchmark datasets with individual trajectories.

CHAPTER 6

Scene Uncertainty of Deterministic Image Classifiers

Finally, we turn our attention to Assumption 4 in Figure 1.1. Object detection neural networks, such as the one in used in Chapter 2, identify the class and location of detected objects. Since assessing the uncertainty of a neural network is a new topic, we simplified our analysis and focused only on the uncertainty of the class of detected objects. The text below is therefore about simpler image classification networks instead of object detection networks.

6.1 Introduction

It appears that Deep Neural Networks (DNNs) can classify images as well as humans, at least as measured by popular benchmarks, yet small perturbations of the images can cause changes in the predicted class. Even excluding adversarial perturbations, simply classifying consecutive frames in a video shows variability inconsistent with the reported error rate (see Figure 6.1). So, how much should we trust image classifiers? How *confident* should we be of the outcome they render on a given image? There is a substantive literature on uncertainty quantification, including work characterizing the (epistemic and aleatoric) uncertainty of trained classifiers (see Section 1.3.4). Such uncertainty is a property of the *classifier*, not of the *outcome* of inference on particular datum. We are instead interested in ascertaining how confident to be in the response of a particular DNN model to a particular image, not generally how well the classifier performs on images from a given class: Say we have an image x, and a DNN that computes a discriminant vector $y = f(x) \in \mathbb{R}^K$ with as many components as the number of classes K (*e.g.,* logits or softmax vector) from which it returns the estimated label $\hat{k} = \arg\max_k y_k$ ="cat." How sure are we that there is a cat in *this*

image? Even if the classifier in question was wrong most of the times, even on the class "cat," so long as it is confident that its answer on this particular image is correct, we would be content. If faced with the question "are you sure?" a human would take a second look, or capture a new image, to either confirm or profess doubt. But a DNN classifier would return the same answer, correct or not, since most real-world deep networks in use today are deterministic maps f from the input x to the output y.

Since the classifier is deterministic, and the image is given, the first **key question** we must address is *with respect to what variability should uncertainty be defined and evaluated.* To this end, we define *scene uncertainty* of a deterministic image classifier, which is *the distribution of outcomes that* would *have been obtained in response to data that* could *have been generated by the same scene that produced the given image.* Simply put, there are no cats in images, only pixels. Cats are in the *scene*, about which images provide evidence. The question of whether we are "sure" of the outcome of the an image classifier is therefore of counterfactual nature: Had we been given *different* images of the same scene, would an image-based classifier have returned the same outcome?

Formally, given an image x, if we could compute the posterior probability of the estimated label, $\hat{k}$ from which we can then measure confidence intervals, entropy, and other statistics commonly used to measure uncertainty, what we are after is *not* $P(\hat{k}|x)$. Instead, it is $P(\hat{k}|\{x'\})$ where $x' \sim p(x'|S)$, and S is the scene that yielded the given image x. The scene S, whether real or imagined, is the vehicle that allows one to *guess what is not known (S) from what is known (x)*, which is the inductive process. Computing scene uncertainty requires knowledge of the scene that generated the given image, or possibly a very good guess, which we discuss in Sect. 6.2.

Contribution. Unlike work described in Section 1.3.4, ours does not propose a new measure of uncertainty nor a new way to calibrate the discriminant to match empirical statistics: We use standard statistics computed from the "posterior probability", such as covariance and entropy, to measure uncertainty. The core of our work aims to specify *with respect to what posterior* to measure uncertainty. Since deep network image classifiers used in the real world

are deterministic, the choice is consequential, and yet seldom addressed explicitly in the existing literature. Scene uncertainty is introduced to explicitly characterize the variability with respect to which uncertainty is measured.

6.2 Method

We start by introducing the nomenclature used throughout the rest of the paper:

– The **scene** S is an abstraction of the physical world. It can be thought of as a sample from some distribution $S \sim p_S$ that is unknown and arguably unknowable. The scene itself (not just its distribution) is arguably unknowable but for some of its "attributes" manifest in sensory data.

– An **attribute** k is a characteristic of the scene that belongs to a finite set (*e.g.,* names of objects), $k(S) \in \{1, \dots, K\}$. Note that there can be many scenes that share the same attribute(s) (**intrinsic variability**). For instance, k can be the label "cat" and $p_S(\cdot|k)$ is the distribution of scenes that contain a "cat." Continuous, but finitely-parametrized, attributes are also possible, for instance related to shape or illumination.

– **Extrinsic variability** g_t is an unknown transformation of the scene that changes its manifestation (measurements, see next point) but not its attributes. It can be thought of as a sample from some nuisance distribution $g_t \sim p_g$. For instance, extrinsic variability could be due to the vantage point of the camera, the illumination, partial occlusion, sensor noise, quantization *etc.*, none of which depends on whether the scene is labeled "cat". Note that there can be spurious correlations between the attribute and nuisance variability: An indoor scene is more likely to contain a cat than a beach scene. Nevertheless, if there were a cat on the beach, we would want our classifier to say so with confidence. The fact that nuisance variables can correlate with attributes on a given dataset may engender confusion between intrinsic and extrinsic variability. To be clear, phenomena that generate intrinsic variability would not exist in the absence of the attribute of interest. The pose, color and shape of a cat do not exist without the cat. Conversely, ambient illumination (indoor vs. outdoor) exists regardless of whether there is a cat, even if correlated with its presence. The effect

of nuisance variability on confidence, unlike intrinsic variability, is correlational, rather than causal.

– A **measurement** x_t is a known function of both the scene S (and therefore its attributes) and the nuisances. We will assume that there is a generative model that, if the scene S was known, and if the nuisances g_t were known, would yield a measurement up to some residual (white, zero-mean, homoscedastic Gaussian) noise n_t

$$x_t = h(g_t, S) + n_t. \tag{6.1}$$

For example, h can be thought of as a graphics engine where all variables on the right hand-side are given.

– The **discriminant** $y = f(x)$ is a deterministic function of the measurement that can be used to infer some attributes of the scene. For instance, $y = P(k|x) \in \mathbb{S}^{K-1}$ is the Bayesian discriminant (posterior probability). More in general, y could be any element of a vector (embedding) space.

– The **estimated class** $\hat{k}(x)$ is the outcome of a classifier, for instance $\hat{k} = \arg\max_k [f(x)]_k$. Given an image x and a classifier $\hat{k}(\cdot)$, we reduce questions of confidence and uncertainty to the posterior probability $P(\hat{k} = k|x)$. In the absence of any variability in the estimator f, defining uncertainty in the estimate $\hat{k}$ requires *assuming* some kind of variability. The Wellington Posterior hinges on the following assumptions:

- The class k is an attribute of the scene S and is independent of intrinsic and extrinsic variability, by their definition.
- We posit that, when asking "how confident we are about the class $\hat{k}(x)$" we do *not* refer to the uncertainty of the class given that image, which is zero. Instead, we refer to uncertainty of the estimated class *with respect to the variability of all possible images of the same scene S which could have been obtained* by changing nuisance (extrinsic) variability.

In other words, if in response to an image, a classifier returns the label "cat," the question is *not* how sure to be about whether there is a cat in the image. The question is how sure

to be that there is a cat *in the scene portrayed by the image.* For instance, if instead of the given image, one was given a slightly different one, captured slightly earlier or a little later, and the classifier returned "dog," would one be less confident in the answer than if it had also returned "cat"? Intuitively yes. Hypothetical repeated trials would involve not running the same image through the classifier over and over, but capturing different images of the same scene, and running each through the classifier. Of course, different images obtained by adding noise would be a special case where the world is static and the only nuisance variability is due to sensor noise.

Intrinsic variability does not figure in the definition of scene uncertainty. The fact that we are given *one* image implies that we are interested in *one* scene, the one portrayed in the image. So, the question of how sure we are of the answer "cat" given an image is not how frequently the classifier correctly returns the label "cat" on different images *of different scenes* that contain different cats. It is a question about *the particular scene portrayed by the image we are given*, with the given cat in it. The goal is *not* $P(\hat{k}|k)$, which would be how frequently we say "cat" when there is one (in some scene). We are interested in *this* scene, the one portrayed by the image. Written as a Markov chain we have

$$k \to S \to x \to y \to \hat{k} \tag{6.2}$$

where the first arrow includes intrinsic variability (a particular attribute is shared by many scenes) and the second arrow includes nuisance/extrinsic variability (a particular scene can generate infinitely many images). The last two arrows are deterministic. We are interested *only* in the variability in the second arrow. To compute scene uncertainty, we observe that

$$\begin{aligned} & P(\hat{k} = k|S) \\ & = \int P(\hat{k}|x) dP(x|S) \\ & = \int \delta(\hat{k}(x) - k) dP(x|S) \\ & = \int \delta(\arg\min f(\underbrace{h(g,S)}_{x}) - k) dP(g) \\ & \simeq \frac{1}{T} \sum_{t=1}^{T} \delta(\arg\min f(\underbrace{h(g_t,S)}_{x_t}) - k). \end{aligned} \tag{6.3}$$

That is, given samples from the nuisance variability g_t, or sample images generated by changing nuisance variability, x_t, we can compute the probability of a particular label by counting the frequency of that label in response to different nuisance variability. We defer the question of whether the samples given are fair or sufficiently exciting. In the expression above, f is computed by the given DNN classifier, g_t is from a chosen class of nuisance transformations, and h is an image formation model that is also chosen by the designer of the experiment.

As we defined it, the scene is an abstraction of the physical world. Such abstraction can live inside the memory of a computer. Since a scene is only observed through images of it, if a synthetic scene generates images that are indistinguishable from those captured of a physical scene, the real and synthetic scenes – while entirely different objects – are equivalent for the purpose of computing scene uncertainty. Thus a "scene" could be any generative model that can produce images that are indistinguishable from real ones *including* the given one. Different images are then obtained by perturbing the scene with nuisance variability.

6.2.1 Proposed Measures of Scene Uncertainty

We propose to use the following as measures of uncertainty that can computed from a posterior distribution $p(y|S)$ or $p(\hat{k}|S)$. In the text below, assume that for a sample of N images from a single scene, we have logits y_i, softmax vectors s_i, and predictions $\hat{k}_i$ for $i = 1, \dots N$.

- **Logit Spread:** Frobenius norm of the covariance matrix of the logits, $\| \operatorname{Cov}(y) \|_F$.
- **Softmax Spread:** Standard deviation of cosine similarity distance between the softmax vectors, $\operatorname{StdDev}(\arccos(s \cdot \bar{s}))$, where $\bar{s}$ is the mean softmax vector.
- **Percent Non-Mode:** The percentage of predictions $\hat{k}_i$ that are not equal to $\bar{k}$, $P(\hat{k} \neq \bar{k})$, where $\bar{k}$ is the most often predicted class within that scene.
- **Scene Entropy:** Entropy of the histogram of the $\hat{k}_i$ within a scene, $\operatorname{Entropy}(P(\hat{k}|S))$.

6.3 Measuring Scene Uncertainty

6.3.1 Datasets

We performed experiments on two datasets, Objectron [AZA21], and the ImageNetVid [RDS15] and ImageNetVid-Robust [SDR19] datasets. These datasets are ideal for our image classification experiments because they contain multiple images of the same scene and each scene contains a single object of interest that is visible in all the images. ImageNetVid was created in 2015 and consists of videos with bounding boxes for 30 classes. In 2017, additional annotated videos were added to the ImageNetVid validation dataset and in 2019, a set of frames, mostly selected from the 2017 videos, were human-curated into the ImageNetVid-Robust dataset. Each "video" in ImageNetVid-Robust consists of 21 frames: 20 regular frames and one designated "anchor" frame. Temporally, the anchor frame is the middle of the other frames. Human curation guaranteed that the object of interest would be clearly visible in each frame and that the frames would be quite similar. We use ImageNetVid frames as a training set and ImageNetVid-Robust frames as a test set in our experiments.

Objectron [AZA21] contains short video clips of scenes with 9 classes: Bikes, Books, Bottles, Cameras, Cereal Boxes, Chairs, Cups, Laptops, and Shoes. Videos in the dataset consist of humans moving a smartphone around a static object. Motions in all videos are quite similar: a human waves a smartphone around a stationary object with large changes in the azimuth angle between a point on the object and the camera, but smaller changes in the elevation between a point on the object and the camera and in the distance from the object to the camera. We selected a subset of 5000 videos for training set, 500 videos for validation set, and 500 for testing — there is no overlap in video frames between training, validation and test splits. From each video clip, we selected 20 evenly-spaced frames to serve as an empirical paragon and designated the middle frame the anchor image.

In general, evenly distributed frames across a short video clip do not provide a fair sample from the population of all possible images that the scene could have generated, so any resulting statistics cannot be considered proper ground truth. A better experiment would use synthetically generated 3D scenes containing all possible intrinsic and extrinsic variability

instead of natural videos. However, this would require manually creating thousands of 3D scenes, each focused on a different object; the cost of this task is prohibitive. Another possibility would be to create a smaller group of synthetic scenes containing the same classes of objects as Objectron and ImageNetVid and use domain adaptation to adapt the synthetic scenes to natural images; the cost-prohibitive part would then become manually creating many extrinsic variabilities found in scenes. For example, creating a laptop scene for the Objectron dataset in this manner would require a 3D model of not only the laptop, but the table beneath it and decorations around it. Therefore, an empirical baseline of 20 uniformly sampled frames from each video serves as a first step in evaluating methods to predict scene uncertainty.

So that we would have a fairer dataset, we created two additional tiny datasets to accompany the Objectron dataset: one synthetic and one real. Each dataset consists of two items from each class. The one consisting of real images "MiniObjectron", has 20 images per scene, captured from a set of positions around the object with low precision. The one consisting of synthetic images, "SyntheticObjectron", consists of 468 images per scene, captured from a set of positions around the object with perfect precision.

6.3.2 Discriminant

In order to compute the empirical posterior and the various forms of Wellington Posterior, we need a discriminant function $f(\cdot)$, from which to build a classifier. We use the backbone of an ImageNet-pretrained ResNet-50 or ResNet-101 [HZR16], where $f(x)$ is called the vector of logits, whose maximizer is the selected class, and whose normalized exponential is called softmax vector. Fine-tuning and validation for ResNet-50 on Objectron, and ResNet-101 on ImageNetVid, used one frame from each scene. The focus is not to achieve the highest possible accuracy, but to provide a meaningful estimate of uncertainty relative to the variability of different images of the same scene. For this reason, we select the most common, not the highest performing, baseline classifier. Hyperparameters, training error, and validation error are given in Table 6.1. Test error on the Objectron test set was 0.9607 ± 0.0015 for all

frames and 0.9693 ± 0.0076 for only the anchor frames. Test error on ImageNetVid-Robust was 0.7187 ± 0.0084 for all frames and 0.7207 ± 0.0139 for only the anchor frames. These test errors do not reflect the variation shown in Figure 6.1.

Parameter	**Objectron**	**ImageNetVid**
Architecture	ResNet-50	ResNet-101
Init. Learning Rate	0.01	0.001
Momentum	0.9	0.9
Weight Decay	1e-5	1e-5
γ	0.1	0.1
Milestones	25, 35	25, 35
Max Epochs	50	50
Batch Size	32	32
Selected Epoch	50, 40, 40	30, 50, 50
Training Error	0.0326,0.0366,0.0318	0.0478,0.0478,0.0476
Validation Error	0.0320,0.0220,0.0320	0.2747,0.2711,0.2912

Table 6.1: Hyperparameters and decision variables used in fine-tuning image classifiers on the Objectron and ImageNetVid datasets using stochastic gradient descent and the cross-entropy loss for all three trials. Reported training and validation errors are for the selected epoch, not after the maximum number of epochs.

6.3.3 Scene Uncertainty Values

We computed the measures of uncertainty proposed in Section 6.2.1 (logit spread, softmax spread, percent non-mode predictions, and scene entropy) over empirical paragon scenes to compute baseline uncertainty values. Baseline values are plotted as a histogram in Figure 6.1. Means and standard deviation of baseline values from three separate experiments are shown in Table 6.2. Both Figure 6.1 and Table 6.2 show that scene uncertainty is a real phenomena.

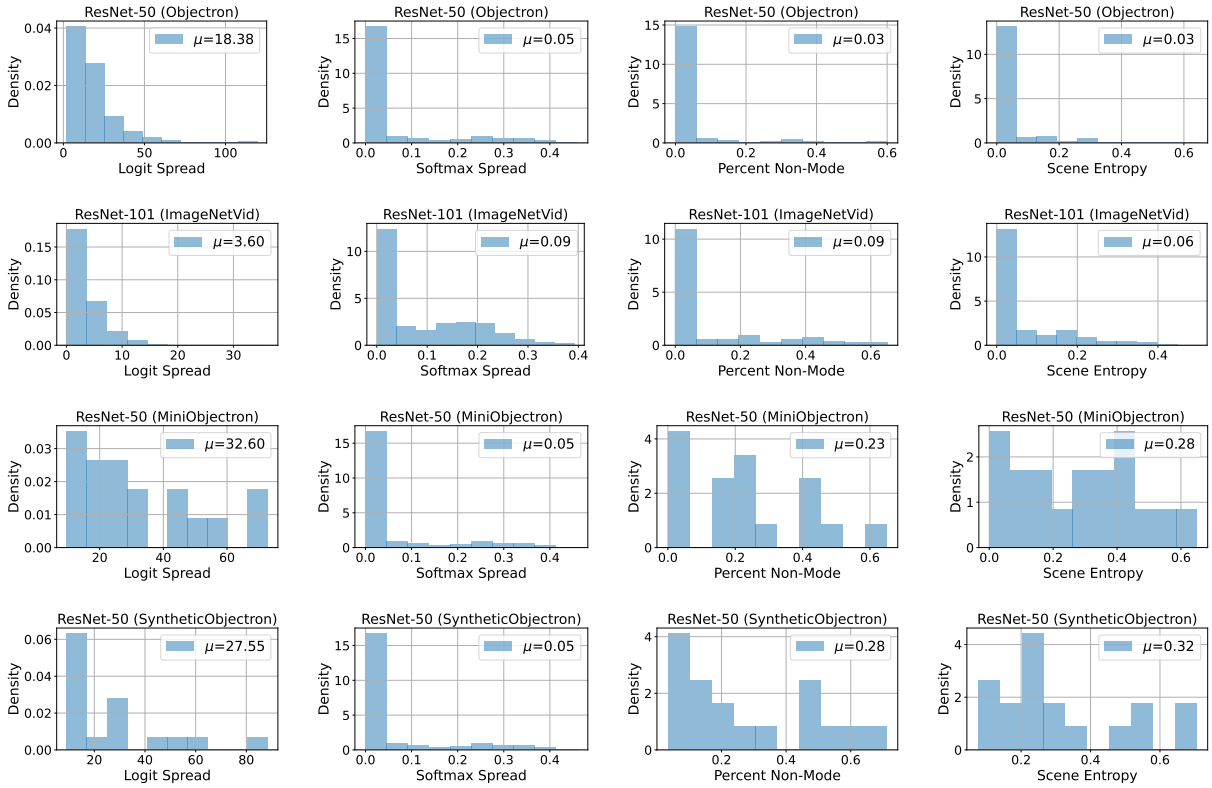


Figure 6.1: **Scene Uncertainty is Real.** Histograms of the four measures of scene uncertainty (logit spread, softmax spread, percent non-mode, and scene entropy) from Section 6.2.1 for ResNet-50 on Objectron (top row), ResNet-101 on ImageNetVid (second row), ResNet-50 on MiniObjectron (third row), and ResNet-50 on SyntheticObjectron (bottom row). Each point in the histograms above is a scene. The mean of each distribution is shown in the top-left. For the cases when the uncertainty of the network is measured over the test split of the same dataset it was trained on (top two rows) thick bars at the left of a histogram indicates that most images of most scenes are classified identically within a scene and that discriminants are similar. However, there is a long tail in the plots showing that scene uncertainty is a real phenomenon that will be encountered in real systems. In the bottom two rows, where the uncertainty of ResNet-50 was measured on our two "more fair" constructed datasets, the measures of scene uncertainty are much higher.

	ImageNetVid	**Objectron**	**Mini Objectron**	**Synthetic Objectron**
Logit Spread	0.5618 ± 0.0131	1.5803 ± 0.0077	32.8264 ± 0.5193	29.7664 ± 2.0210
Softmax Spread	0.0619 ± 0.0000	0.0418 ± 0.0000	0.2652 ± 0.0000	0.3142 ± 0.0000
Percent Non-Mode	0.0933 ± 0.0026	0.0315 ± 0.0019	0.2269 ± 0.0273	0.2908 ± 0.0125
Scene Entropy	0.2129 ± 0.0041	0.0774 ± 0.0036	0.2692 ± 0.0254	0.3403 ± 0.0135

Table 6.2: This table accompanies Figure 6.1. Entries are means and standard deviations of proposed measures of uncertainty (logit spread, softmax spread, percent non-mode predictions, and scene entropy) computed using the empirical paragon. Means and standard deviations come from performing all calculations over all datasets three times using different trained networks. The only difference between networks is the data shuffling used at training time.

6.4 Summary

We have defined a new concept, *uncertainty with respect to variation in the scene*, or $p(y|S)$. Experiments in Section 6.3 illustrate that this phenomena exists and that Assumption 4 from Figure 1.1 is false. Integration of neural networks into architectures such as Figure 1.1 should therefore be cognizant of Scene Uncertainty.

CHAPTER 7

Discussion

In the introduction, we noted that if the assumptions made by the components of the framework presented in Figure 1.1 and Chapter 2 were not met, that any successful experiment is successful because of luck rather than correctness. Work in the subsequent chapters showed that there was some luck in the experiment's success: the overconfident covariance estimate from XIVO was just large enough that the MPC navigated Alphred around the obstacle. The simulated obstacle looked just enough like the real cardboard boxes in the training set that the object detection algorithm found an obstacle. However, the demonstration of Chapter 2 was meant to be a preliminary demonstration. Less luck and better covariance estimates, and an uncertainty-aware integration of the object detector would have achieved the same successful result in Figure 2.6.

The motivation for this work was the verification and validation of autonomous systems with computer vision in the loop. Our proposed framework, and subsequent analyses, is a classical architecture with a combination of classical components and neural network components. The neural network components perform semantic vision and belief-space prediction, two tasks where classical algorithms have never been successful. We could have chosen to use a reinforcement learning network, such as those described in [SZY21], instead of a model predictive controller. We could also have used a deep network, such as [CWW17,KGC15], instead of XIVO. The exact opposite approach would be to an end-to-end learning framework, such as [ZHL21].

Our choices are themselves a statement. For safety-critical systems, we value the explainability and transparency that classical algorithms provide. Designing components separately, rather than end-to-end, also enables transparently. Isolating neural networks to specific tasks

avoids the difficult problem of explainability and reduces the amount of data that must be collected as much as possible. However, when the performance gap between a classical component and a learned component widens enough, the correct choice is to use the learned component, with uncertainty quantification and awareness providing system safety.

7.1 Future Work

We list ways in which the content of each chapter could be used in future endeavors.

7.1.1 Chapter 2: "Risk-Averse MPC via Visual-Inertial Input and Recurrent Networks for Online Collision Avoidance"

The architecture in Chapter 2 is an initial demonstration using a low-dimensional MPC and "off-the-shelf" components. Since the demonstration was successful, future work that builds off of the demonstration will consist of either improving existing components, or extending or adding components for more complex scenarios. For example, rather than using generic VIO equations in XIVO, XIVO's equations of motion could explicitly model a walking gait. Then, the performance-limiting detuning and signal clipping will become unnecessary. The object-detection combined with classical feature detection and unprojection could be replaced with a modern semantic mapping algorithm, such as [FS18] configured for the environment of interest. Finally, the uncertainty-aware MPC could be extended for more complex scenarios, such as multi-robot planning [STS21] or 3D navigation.

7.1.2 Chapter 3: "Learned Uncertainty Calibration for Visual Inertial Localization"

The most immediate direction could investigate how generalizable the results of Chapter 3 are. The datasets in this work consisted of several sequences all collected using the same sensor in the same environment. We suspect that the calibration described in this work can be performed once for each sensor and VIO implementation, but generalize across multiple

environments. This would require performing the covariance calibration experiment for multiple VIO systems on multiple sensor systems with enough sequences to enable deep learning. At least one dataset would have to encompass multiple environments, and all sequences need ground-truth pose. We anticipate this exercise would require a combination of existing VIO datasets and newly collected data.

Provided that the results of Chapter 3 are sufficiently generalizable, the neural network presented in this paper could be integrated into the architecture in Figures 1.1 and 2.1. Its input would be the covariance estimates from the RNN and its output would be sent to the MPC. The RNN could be trained exactly as described in Chapter 2, but the covariance predictions given to the MPC would be more accurate. As the navigation exercise in Chapter 2 only consists of moving around one box in a room that is not narrow, it may not be difficult to set up a more complex navigation exercise where the architecture in Chapter 2 fails in its current state, but then succeeds with more accurate covariance estimates.

A third direction of research would include using information from a neural network to modify an EKF in the loop in an end-to-end fashion instead of simply adjusting the covariances post-hoc. In other words, the final line of equation (3.2) would take the form:

$$\hat{P}_{k+1} = \phi_w(P_k, \hat{x}_k) \tag{7.1}$$

where ϕ_w is a recurrent network. This raises questions not only of prediction accuracy, but of stability, since the online optimization would create closed-loop dependencies.

7.1.3 Chapter 4 and 5: "Feature Tracks are not Zero-Mean Gaussian" and "Quantifying VIO Uncertainty"

The results of Chapter 4 can be used in several areas future work. The most immediate direction is to continue to use the Extended Kalman Filter, but dynamically adapt filter parameters, such as covariance estimates and the number of tracked features, to the scene. Since feature tracks are not zero-mean, covariance estimates will have to be enlarged so that feature tracks containing the extra bias are not outliers. Machine learning approaches to adapting the covariance already exist [VBB13, LOV18]. Since statistical methods are

not often desirable in safety critical systems, it is of interest to compare performance when covariance is adjusted by a learned model to when covariance is adjusted by a finite state machine. While this approach is the most immediate, it does not address the fact that it brings no convergence guarantees in a downstream state estimation process and will therefore require extensive testing for each application.

The second area of future work is to adapt existing state estimation algorithms to accommodate feature tracks that are not zero-mean Gaussian, with the results from this chapter serving as a guide for what type of errors to accommodate. It may not be possible, however, to design a filter that is both computationally tractable, guaranteed to converge, and simple enough to implement on a complex, realistic system. This motivates the third area of future work.

The third direction of future work is to adjust individual feature tracks *before* they are used by a state estimation algorithm that assumes that measurements are zero-mean Gaussian. This is the approach used for IMUs: errors in IMU measurements are primarily dependent on temperature and mechanical alignment errors, so IMU measurements are adjusted for temperature and any known mechanical misalignment before they are passed to a downstream computer. For feature tracks, the calibration table would be more complex, as it is dependent on speed, motion type, and the type of tracker used.

While Chapter 5 provides no direct input into any future experiment, its methodology is a template for evaluating visual and visual-inertial SLAM algorithms. The effects of any future work arising from Chapter 4 on state estimation performance, state uncertainty, and scale uncertainty can be quantified in simulations.

7.1.4 Chapter 6: "Scene Uncertainty of Deterministic Image Classifiers"

Chapter 6 introduced the concept of scene uncertainty, or uncertainty with respect to variation in the scene. The measurements of scene uncertainty in Chapter 6 use multiple images of the same scene. However, at test-time, an image classifier is only given *one* image x. Estimating scene uncertainty given a single image x remains an open problem.

Although Chapter 6 is focused exclusively on image classification, the concept of scene uncertainty may apply to other tasks that are more relevant to robotics, such as object detection and image segmentation. The nature of computing or estimating for those tasks may be very different from the procedure in Chapter 6. Furthermore, if the distributions of intrinsic and extrinsic variabilities and nuisances are known, such as for a robot on a factory floor, then it may be possible in some applications to use a calibration procedure to measure scene uncertainty. For example, the scene uncertainty of an IMU measurement can be safely assumed to be a normal distribution centered around the correct value. Simply knowing the distribution of measurement errors has allowed engineers to create safety-critical technologies relying on imperfect sensors. Understanding scene uncertainty of neural networks in this way may be key to incorporating imperfect neural networks into safety-critical systems to perform functions that could not be performed otherwise.

APPENDIX A

XIVO

XIVO, or "Xiaohan's Inertial-aided Visual Odometry" is a SLAM algorithm and software package based on the Extended Kalman Filter (EKF) that estimates state using measurements of angular velocity and linear acceleration from an IMU and feature tracks extracted from a stream of RGB images. XIVO was first introduced as "Corvis" in [JS11], which presented a basic observability analysis. In the original implementation of Corvis, the equations of motion contained no inputs – both the IMU measurements and feature tracks are modeled as system outputs. By 2015, Corvis had been modified so that IMU measurements would become model inputs and only the feature tracks were system outputs. This second version is the model analyzed in [HTS15]. Additional features added at that time, some of which made Corvis no longer an EKF, were One-Point RANSAC outlier rejection [CGD09], advanced feature covariance initialization, and the option to use a Huber Loss instead of the standard L2 loss in the EKF measurement update step.

When technical debt was deemed insurmountable in 2019, the Corvis software was scrapped and rewritten as XIVO by Xiaohan Fei. XIVO was then maintained, instrumented, and modified as needed for the experiments detailed in this thesis. The basic equations are still those in [HTS15] – IMU measurements are system inputs. However, not all features of Corvis were kept and there are more (optional) calibration states than those mentioned in [HTS15] and [JS11]. This Appendix aims to give the full description of the state, autocalibration state, and dynamics equations implemented in XIVO as of January 2023.

The steps implemented in XIVO are: EKF prediction, EKF measurement update, feature detection and tracking, 2D outlier rejection, 3D outlier rejection, and loop closure. A flowchart of the steps is given in Figure A.1. The software modules are listed in Figure A.2.

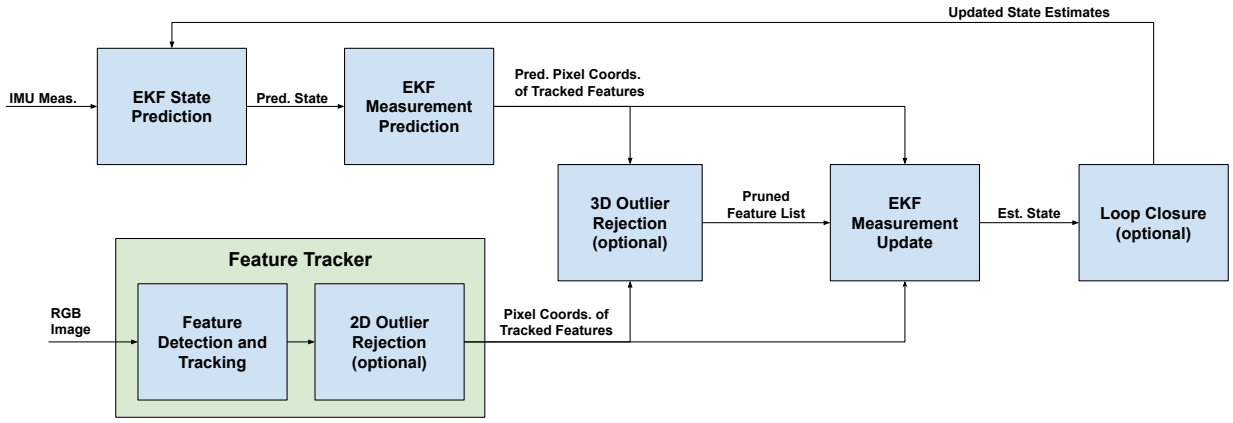


Figure A.1: **XIVO Overview.** IMU Measurements are first used to propagate the estimated state χ forward in time. After propagation of χ, XIVO then calculates predicted image locations of all features currently tracked by the Feature Tracker. In the Feature Tracker, feature tracks are extracted from RGB images using either Lucas-Kanade Sparse Optical Flow or Correspondence Matching. Feature Tracks may then be pruned for outliers using planar outlier rejection (e.g. RANSAC, LMEDS, RHO), an optional step when using sparse optical flow, but effectively a required step when using Correspondence Matching. For further rejection of outlier feature tracks, XIVO also contains implementations of the 3D outlier rejection algorithms Mahalnobis Gating and One-Point RANSAC [CGD09]. With all outliers removed, the last steps are the standard EKF measurement update and an optional loop closure.

This Appendix is a guide to understanding the components of XIVO, but is not a user's guide or software documentation.

A.1 Preliminaries

A.1.1 Notation

We use the following conventions for notation in this Appendix:

- Vectors are represented by lower-case symbols, such as x, y, and z. Individual scalar elements inside the vectors are denoted with subscripts, e.g. x_1, x_2, x_3 for $x \in \mathbb{R}^3$.

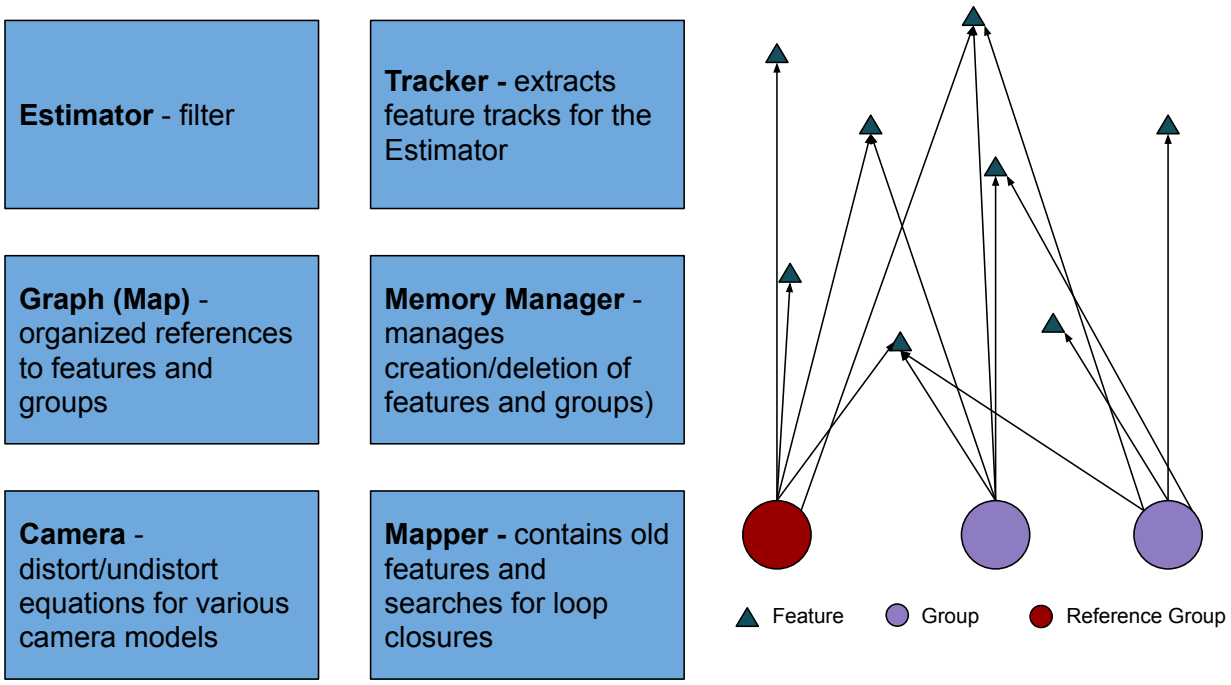


Figure A.2: **Software Objects in XIVO.** The `Estimator`, `Tracker`, `Graph`, `Memory Manager`, `Camera`, and `Mapper` are implemented as C++ Singletons. `Features` and `Groups` exist outside the Singletons. The components that require access to `Features` and `Groups` contain organized pointers.

Vector indices are 1-indexed (like in Matlab).

- Matrices are represented with upper-case symbols. All upper-case symbols are matrices, unless specifically specified to be something else below.
- Coordinated frames are designated as lower-case subscripts, e.g. x_a refers to a vector x resolved in coordinate frame a.
- An upper-case R, with or without subscripts, always refers to a rotation matrix; $R \in SO(3)$.
- An upper-case T, with or without subscripts, always refers to the translation component of a rigid body transformation; $T \in \mathbb{R}^3$
- A lower-case g pared with no subscripts always refers to the gravity vector $[0, 0, -9.8]$ $\mathrm{m/s^2}$.
- A lower-case w, with or without subscripts, always refers to a rotation vector. $R(w)$ is the rotation matrix corresponding to the rotation vector.
- A lower-case ω always refers to angular velocity.
- An upper-case V always refers to linear velocity.
- A lower-case a, when not part of a subscript, always refers to linear acceleration.
- Linear velocity, angular velocity, and linear acceleration are written with three subscripts. For example ω_{sb}^{b} is the angular velocity of the body frame with respect to the spatial frame resolved in the body frame.
- The upper-case symbols X, Y, Z with no subscripts represent the x, y, and z-coordinates of a position of a feature in $\mathbb{R}^3$.
- The symbol X with a subscript q (e.g. $X_q = [X, Y, Z]$) is a vector in $\mathbb{R}^3$ containing the position of a feature or group in coordinate frame q.
- The symbol χ represents the entire EKF-state.

- For $x \in \mathbb{R}^3$, $[x]_\times$ is the skew-symmetric matrix constructed with elements of x.
- The projection operator $\pi : \mathbb{R}^3 \to \mathbb{R}^2$ is given by:

$$\pi(x) = \begin{bmatrix} x_1/x_3 \\ x_2/x_3 \end{bmatrix}$$

- The camera intrinsics operator is denoted $\pi^c : \mathbb{R}^2 \to \mathbb{R}^2$ and depends on the camera model. π^c depends on both the camera intrinsics matrix K and any lens distortion parameters.
- A row of dots $\dots$ at the top, bottom, or side of a matrix indicates that the rest of the matrix is all zeros. For example:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ \dots & \dots & \dots \end{bmatrix}$$

A.1.2 Representation of 3D Rotations and Coordinate Transformations

Our notation is consistent with Chapter 2 of [ML94]. Consider two coordinate systems a and b and a point $x_b \in \mathbb{R}^3$, resolved in b. Then, the coordinates of the same point resolved in coordinate system a is given by:

$$x_a = R_{ab} x_b + T_{ab}. \tag{A.1}$$

From (A.1), we have the following expressions for the inverse:

$$\begin{aligned} x_b &= R_{ab}^{-1}(x_a - T_{ab}) \\ R_{ba} &= R_{ab}^{-1} = R_{ab}^T \\ T_{ba} &= -R_{ab}^{-1} T_{ab}. \end{aligned} \tag{A.2}$$

XIVO uses rotation vectors to represent 3D rotation in its state equations, since rotation vectors are the tangent space $so(3)$ of group $SO(3)$. For a rotation vector $w = [w_1, w_2, w_3]^T$,

the magnitude of w, $\theta = \|w\|$ is the magnitude, or angle in radians, and $\hat{w}$ is the axis. The rotation matrix representation of w is given by

$$R(w) = \exp\left([w]_\times\right)$$

where exp is the matrix exponential.

In order to avoid singularities and for ease of numerical integration, however, XIVO represents internal rotations using quaternions rather than rotation vectors. The Sophus library[1] is used for quaternion integration, and for conversion to rotation matrices and rotation vectors as needed.

A.1.3 Special Jacobians

Jacobian with Respect to a Rotation Vector A closed-form expression for The Jacobian of $R(w)$ is given in [GY15]. However, due to issues with numerical stability, we do not derive dynamics and measurement Jacobians using the equations in [GY15]. Instead, we substitute the first-order Taylor series approximation $R(w) \approx I + [w]_\times$ and then compute derivatives with respect to w. The identity $[w]_\times a = -[a]_\times w$ is useful in the derivatives.

Matrix Product Jacobian Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$. Then,

- $\frac{\partial AB}{\partial B}$ has dimension $mp \times np$. If the size of B is known, it is a function of the matrix A. In the texts below, this is written as a function $\frac{\partial AB}{\partial B}_{n,p}(X)$.
- $\frac{\partial AB}{\partial A}$ has dimension $mp \times mn$. If the size of A is known, it is a function of the matrix B. In the texts below, this is written as a function $\frac{\partial AB}{\partial A}_{m,n}(X)$.
- $\frac{\partial A^T}{\partial A}$ has dimension $mn \times mn$. It is a function of the matrix A. In the texts below, this is written as a function $\frac{\partial A^T}{\partial A}(X)$.
- $\frac{\partial A}{\partial A^u}$, where A_u is the upper triangle of a square matrix $A \in \mathbb{R}^{m\times m}$, has dimension $m^2 \times \frac{m(m+1)}{2}$ and is a constant. In the texts below, this is written as $\frac{\partial A}{\partial A^u}(X)$.

[1] `https://github.com/strasdat/Sophus`

A.2 Filter Equations

A.2.1 Coordinate Frames

XIVO uses the following coordinate frames:

- Spatial/world frame s – A static frame that is never moving. For issues related to observability [JS11, HTS15], it is fixed to be the position and orientation of the IMU at startup.
- Body/IMU frame b – A frame attached to the IMU. The exact direction of the axes are determined by the manufacturer and how the IMU is mounted.
- Camera frame c – A frame attached to the camera's pinhole. The Z axis points outward perpendicular to the image plane. The X axis points horizontally to the right side of the image plane. The Y axis completes the triad, i.e. the Y axis points down.
- Pixel frame p – A 2D frame whose origin is the top-left corner of the image. The projection operator and the camera intrinsics map points in the c frame with units of meters to points in the p frame with units of pixels.
- Gravity frame g – A frame that is aligned with gravity and co-located with the b frame. In this frame, gravitational acceleration has value $[0, 0, -9.8]^T$.
- Reference body frame b_r – A frame fixed at the past position and orientation of a body frame b. The EKF state χ will contain multiple values of g_{sb_r}, but no more than one instance of g_{sb_r} will appear in any equation.
- Reference camera frame c_r – A frame fixed at the past position and orientation of a camera frame c. The transformation $g_{sc_r} = g_{sb_r} \circ g_{bc}$

A.2.2 States

Ego-state: These states describe the position and orientation of the IMU relative to its initial condition. The dimension of each state is shown in parentheses:

- w_{sb} (3), T_{sb} (3) - pose of the body frame with respect to the spatial frame, i.e. $x_s = R(w_{sb})x_b + T_{sb}$.

- V_{sb}^b (3) - velocity of the body frame with respect to the spatial frame.

- b_g^b (3) - bias of the IMU's gyroscope in the body frame, i.e.

$$\bar{\omega}_{sb}^b = \omega_{sb}^b - b_g^b \tag{A.3}$$

  where ω_{sb}^b is the measured angular velocity and $\bar{\omega}_{sb}^b$ is the true angular velocity.

- b_a^b (3) - bias of the IMU's accelerometer in the body frame, i.e.

$$\bar{a}_{sb}^b = a_{sb}^b - R_{bs}R_{sg}g - b_a^b \tag{A.4}$$

  where a_{sb}^b is the measured angular velocity and $\bar{a}_{sb}^b$ is the true angular velocity.

Calibration States: These states enable autocalibration. The dimension of each state is shown in parentheses. Optional states are those that may be excluded from the state vector.

- w_{bc} (3), T_{bc} (3) - pose of the camera frame with respect to the body frame.

- w_{sg} (2) - First two elements of the orientation of the gravity vector with respect to the spatial frame. Since the rotation around the z-axis of the gravity frame is unobservable (and does not matter), we take the third element of w_{sg} to be 0.

- (Optional) t_d (1) - An estimate of the time difference between when an image is acquired and when an image is timestamped.

- (Optional) C_g (9) and C_a (6) - IMU Calibration parameters. If enabled, equations (A.3) and (A.4) become

$$\begin{aligned} \bar{\omega}_{sb}^b &= C_g\omega_{sb}^b - b_g^b \\ \bar{a}_{sb}^b &= C_a(a_{sb}^b - R_{bs}R_g g) - b_a^b \end{aligned} \tag{A.5}$$

where the matrices C_a and C_g take the form

$$C_a = \begin{bmatrix} C_{a,1} & C_{a,2} & C_{a,3} \\ 0 & C_{a,3} & C_{a,4} \\ 0 & 0 & C_{a,6} \end{bmatrix} \qquad C_g = \begin{bmatrix} C_{g,1} & C_{g,2} & C_{g,3} \\ C_{g,4} & C_{g,5} & C_{g,6} \\ C_{g,7} & C_{g,8} & C_{g,9} \end{bmatrix}. \tag{A.6}$$

An IMU with no misalignment or scale error has $C_a = C_g = I_{3\times3}$. These parameters can be found using the procedure detailed in [TPM14].

- (Optional) Intrinsic camera calibration parameters (up to 9). The exact number of camera calibration parameters depends on the chosen model and includes both projection and distortion parameters. XIVO currently supports the pinhole (4 parameters), equidistant (8 parameters), arctangent (5 parameters), and radial-tangential (9 parameters) models.

Map states:

- w_{sb_r} (3 per group), T_{sb_r} (3 per group) - pose of each group with respect to the spatial frame, i.e. the value of w_{sb} and T_{sb} at the time the group was created.

- x_{c_r} (3 per feature) - position of each feature resolved in the camera frame in a log-depth or inverse-depth representation *at the time the feature was first observed.* In other words, if the position of the feature in the camera frame at the time it was first observed is $[X, Y, Z]$, then $x_{c_r} = [X/Z, Y/Z, \log(Z)]$ or $x_{c_r} = [X/Z, Y/Z, 1/Z]$. Whether to use a log-depth or an inverse-depth representation is a compile-time parameter.

A.2.3 The Error State

Due to issues of numeric stability, we do not estimate the mean and covariance of the entire state directly. Instead, we estimate the state and the covariance of the error state. At the

end of each measurement update, the error is "absorbed" into the state. More precisely, let

- n_x be the dimension of the state
- n_u be the dimension of the system input
- n_y be the dimension of the measurements
- $\chi \in \mathbb{R}^{n_x}$ be the state and $\hat{\chi} \in \mathbb{R}^{n_x}$ be the estimated state.
- $e \in \mathbb{R}^{n_x}$ be the error state
- $u \in \mathbb{R}^{n_u}$ be the system input, $\bar{u}$ be the nominal, noise-free input
- $y \in \mathbb{R}^{n_y}$ be the measurements
- P be the covariance of e
- $\mu \sim \mathcal{N}(0, Q)$ be additive noise to u
- $\eta \sim \mathcal{N}(0, R)$ be sensor noise

The dynamics and measurement model are given by:

$$\begin{aligned} \dot{\chi} &= f(\chi, e, u, \mu) \\ y &= h(\chi, e) + \eta. \end{aligned} \tag{A.7}$$

Note that even though the error state is additive to the state and noise terms are additive to u and y, they can be incorporated nonlinearly into $f(\chi, e, u, \mu)$ and $h(\chi, e)$. During the prediction step of the EKF, we integrate the continuous dynamics one timestep:

$$\begin{aligned} \dot{\hat{\chi}} &= f(\hat{\chi}, \bar{u}) \\ \dot{P} &= FP + PF^T + GQG^T \\ \dot{e} &= 0 \end{aligned} \tag{A.8}$$

where $\hat{\chi}$ is the current estimated value of χ, $\bar{u}$ is the nominal value of the input, $F = \frac{\partial f}{\partial e}|_{e=0,\chi=\hat{\chi}}$ and $G = \frac{\partial f}{\partial u}|_{u=\bar{u},\mu=0}$. The measurement step then becomes

$$\begin{aligned} K &= PH^T(HPH^T + R)^{-1} \\ e &= K(y - h(\hat{\chi}, e)) \\ P &= (I - KH)P(I - KH)^T + KRK^T \end{aligned} \tag{A.9}$$

where $H = \frac{\partial h}{\partial e}|_{e=0,\chi=\hat{\chi}}$, y is the measurement, and K is the Kalman gain.

Finally, after each measurement step, we reset the error state:

$$\begin{aligned} \hat{\chi} &\leftarrow \hat{\chi} + e \\ e &\leftarrow 0. \end{aligned} \tag{A.10}$$

A.2.4 Nominal Equations of Motion (EKF State Prediction)

Let t_0 and t_f be two timesteps at which IMU measurements are received. Since IMUs operate very fast, we will assume that the average value of the two measurements is the constant measured velocity and linear acceleration during this time. Denote these values as ω^b_{sb} and a^b_{sb}. These are the system inputs.

Adjusting for gravity, bias, and IMU calibration states,

$$\begin{aligned} \bar{\omega}^b_{sb} &= C_g\omega^b_{sb} - b^b_b \\ \bar{a}^b_{sb} &= C_a(a^b_{sb} - R_{bs}R_g g) - b^b_a \end{aligned} \tag{A.11}$$

Time derivatives of components of the state vector are then:

$$\begin{aligned} \dot{R}_{sb} &= R_{sb}[\bar{\omega}^b_{sb}]_\times \\ \dot{T}_{sb} &= R_{sb}V^b_{sb} \\ \dot{V}^b_{sb} &= \bar{a}^b_{sb} \end{aligned} \tag{A.12}$$

For issues related to observability, we enforce $R_{sb}(0) = I$ and $T_{sb}(0) = 0$ by holding one group in the map fixed (see A.3.1). All other states (calibration and map states) are static and therefore have time derivatives equal to 0.

A.2.5 Nominal Measurement Model (EKF Measurement Prediction)

For each tracked feature, the low-level feature tracking measures the location of the feature in pixels $x_p = (x_{pix}, y_{pix})$. Let R_{sb_r} and T_{sb_r} be the pose of the feature's associated group. Then, the nonlinear measurement equation is:

$$x_p = \pi^c \left(\pi \left(X_c \right) \right) \tag{A.13}$$

where

$$X_c = R_{bc}^T (R_{sb}^T (X_s - T_{sb}) - T_{bc})$$
$$X_s = R_{sb_r} (R_{bc} X_{c_r} + T_{bc}) + T_{sb_r}$$

A.2.5.1 Camera Models

XIVO supports the following implementations of camera distortion models π^c. In the below text, let $x_c^{2d} = [X/Z, Y/Z]^T = [\bar{x}, \bar{y}]^T$

Pinhole (no distortion). Parameters: f_x, f_y, c_x, c_y

$$\begin{bmatrix} x_{pix} \\ y_{pix} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{x} \\ \bar{y} \\ 1 \end{bmatrix} \tag{A.14}$$

Equidistant. Parameters: $f_x, f_y, c_x, c_y, k_0, k_1, k_2, k_3$

Let $\theta = \arctan\left(\|x_c^{2d}\|\right)$ and $\phi = \arctan(\bar{y}, \bar{x})$. Then the pixel coordinates are:

$$\begin{aligned} x_{pix} &= f_x r \cos(\phi) + c_x \\ y_{pix} &= f_y r \sin(\phi) + c_y \end{aligned} \tag{A.15}$$

where

$$r = \theta + k_0\theta^3 + k_1\theta^5 + k_2\theta^7 + k_3\theta^9. \tag{A.16}$$

Radial-Tangential. Parameters: $f_x, f_y, c_x, c_y, p_1, p_2, k_1, k_2, k_3$ Let $r = \|x_c^{2d}\|$. Then,

$$\begin{aligned} x_{pix} &= c_x + f_x \left(2p_1\bar{x}\bar{y} + p_2(2\bar{x}^2 + r^2) + \bar{x}(1 + k_1r^2 + k_2r^4 + k_3r^6)\right) \\ y_{pix} &= c_y + f_y \left(2p_2\bar{x}\bar{y} + p_1(2\bar{y}^2 + r^2) + \bar{y}(1 + k_1r^2 + k_2r^4 + k_3r^6)\right). \end{aligned} \tag{A.17}$$

Arctangent. Parameters: f_x, f_y, c_x, c_y, w

Let

$$\begin{aligned} w_2 &= 2\tan\left(\frac{w}{2}\right) \\ f &= \frac{1}{w}\frac{\arctan w_2\|x_c^{2d}\|}{\|x_c^{2d}\|}. \end{aligned} \tag{A.18}$$

Then,

$$\begin{aligned} x_{pix} &= f_x f\bar{x} + c_x \\ y_{pix} &= f_y f\bar{y} + c_y. \end{aligned} \tag{A.19}$$

A.2.6 Incorporating the Error State and Noise into Nominal Equations

Let $\tilde{x}$ be a perturbation on variable x. Equations of motion that include the error state can be made by substituting the following perturbations into all equations in Sections A.2.4 and A.2.5.

- $w_{sb} \leftarrow w_{sb} + \tilde{w}_{sb}$ and $R_{sb} \leftarrow R_{sb}R(\tilde{w}_{sb})$
- $T_{bc} \leftarrow T_{bc} + \tilde{T}_{bc}$
- $V_{sb}^b \leftarrow V_{sb}^b + \tilde{V}_{sb}^b$
- $b_a \leftarrow b_a + \tilde{b}_a$
- $b_g \leftarrow b_g + \tilde{b}_g$
- $w_{bc} \leftarrow w_{bc} + \tilde{w}_{bc}$ and $R_{bc} \leftarrow R_{bc}R(\tilde{w}_{bc})$
- $T_{bc} \leftarrow T_{bc} + \tilde{T}_{bc}$
- $w_g \leftarrow w_g + \tilde{w}_g$ and $R_g \leftarrow R_gR(\tilde{w}_g)$
- **(Optional) IMU Calibration:** $C_g \leftarrow C_g + \tilde{C}_g$ and $C_a \leftarrow \tilde{C}_a$

- **(Optional) Temporal Calibration:** $R_{sb}R(\tilde{w}_{sb}) \leftarrow R_{sb}R(\tilde{w}_{sb})R(\delta_{rot})$ and $T_{sb}+\tilde{T}_{sb} \leftarrow T_{sb}+\tilde{T}_{sb}+\delta_T$. If using online IMU calibration, then

$$\begin{aligned} \delta_t &= V_{sb}^b(t_d + \tilde{t}_d) \\ \delta_{rot} &= \bar{\omega}\tilde{t}_d + (\tilde{C}_g\omega - \tilde{b}_g)(t_d + \tilde{t}_d). \end{aligned} \tag{A.20}$$

  Otherwise, the perturbation is

$$\begin{aligned} \delta_t &= V_{sb}^b(t_d + \tilde{t}_d) \\ \delta_{rot} &= \bar{\omega}\tilde{t}_d \end{aligned} \tag{A.21}$$

- **(Optional) Camera Calibration:** all additive perturbations
- **Group States:** $w_{sb_r} \leftarrow \tilde{w}_{sb_r}$, $R_{sb_r} \leftarrow R_{sb_r}R(\tilde{w}_{sb_r})$, $T_{sb_r} \leftarrow \tilde{T}_{sb_r}$
- **Feature States:** $x_{c_r} \leftarrow x_{c_r} + \tilde{x}_{c_r}$
- **IMU Noise:** $\omega \leftarrow \omega + n_{imu}$

A.2.7 Covariance Matrix Prediction

The dynamics of the covariance matrix are given by the Lyapunov equation

$$\dot{P} = FP + PF^T + GQ_{imu}G^T \tag{A.22}$$

where $F = \frac{\partial \dot{\chi}}{\partial e}$ is the Jacobian of (A.12) with respect to the error state e and $G = \frac{\partial \dot{\chi}}{\partial n_{imu}}$ is the Jacobian of (A.12) with respect to modeled IMU noise n_{imu}.

A.2.8 Augmenting the State and Covariance Matrix with new Features and Groups

The number of map states (tracked features and reference groups) within the EKF state at any given time is variable, but XIVO's implementation does not vary the size of the state χ, error state e, or covariance matrix $\hat{P}$. All three are allocated enough space for the maximum number of features and groups at initialization to limit the number of system calls that XIVO

will make. The maximum number of tracked features and reference groups within the EKF state are compile-time parameters.

Each tracked feature and reference group is assigned a “slot" in the error vector e and covariance P. Unused slots have zero error, zero covariance, and zero correlation with any other part of the EKF state. Therefore, they will not affect any other part of the state during EKF measurement update, but will just make the measurement update more expensive. Removing a feature or group from the state is achieved by zeroing out the relevant rows and columns of the covariance matrix and error vector.

When adding a new reference group to the EKF state, the group is initialized with the current value of R_{sb} and T_{sb}. Its 6×6 covariance block is also initialized with the current covariance of R_{sb} and T_{sb}. When adding a new feature to the EKF state, its value and covariance are initialized with the values and 3×3 covariance from the subfilter performing its depth initialization (see A.5). Neither new features nor groups are initialized with any cross-correlation with other states.

A.3 Mapping: Management of Features and Groups

A.3.1 Killing Two Birds with One Stone: A Note on Observability and Feature Position Uncertainty.

Detected features, indexed i, are 3D points in space. The map could, theoretically, be represented as a collection of points $X_s^i \in \mathbb{R}^3$ rather than in the groups noted above. The representation we use, however, simultaneously addresses two problems:

1. Global gauge, the initial value of g_{sb}, is unobservable and must be fixed – fixing it requires removing six degrees of freedom from the state while ensuring that values of g_{sb} may be updated. An Extended Kalman Filter does not naturally provide a means of enforcing an initial condition; it only provides a means of fixing a portion of the state of a particular value.

2. The uncertainty of a feature's position is not well-represented by a ball or spheroid in

$\mathbb{R}^3$. Because features, 3D points in space, are detected and tracked through 2D images, when their position is represented in the camera frame c, it is much easier to estimate their direction (X and Y values), than their depth (Z value). This observation is only true when its position is represented in a c frame where it is visible, such as frame c_r, or the frame at which each feature was first detected; if its location were estimated in the s frame the uncertainty would be distributed in all three cardinal directions.

The location of the camera at which a feature was first detected, however, is something that is not exactly known. Since past c frames are not exactly known, estimating the location of a feature requires estimating the location of the past c frame as well. This means that the location of a feature i is represented by the following quantities:

- $g_{sb_r} \in \mathrm{SE}(3)$ - the location of the body coordinate frame when the feature was first detected
- $g_{bc} \in \mathrm{SE}(3)$ - the camera-IMU extrinsic calibration, which is already estimated as part of the state
- $X^i_{c_r} \in \mathbb{R}^3$ - the location of the feature in the camera frame when it was first detected

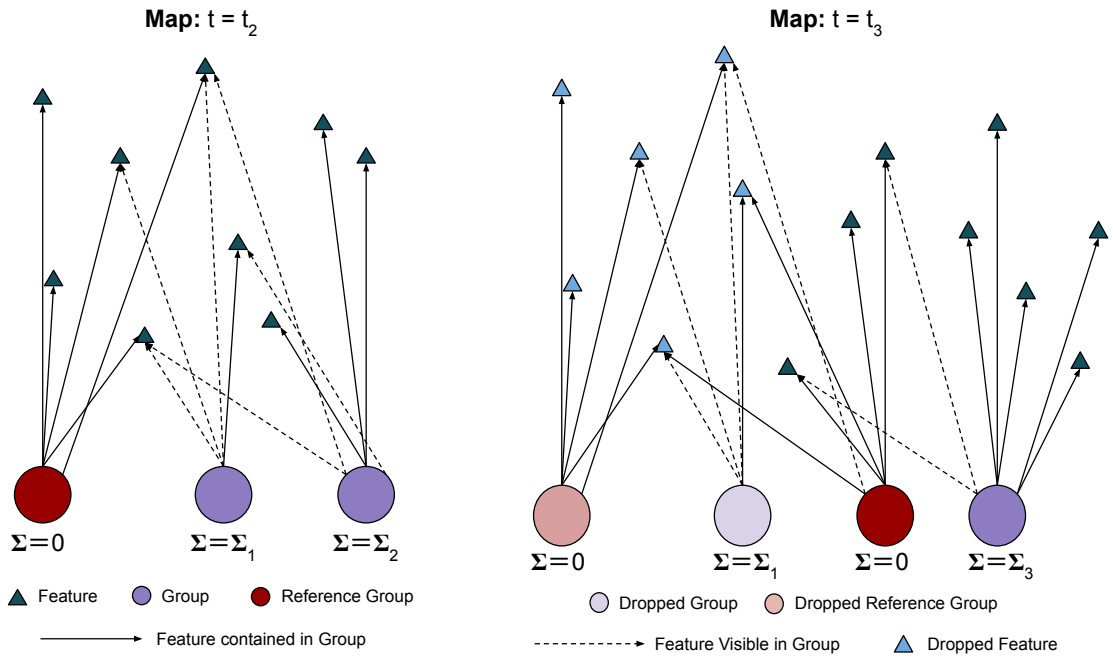
This representation requires 9 additional parameters per feature (g_{sb_r}, X_{c_r}) instead of 3, which is both computationally inefficient and an overparameterization. Since a feature is a point in 3 dimensions, there are an infinite number of ways to make 9 parameters satisfy the observed feature locations during state estimation. This computational inefficiency and overparameterization can be solved by having multiple features, first detected in the same frame, share the 6 parameters in g_{sb_r}. Then, the number of states used to represent N features is $6 + 3N$ instead of $9N$. As the total number of degrees of freedom is $3N$, this is still an overparameterization. However, if the X and Y values of three non-collinear $X^i_{c_r}$ that use the same g_{sb_r} are fixed, then the total number of degrees of freedom in the group is the correct value of $3N$. The three features that are held fixed in each group are called *gauge features*. Groups must therefore "own" at least three features.

With this representation of the map, the enforcement of a global gauge transformation may be accomplished by holding one of the g_{sb_r} fixed. At initialization, XIVO will always fix the g_{sb_r} corresponding to its initial position because it enforces that the initial condition $g_{sb}(0) = (I, 0)$ and because there will be many possible features to choose as the gauge feature. This is pictured in the left portion of Figure A.3.

A.3.2 Feature and Group Management

Since the state of the EKF vector has size $O(N)$ and the measurement update step has time complexity $O(N^3)$ and still must run in real-time, it is impractical to keep all visible features in the state. On the other hand, the complexity of XIVO's feature tracker is only $O(N)$ for the Lucas-Kanade Tracker and $O(N^2)$ for the Correspondence Tracker. Furthermore, observing 3D features in 2D means that a depth value must be estimated (or guessed) before it is added to the EKF state vector; more details on feature depth initialization are given in Section A.5. Enforcing observability also means that groups within the EKF state must contain at least three features. These facts lead to the following design choices:

- The number of features, and groups of features, in the EKF state is limited to the compile-time parameters `EKF_MAX_FEATURES` and `EKF_MAX_GROUPS`.
- The number of features in the feature tracker is limited to `tracker.max_features`, a run-time parameter that is larger than `EKF_MAX_FEATURES`.
- Features tracked by the feature tracker, but are not part of the EKF state, are in a depth initialization mode, in which a $O(1)$ complexity filter estimates its depth. Features that have been in a depth initialization mode for at least `subfilter.ready_steps`, a run-time parameter, are deemed `READY`.
- When features are added to the EKF state, they are added either to an existing group already in the EKF state, or in groups of at least three features. When a group is added, three gauge features are chosen. When adding groups to the EKF state, the selection process will add the group(s) with the most number of `READY` features.



(a) Initial Map

(b) After Reference Group Update

Figure A.3: **Map.** XIVO's map can be visualized as a graph with two types of nodes (Features and Groups) and two types of edges (Ownership and Visibility). A feature f_i is *owned* by a group $g_{sb_r} \in \mathrm{SE}(3)$, when its estimated position in the spatial frame X_s^i is calculated using the parameters of group g_{sb_r}. A feature f_i may also be *visible* in other groups, or past values of R_{sb} and T_{sb} in the map. Although features are initially owned by the group where it is first detected, its state may be parameterized by any group in which it is visible. In order to enforce a global gauge, the covariance of a single group containing at least three features must be fixed at all times – the first reference group is always the initial position (left figure). A group is *dropped* when fewer than three features remain visible. When a reference group is dropped, a new group is chosen as the reference and its covariance is fixed (right figure).

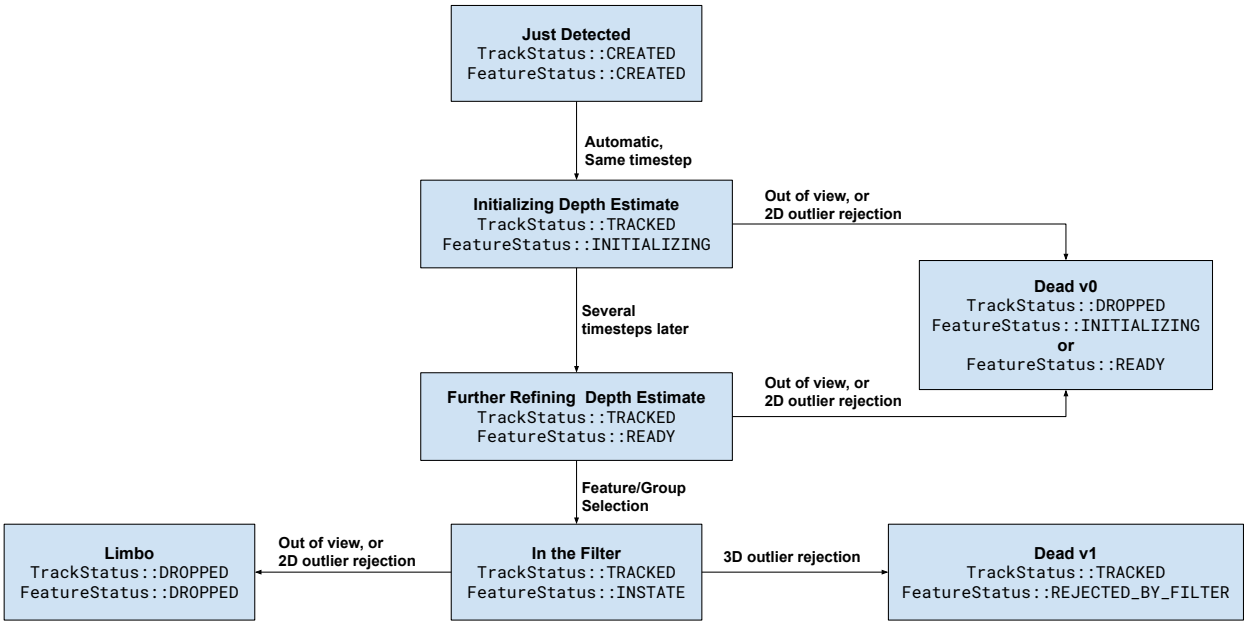


Figure A.4: **Life of a XIVO Feature.** A feature has two state variables, one maintained by the feature tracker, `TrackStatus`, the other maintained by the EKF, `FeatureStatus`. Details about transitions are described in the main text of Section A.3.2.

- Individual features are removed from the EKF state when they fall out-of-view or when they are flagged by outlier rejection.
- When a gauge feature is removed from the EKF state and its group has more than three features remaining, a new gauge feature is chosen from the remaining features.
- When a group has fewer than three features remaining, the entire group is removed from the EKF state. XIVO will attempt to transfer the features to other groups within the EKF state by recalculating x_{c_r} for a separate group.

The possible lifecycles a feature may have is captured by in Figure A.4. The logic for adding features into the EKF state is given in Figure A.5.

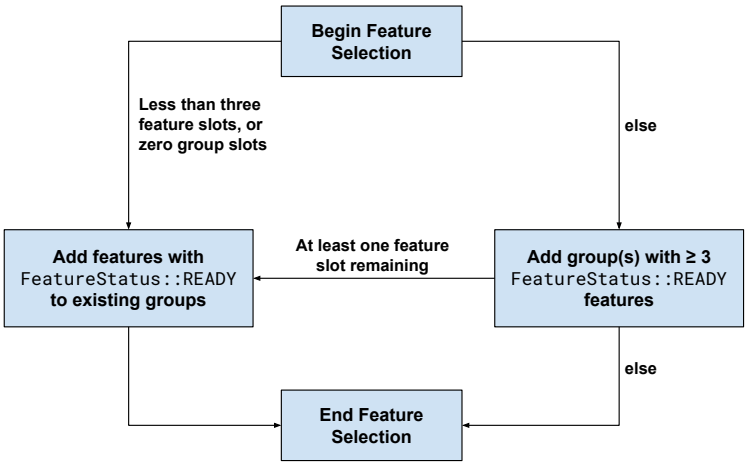


Figure A.5: **Adding Features into the EKF.** At each timestep, XIVO will attempt to add features into the EKF state. It will always try to add at least one new group before adding features to existing groups.

A.4 Jacobians

The order of elements in the state vector χ are:

$$\chi = \begin{bmatrix} \chi_{\text{IMU}} & \chi_{\text{calib}} & \chi_{\text{opt}} & \chi_{\text{group}} & \chi_{\text{features}} \end{bmatrix} \tag{A.23}$$

where

$$\chi_{\text{IMU}} = \begin{bmatrix} w_{sb} & T_{sb} & V_{sb} & b_g^b & b_a^b \end{bmatrix} \tag{A.24}$$

$$\chi_{\text{calib}} = \begin{bmatrix} w_{bc} & T_{bc} & w_{sg} \end{bmatrix} \tag{A.25}$$

$$\chi_{\text{opt}} = \begin{bmatrix} t_d & C_g & C_a & \theta \end{bmatrix} \tag{A.26}$$

$$\chi_{\text{group}} = \begin{bmatrix} w_{sb_r,1} & T_{sb_r,1} & w_{sb_r,2} & T_{sb_r,2} & \dots & w_{sb_r,n_g} & T_{sb_r,n_g} \end{bmatrix} \tag{A.27}$$

$$\chi_{\text{features}} = \begin{bmatrix} x_{c_r}^1 & x_{c_r}^2 & \dots & x_{c_r}^{n_f} \end{bmatrix} \tag{A.28}$$

n_g is the number of feature groups (see Section A.3.1) and n_f is the number of features. χ_{group} will have size $6n_g$ and χ_{features} will have size $3n_f$. Let n_{opt} be the dimension of χ_{opt}. The value of n_{opt} will depend on the number of optional features used. The dimension of χ is $n_\chi = 23 + 6n_g + 3n_f + n_{\text{opt}}$.

A.4.1 Our approach to deriving approximate Jacobians.

For "block-Jacobians" (Jacobians of vectors with respect to vectors), we incorporate the error state and noise into nominal equations of motion and measurement update equations as in Section A.2.6. Then we algebraically isolate the error state variables. The final results are documented in this section.

(Optional) Online Temporal and IMU Calibration If using online temporal and IMU calibration, we need additional Jacobians with respect to error states $\tilde{t}_d$, $\tilde{C}_g$, and $\tilde{C}_a$.

If we are tracking with online temporal calibration and online IMU calibration, then R_{sb} is instead[2]

$$R_{sb} = \bar{R}_{sb} R(\tilde{w}_{sb}) R(\delta_{rot}) \tag{A.29}$$

where

$$\delta_{rot} = (\bar{C}_g \omega_m - \bar{b}_g)\tilde{t}_d + (\tilde{C}_g \omega_m - \tilde{b}_g)(\bar{t}_d + \tilde{t}_d) \tag{A.30}$$

A.4.2 The Matrix F (eqs. (A.8), (A.22))

The matrix $F \in \mathbb{R}^{n_\chi \times n_\chi}$ has form:

$$\begin{bmatrix} \frac{\partial \dot{\chi}}{\partial \chi_{\text{IMU}}} & \frac{\partial \dot{\chi}}{\partial \chi_{\text{calib}}} & \frac{\partial \dot{\chi}}{\partial \chi_{\text{opt}}} & \mathbf{0}_{n_\chi \times 6n_g} & \mathbf{0}_{n_\chi \times 3n_f} \end{bmatrix} \tag{A.31}$$

[2]Without online IMU calibration, $\bar{C}_g = I$ and $\tilde{C}_g = 0$.

Values of individual matrices are:

$$\frac{\partial \dot{\chi}}{\partial \chi_{\text{IMU}}} = \begin{bmatrix} -[\bar{\omega}^b_{sb}]_\times & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ -R_{sb}[\bar{a}^b_{sb}]_\times & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -R_{sb} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \tag{A.32}$$

$$\frac{\partial \dot{\chi}}{\partial \chi_{\text{calib}}} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times2} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times2} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & (-R_{sb}[g]_\times)_{\text{first two cols}} \\ \cdots & \cdots & \cdots \end{bmatrix} \tag{A.33}$$

$$\frac{\partial \dot{\chi}}{\partial \chi_{\text{opt}}} = \begin{bmatrix} \mathbf{0}_{3\times1} & \frac{\partial \dot{R}_{sb}}{\partial C_g} & \mathbf{0}_{3\times6} & \mathbf{0}_{3\times n_\theta} \\ \mathbf{0}_{3\times1} & \mathbf{0}_{3\times9} & \mathbf{0}_{3\times6} & \mathbf{0}_{3\times n_\theta} \\ \mathbf{0}_{3\times1} & \mathbf{0}_{3\times9} & \frac{\partial \dot{V}_{sb}}{\partial C_a} & \mathbf{0}_{3\times n_\theta} \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix} \tag{A.34}$$

where

$$\frac{\partial \dot{R}_{sb}}{\partial C_g} = \begin{bmatrix} (\omega^b_{sb})^T & \mathbf{0}_{1\times3} & \mathbf{0}1 \times 3 \\ \mathbf{0}_{1\times3} & (\omega^b_{sb})^T & \mathbf{0}_{1\times3} \\ \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & (\omega^b_{sb})^T \end{bmatrix} \in \mathbb{R}^{3\times9} \tag{A.35}$$

$$\frac{\partial \dot{V}_{sb}}{\partial C_a} = \left(\frac{\partial AB}{\partial A}_{3\times3}(a^b_{sb})\right)\left(\frac{\partial AB}{\partial B}_{3\times3}(R_{sb})\right)\left(\frac{\partial A}{\partial A^u}\right) \in \mathbb{R}^{3\times6} \tag{A.36}$$

A.4.3 The Matrix G (eqs. (A.8), (A.22))

Let $u = \begin{bmatrix} n_g & n_a & n_{b_g} & n_{b_a} \end{bmatrix}^T$ be the input vector. Then, the matrix G has dimension $n_{imu} \times n_\chi$ and has form:

$$G = \begin{bmatrix} -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & R_{sb} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \\ \dots & \dots & \dots & \dots \end{bmatrix} \tag{A.37}$$

All rows of G not explicitly noted are completely zero.

A.4.4 The Matrix H (eq. (A.9))

The dimension of H is $2n_f \times n_\chi$, consisting of n_f blocks with dimension $2 \times n_\chi$. Each block is the Jacobian of predicted pixel coordinates x_p of a feature j with respect to the error states $\tilde{\chi}$ at $\tilde{\chi} = 0$. Assume that feature j is owned by group i. Then, each two-row block is given by:

$$\frac{\partial x_p}{\partial \tilde{\chi}}|_{\tilde{\chi}=0} = \frac{\partial \pi^c(x_c)}{\partial \theta} + \frac{\partial \pi^c(x_c)}{\partial x_c} \cdot \frac{\partial \pi(X_c)}{\partial X_c} \cdot \frac{\partial X_c}{\partial \tilde{\chi}}|_{\tilde{\chi}=0}. \tag{A.38}$$

The quantity $\frac{\partial \pi^c(x_c)}{\partial \theta}$ is equal to zero if online camera calibration is not enabled.

- $\frac{\partial \pi^c(x_c)}{\partial x_c} \in \mathbb{R}^{2\times3}$ depends on the camera model used. For the pinhole projection model,

$$\frac{\partial \pi^c(x_c)}{\partial x_c} = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \end{bmatrix}. \tag{A.39}$$

  Expressions for the other camera models are more complex. They were automatically differentiated and autocoded from equations given in Section A.2.5.1 using the MATLAB Symbolic Toolbox.

- If using log-z projection,

$$\frac{\partial \pi(X_c)}{\partial X_c} = \begin{bmatrix} 1/Z & 0 & -X/Z^2 \\ 0 & 1/Z & -Y/Z^2 \\ 0 & 0 & 1/Z \end{bmatrix} \tag{A.40}$$

  for $X_c = [X, Y, Z]^T$.

- If using inverse-z projection,

$$\frac{\partial \pi(X_c)}{\partial X_c} = \begin{bmatrix} 1/Z & 0 & -X/Z^2 \\ 0 & 1/Z & -Y/Z^2 \\ 0 & 0 & -1/Z^2 \end{bmatrix} \tag{A.41}$$

- $\frac{\partial X_c}{\partial \tilde{\chi}}$ is a matrix with dimension $3 \times n_\chi$ with form (the three "rows" below are really one row):

$$\frac{\partial X_c}{\partial \tilde{\chi}} = \begin{bmatrix} \frac{\partial X_c}{\partial \tilde{w}_{sb}} & \frac{\partial X_c}{\partial \tilde{T}_{sb}} & \mathbf{0}_{3\times 3} & \frac{\partial X_c}{\partial b_g} & \mathbf{0}_{3\times 3} & \frac{\partial X_c}{\partial \tilde{w}_{bc}} & \frac{\partial X_c}{\partial \tilde{T}_{bc}} \\ \mathbf{0}_{2\times n_\chi} & \frac{\partial X_c}{\partial t_d} & \frac{\partial X_c}{\partial \tilde{C}_g} & \mathbf{0}_{6\times n_\chi} & \mathbf{0}_{n_\theta \times n_\chi} & \mathbf{0}_{6(i-1)\times n_\chi} & \frac{\partial X_c}{\partial \tilde{w}_{sb_r}} \\ \frac{\partial X_c}{\partial \tilde{T}_{sb_r}} & \mathbf{0}_{6(n_g-i)\times n_\chi} & \mathbf{0}_{3(j-1)\times n_\chi} & \frac{\partial X_c}{\partial x_{c_r}} & \mathbf{0}_{3(n_f-j)\times n_\chi} & & \end{bmatrix} \tag{A.42}$$

(Approximate) expressions within $\frac{\partial X_c}{\partial \tilde{\chi}}$ that are always present are given by:

$$\frac{\partial X_c}{\partial \tilde{T}_{sb_r}} = -R_{bc}^T R_{sb}^T \tag{A.43}$$

$$\frac{\partial X_c}{\partial \tilde{T}_{bc}} = -R_{bc}^T + R_{bc}^T R_{sb}^T R_{sb_r} \tag{A.44}$$

$$\frac{\partial X_c}{\partial \tilde{T}_{sb_r}} = R_{bc}^T R_{sb}^T \tag{A.45}$$

$$\frac{\partial X_c}{\partial \tilde{w}_{sb}} = R_{bc}^T [R_{sb}^T (X_s - T_{sb})]_\times \tag{A.46}$$

$$\frac{\partial X_c}{\partial \tilde{w}_{sb_r}} = -R_{bc}^T R_{sb}^T R_{sb_r} [R_{bc} X_c + T_{bc}]_\times \tag{A.47}$$

$$\frac{\partial X_c}{\partial \tilde{w}_{bc}} = [X_c]_\times - R_{bc}^T R_{sb}^T R_{sb_r} R_{bc} [X_{c_r}]_\times \tag{A.48}$$

$$\frac{\partial X_c}{\partial x_{c_r}} = R_{bc}^T R_{sb_r} R_{bc} \frac{\partial X_{c_r}}{\partial x_{c_r}} \tag{A.49}$$

If using the log-z projection, $\frac{\partial X_{c_r}}{\partial x_{c_r}} \in \mathbb{R}^{3\times 3}$ is given by

$$\frac{\partial X_{c_r}}{\partial x_{c_r}} = \begin{bmatrix} \exp(Z) & 0 & X\exp(Z) \\ 0 & \exp(Z) & Y\exp(Z) \\ 0 & 0 & \exp(Z) \end{bmatrix} \tag{A.50}$$

where $x_{c_r} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$. If using the inverse-z projection, $\frac{\partial X_{c_r}}{\partial x_{c_r}} \in \mathbb{R}^{3\times 3}$ is instead.

$$\frac{\partial X_{c_r}}{\partial x_{c_r}} = \begin{bmatrix} 1/Z & 0 & X/Z \\ 0 & 1/Z & Y/Z \\ 0 & 0 & -1/Z^2 \end{bmatrix}. \tag{A.51}$$

(Optional) Online Camera Calibration. If online camera calibration is enabled, the quantity $\frac{\partial \pi^c(x_c)}{\partial \theta} \in \mathbb{R}^{2\times n_\theta}$ is nonzero. (The matrix is a sub-block of $\frac{\partial x_p}{\partial \tilde{\chi}}$.) For the pinhole camera model with $\theta = [f_x, f_y, c_x, c_y]^T$, this quantity is:

$$\frac{\partial \pi^c(x_c)}{\partial \theta} = \begin{bmatrix} X & 0 & 1 & 0 \\ 0 & Y & 0 & 1 \end{bmatrix} \tag{A.52}$$

where $x_c = [X, Y, Z]^T$.

For other camera models, the expressions are complex and were autocoded from the equations in Section A.2.5.1 using MATLAB Symbolic Toolbox.

(Optional) Online Temporal Calibration If online temporal calibration is enabled, then the following Jacobians are nonzero:

$$\frac{\partial X_c}{\partial t_d} = -R_{bc}^T[\bar{\omega}_{sb}^b]_\times X_b + R_{sb}^T V_{sb} \tag{A.53}$$

$$\frac{\partial X_c}{\partial b_g} = -\frac{\partial AB}{\partial B}_{3,1}(R_{bc}^T[X_b]_\times t_d) \tag{A.54}$$

(Optional) Online IMU Calibration with Online Temporal Calibration. If online IMU calibration and online temporal calibration are both enabled, then the following Jacobian is nonzero and takes the form:

$$\frac{\partial X_c}{\partial C_g} = \frac{\partial AB}{\partial B}_{3,1}(R_{bc}^T[X_b]_\times t_d) \cdot \frac{\partial C_g \omega_{sb}^b}{\partial \omega_{sb}^b} \tag{A.55}$$

where

$$\frac{\partial C_g \omega_{sb}^b}{\partial \omega_{sb}^b} = \begin{bmatrix} (\omega_{sb}^b)^T & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & (\omega_{sb}^b)^T & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & (\omega_{sb}^b)^T \end{bmatrix} \in \mathbb{R}^{3\times9} \tag{A.56}$$

(Optional) Online IMU Calibration with no Online Temporal Calibration. If online IMU calibration, but not online temporal calibration, is enabled, then the measurement equation is not dependent on C_a or C_g.

A.5 Feature Depth Initialization

In Section A.3.1, it was noted that since features are 3D points observed in only two dimensions, that when the position of a feature is estimated in the camera frame at the time it was first detected, there is a lot more uncertainty in its depth than the X and Y directions. In the camera frame at the time it was detected, the X and Y values can be observed instantly, but the depth can only be observed, and initialized, over time.

Before a feature can be added into the main EKF state, its depth value must be initialized to some value. An inaccurate initialized value will adversely affect the EKF's estimate of the current state g_{sb}. This section describes the algorithms used to initialize a depth estimate. A flowchart of all possibilities is pictured in Figure A.6.

A.5.1 Subfilter Equations

The main component used to estimate feature depth is a subfilter, an incomplete Extended Kalman Filter. Unlike the main state vector, which contains multiple features, each feature has its own subfilter; subfiltering N features therefore has a computational complexity of $O(N)$.

The state of the subfilter is $x_{c_r} = [X/Z, Y/Z, \log(Z)]^T$ with constant dynamics $\dot{x}_{c_r} = 0$. The measurement equation is:

$$x_p = \pi^c \left(\pi \left(g_{bc}^{-1}(t) \circ g_{sb}^{-1}(t) \circ g_{sb_r} \circ g_{bc}(t) \circ \pi^{-1}(x_{c_r}) \right) \right) \tag{A.57}$$

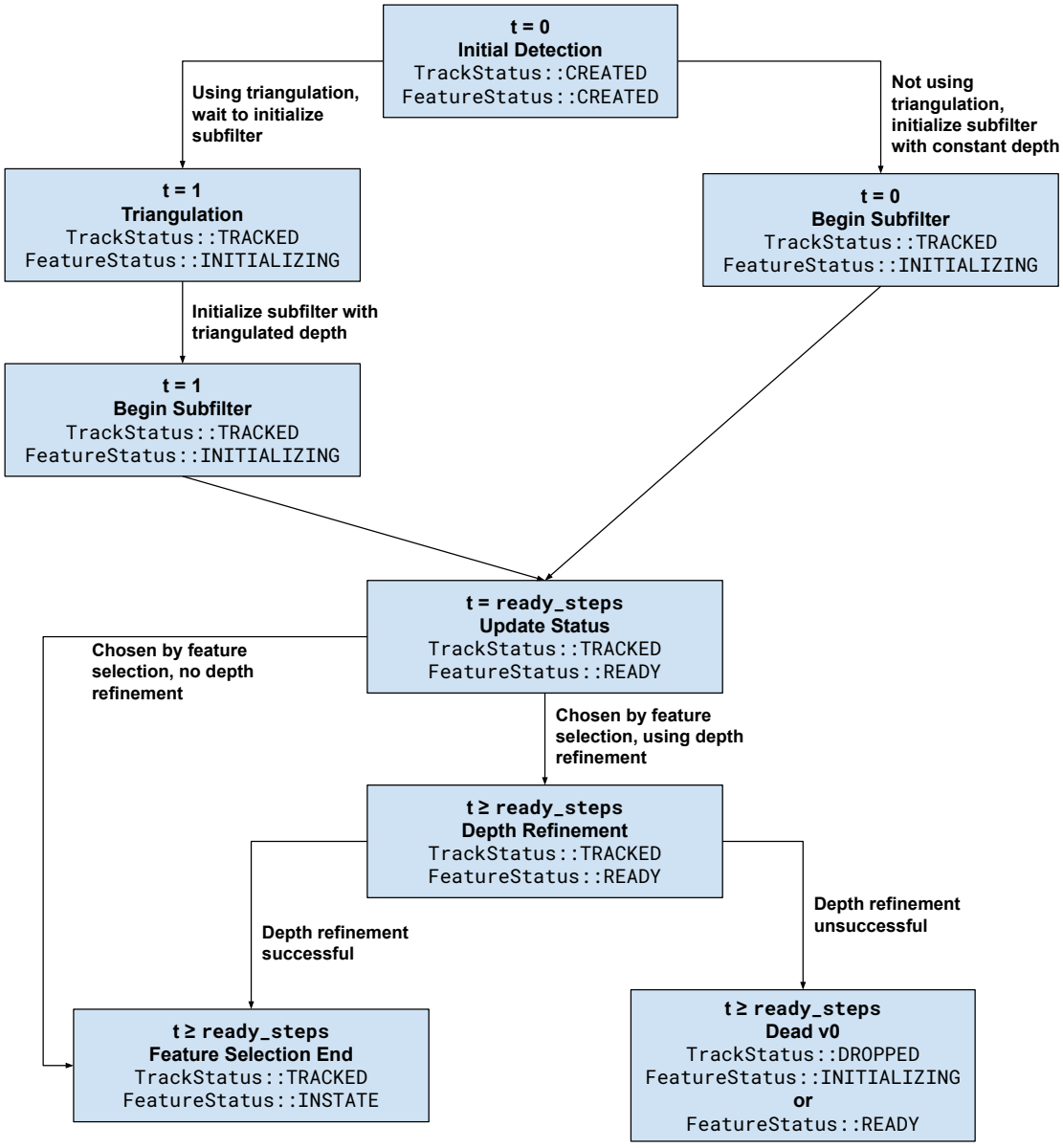


Figure A.6: **XIVO's Feature Depth Initialization Process.** This flowchart illustrates the states of a feature during the depth initialization process. `TrackStatus` and `FeatureStatus` are the same as those used in Figure A.4. The main variables affecting the process are whether or not two-view triangulation is performed, or whether or not depth refinement is performed.

where g_{bc} and g_{sb} are read from the main EKF state at each timestep and used as constants in the subfilter update. If other features sharing the same group parameters g_{sb_r} are currently in the main EKF state, then g_{sb_r} is also read from the EKF state. Otherwise, g_{sb_r} is a constant, initialized to the value of g_{sb} at time the feature was first detected. Equation (A.57) is identical to the main EKF measurement update equation, except that it only estimates the values of x_{c_r}. To compensate for the fact that g_{sb}, g_{sb_r}, and g_{bc} will change every timestep, the covariance values of feature tracks used for the measurement update of the subfilter should be larger than the covariance values of feature tracks used in the main EKF filter.

The initial state of the subfilter is $x_{c_r,0}$. The X and Y coordinates of $x_{c_r,0}$ can be computed by inverting the camera intrinsics $\pi^c(\cdot)$. The Z coordinate is initialized with a constant run-time parameter `z_init`. The initial covariance of the subfilter is a diagonal matrix. The parameters of the diagonal matrix are user-defined run-time parameters.

Before a feature is added to the EKF state, the subfilter must run for `ready_steps`, a user-defined number of parameters. Afterward, it becomes a candidate for addition to the state.

A.5.2 Two-View Triangulation (Optional)

Rather than initializing the subfilter of all features with a constant runtime parameter `z_init`, we may calculate a separate value of `z_init` for each feature using two-view triangulation. The two views are the first two frames in which the feature is visible. Two-view triangulation uses the estimates of g_{sb} at the latest two timesteps and the latest estimate of g_{bc} as if they were correct.

XIVO contains five implementations of two-view triangulation:

- Essential Matrix [Lon81]
- Direct Linear Transformation [Sut74]
- Minimization of L1, L2, and L-∞ angular reprojection error [LC19b]

Since the first two views of a feature are used in triangulation, triangulation will often fail

in practice due to a lack of parallax. When triangulation fails, XIVO will use the user-defined constant `z_init` instead of the triangulation output.

A.5.3 Depth Refinement (Optional)

Depth refinement occurs right before a feature is added to the EKF state. This may be after more timesteps than `ready_steps`. The depth refinement will directly edit the state of the subfilter, but not the covariance.

Depth refinement partially solves the following optimization problem:

$$\underset{x_{c_r}}{\text{minimize}} \quad \frac{1}{2}\sum_{i=1}^{N_{obs}}(x_p(i) - x_p^{obs}(i))^T R^{-1}(x_p(i) - x_p^{obs}(i)) \tag{A.58}$$

where R is the measurement covariance (a runtime parameter), $x_p(i)$ is the predicted measurement at timestep i computed from x_{c_r} and $x_p^{obs}(i)$ is the actual feature measurement at timestep i. The partial solving is done using a fixed number (a runtime parameter) of Newton steps.

If the total L2 error

$$\sum_{i=1}^{N_{obs}}(x_p(i) - x_p^{obs}(i)) \tag{A.59}$$

exceeds the threshold `max_res_norm`, a runtime parameter, then depth refinement *failed*. Otherwise, depth refinement is *successful*.

A.6 Loop Closure (Optional)

Loop closure is an optional measurement update using features with state `Limbo` (see Fig. A.4). If loop closure is enabled, XIVO's mapper module (enabled with the compile-time option `USE_MAPPER`) will look for possible loop closures after each measurement update. The loop closure process is:

1. Search for loop closures by matching descriptors of features in the current EKF state to features in `Limbo`. Descriptor matching is performed using the DBoW2 bag-of-words

library [GT12] and a nearest neighbor search. For each matched feature, retrieve the *old* value of X_s from the map.

2. Use Perspective-and-Point RANSAC to remove outliers from the list of matches. We use the Lambda Twist [PN18] solver. Since Perspective-and-Point RANSAC requires at least four matches, we consider all points outliers if there are fewer than four matches.

3. For the old values of X_s that remain, perform an EKF measurement update with the following measurement equation:

$$x_p = \pi^c(\pi(R_{bc}^T(R_{sb}^T X_s - T_{sb}) - T_{bc})) \tag{A.60}$$

   R_{bc}, T_{bc}, R_{sb}, and T_{sb} are read from the current state vector. X_s is treated like a known constant. The measurement covariance associated with equation (A.60) is a runtime parameter.

APPENDIX B

Supporting Figures for "Feature Tracks are not Zero-Mean Gaussian"

B.1 Supporting Figures for DTU Point Features Dataset

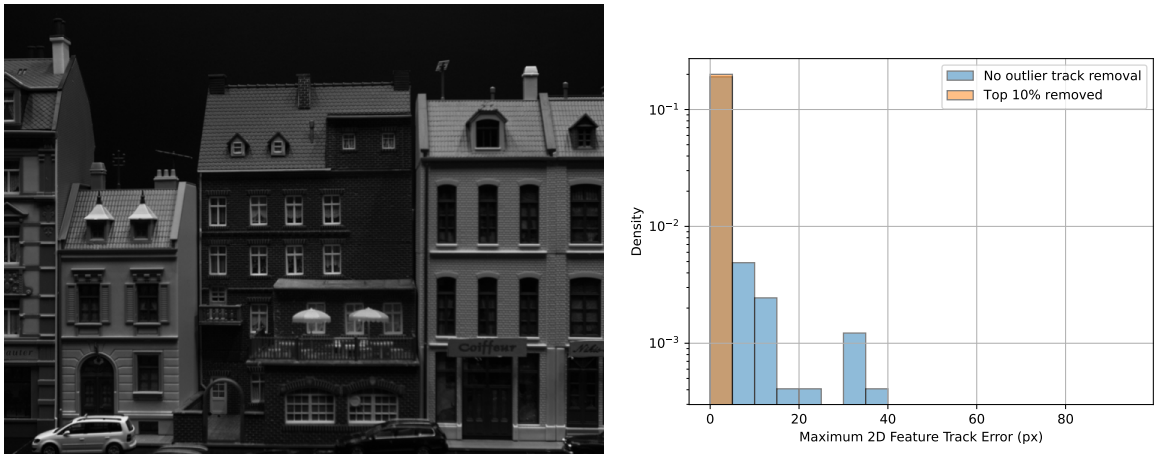


Figure B.1: **DTU Point Features Dataset: We will throw out the 10% of tracks with the most error from each scene.** The right figure plots the histogram density of all feature tracks' maximum L2 error in log scale. The corresponding scene is pictured on the left. Outliers in the blue histogram are caused by noisy depth measurements and the imperfect association of features with laser scan points.

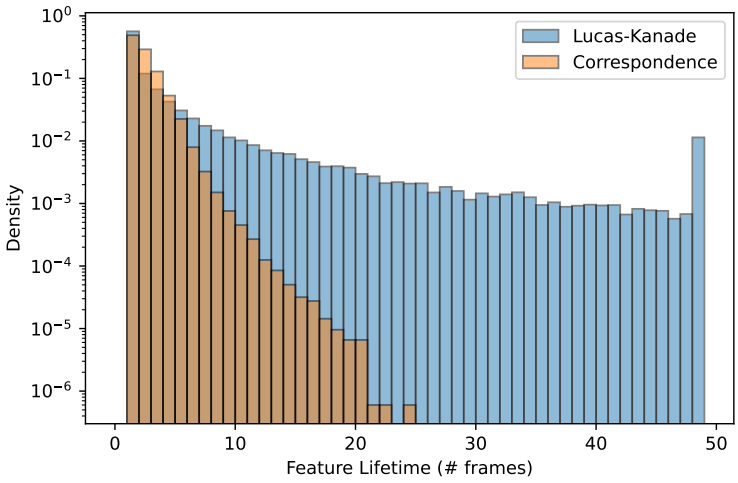
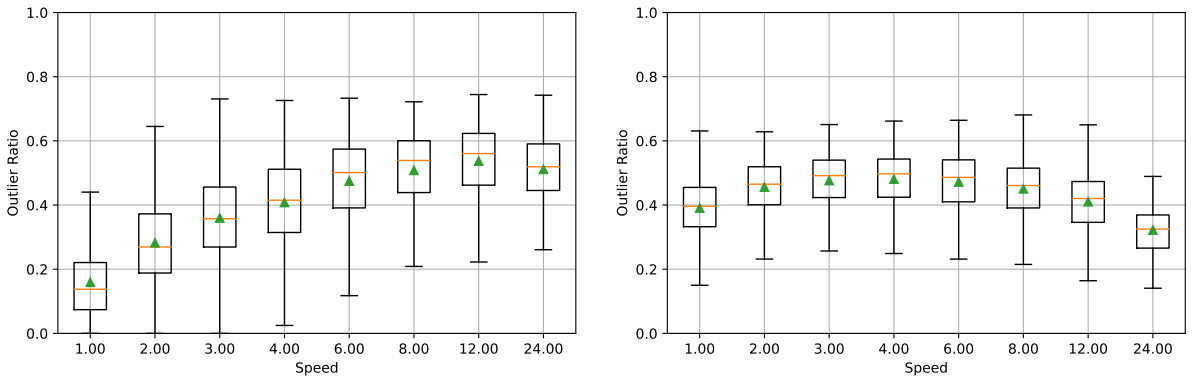


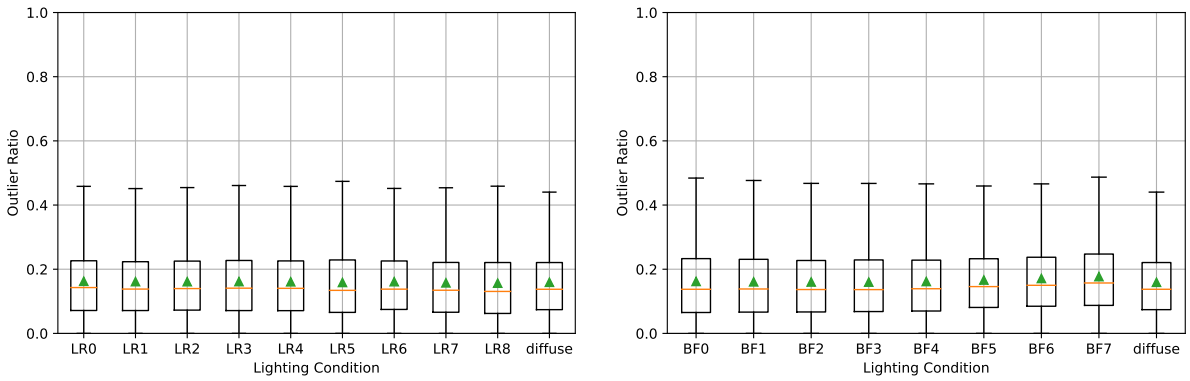
Figure B.2: **DTU Point Features Dataset: Feature lifetimes generated by the Lucas-Kande Tracker is a long tailed distribution.** The histograms above plot feature lifetime density in log scale for scenes with diffuse lighting and no skipped frames (speed=1.00) for the Lucas-Kanade (blue) and Correspondence (orange) Trackers. There is a long tail of tracks with longer lifetimes when we use a sparse optical flow rather than correspondences.



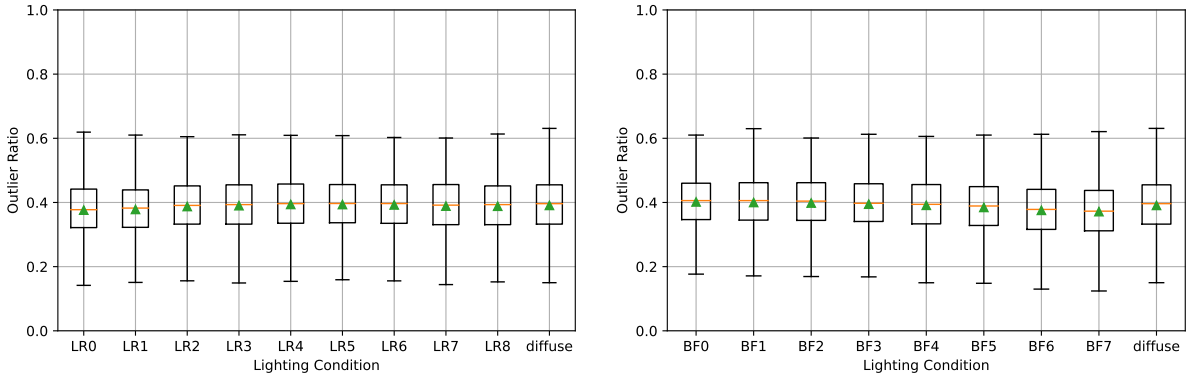
(a) Lucas-Kanade

(b) Correspondence

Figure B.3: **DTU Point Features Dataset: Outlier Ratio Depends on Speed.** In the box-and-whisker plots above, the orange line is the median, the green triangle is the mean, and the box extends from the first to the third quartiles. The whiskers extend up to 1.5x the length of the boxes. Outlier ratios increase with speed for all tested feature trackers to a point, and then falls slightly. Each box-and-whisker is computed using features from all 60 scenes, one tracker, and one speed. Outlier ratios then decrease at higher speeds not because the tracker is more accurate, but because the percentage of features that fail to be tracked from frame to frame increases.

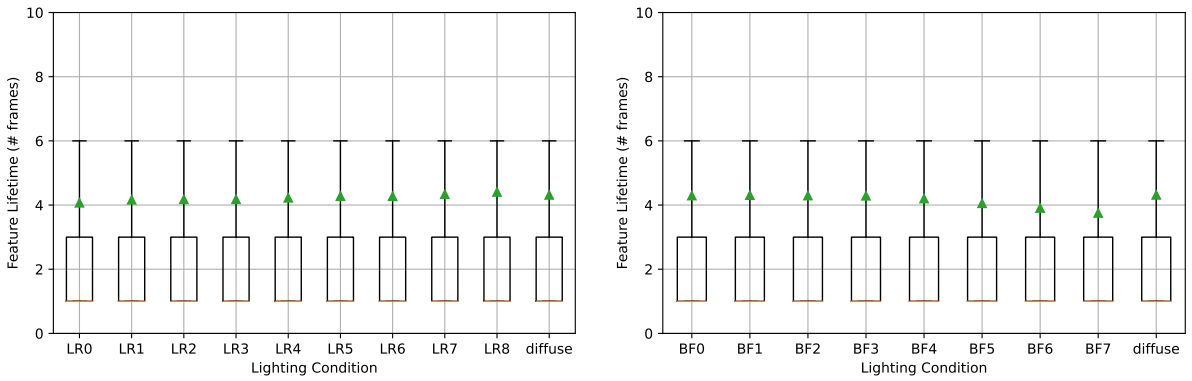


(a) Lucas-Kanade

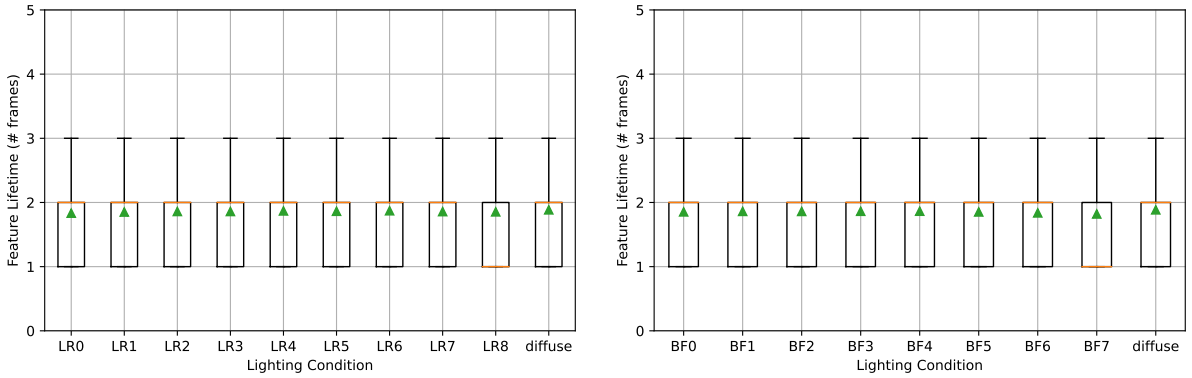


(b) Correspondence

Figure B.4: **DTU Point Features Dataset: Outlier ratio does not depend on the existence of directional lighting.** In the box-and-whisker plots above, the orange line is the median, the green triangle is the mean, and the box extends from the first to the third quartiles. The whiskers extend up to 1.5x the length of the boxes. Each box-and-whisker plot is computed using features from all 60 scenes, one tracker, speed=1.00, and one of the lighting conditions in Figure 4.1. The distribution of outlier ratio is approximately the same for all lighting conditions.

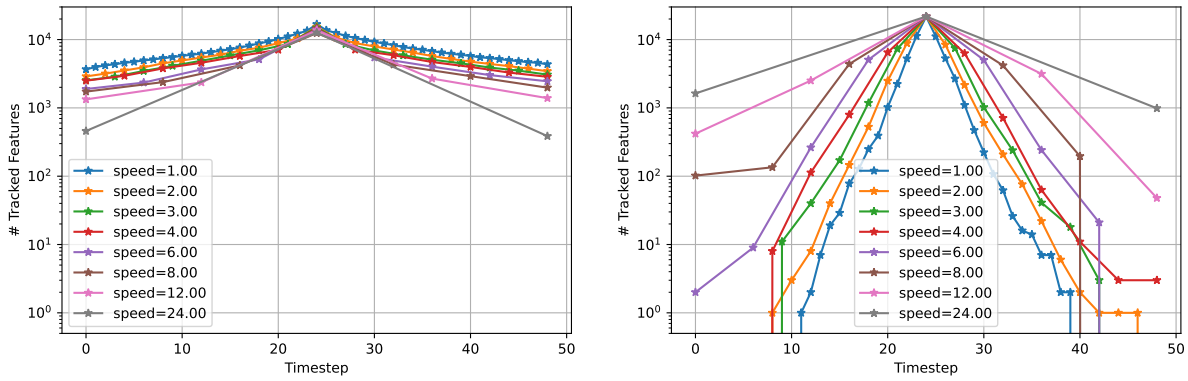


(a) Lucas-Kanade



(b) Correspondence

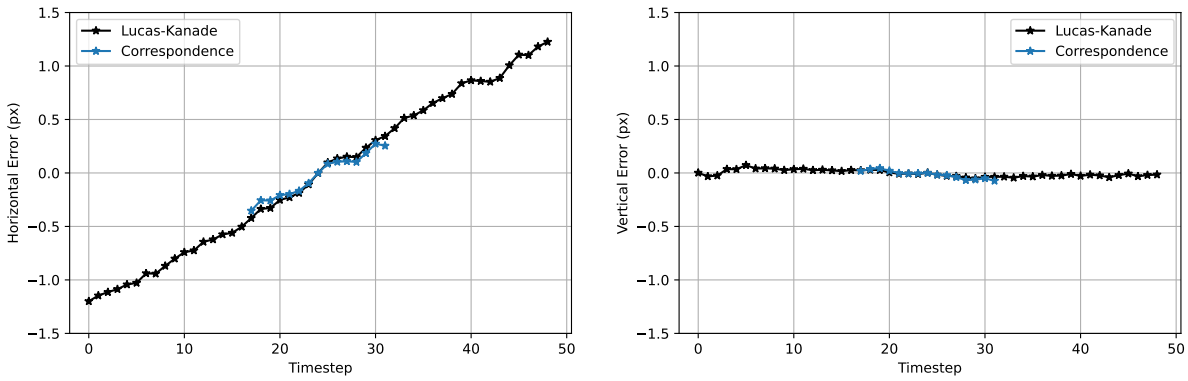
Figure B.5: **DTU Point Features Dataset: Feature lifetime does not depend on the existence of directional lighting.** In the box-and-whisker plots above, the orange line is the median, the green triangle is the mean, and the box extends from the first to the third quartiles. The whiskers extend up to 1.5x the length of the boxes. Outlier ratios increase with speed for all tested feature trackers to a point, and then falls slightly. Each box-and-whisker is computed using features from all 60 scenes, one tracker, and one speed. The distribution of feature lifetime is approximately the same for all lighting conditions.



(a) Lucas-Kanade

(b) Correspondence

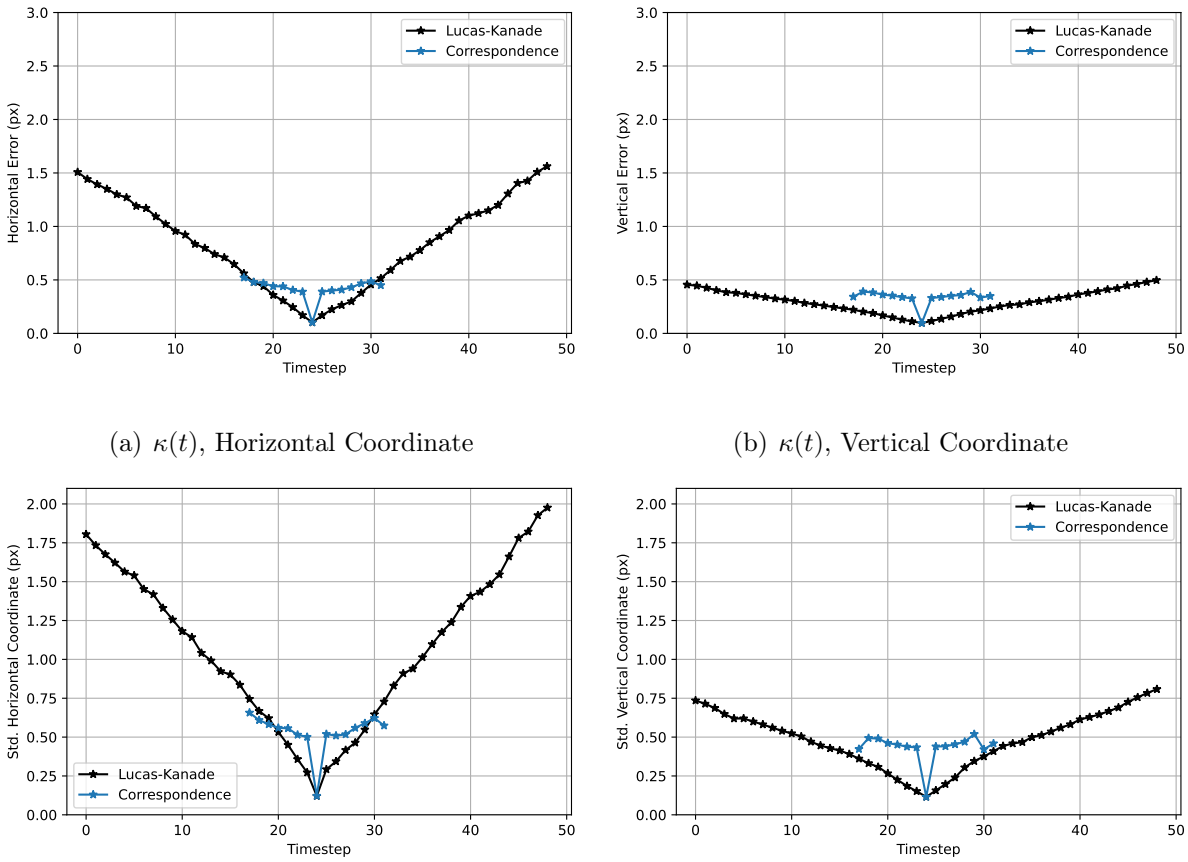
Figure B.6: Each curve shows the total number of tracked features at each timestep for the Lucas-Kanade Tracker (left) and the Correspondence Tracker (right) in log scale. Each dot on a curve is a frame in the sequence and each curve is computed using all features visible in the Key Frame under diffuse lighting and one speed. The number of features that can be used to compute mean $\mu(t)$ and covariance $\Sigma(t)$ declines quickly away from the Key Frame when using the Correspondence Tracker. When using the Lucas-Kanade Tracker, a slower speed means that more features are tracked for more frames. When using the Correspondence Tracker, the number of features tracked is dependent on the number of frames as well as the speed for the reasons noted in Section 4.2.1. The closer two frames are (i.e., the slower the speed), the fewer features are dropped between them. This is consistent with previously known results about the precision and recall of feature descriptors [MS05, SHS17a, WOB17]. **We limit calculations of mean error $\mu(t)$, mean absolute error, $\kappa(t)$, and covariance $\Sigma(t)$ to timesteps that contain at least 100 features.**



(a) $\mu(t)$, Horizontal Coordinate

(b) $\mu(t)$, Vertical Coordinate

Figure B.7: **DTU Point Features Dataset: At nominal speed and with diffuse lighting, the tracker used has little effect on $\mu(t)$.** Lines shown are mean feature track errors $\mu(t)$ at each timestep t calculated over all scenes. The blue lines are feature track errors calculated using the Lucas-Kanade Tracker and the orange lines are feature track errors calculated using the Correspondence Tracker. Lines are cut-off to timesteps where at least 100 features with 3D data are available (see Fig. B.6). The orange lines are on top of the blue lines, therefore the tracker used does not affect mean error.



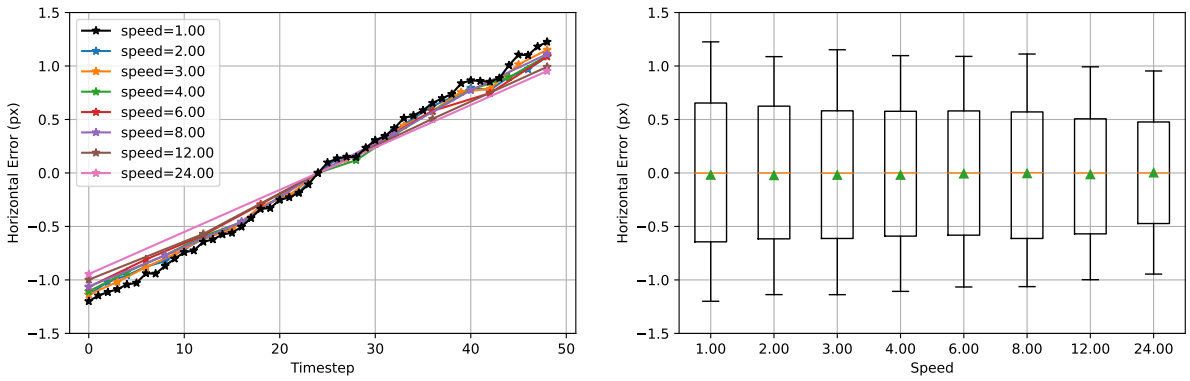
(a) $\kappa(t)$, Horizontal Coordinate

(b) $\kappa(t)$, Vertical Coordinate

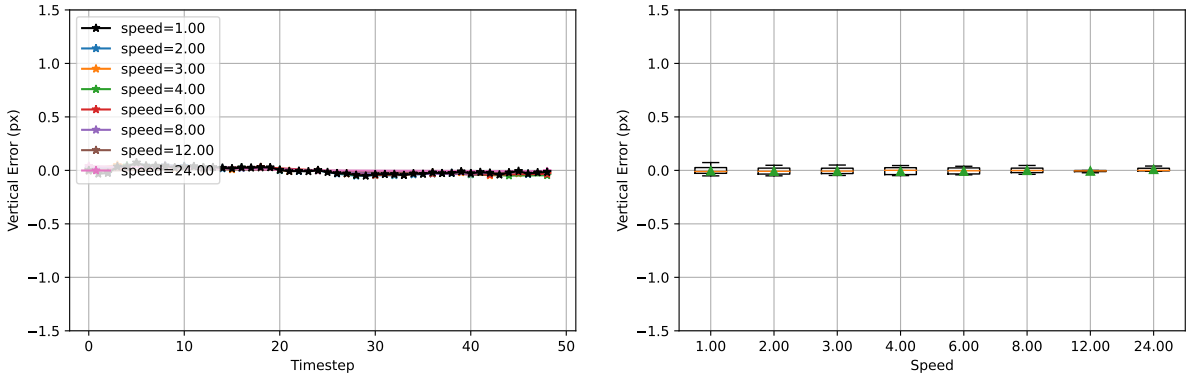
(c) $\Sigma(t)$, Horizontal Covariance

(d) $\Sigma(t)$, Vertical Coordinate

Figure B.8: **DTU Point Features Dataset: At nominal speed and under diffuse lighting, the tracker used does affect mean absolute error $\kappa(t)$ and covariance $\Sigma(t)$.** Lines shown are horizontal and vertical coordinates of $\kappa(t)$ (top row), and $\Sigma(t)$ (bottom row) calculated using all tracks from all scenes. Each dot corresponds to a single frame. Mean absolute error and covariance for the Correspondence Tracker are roughly constant with respect to time, while the same values for the Lucas-Kanade Tracker increases steadily with time away from the Key Frame.

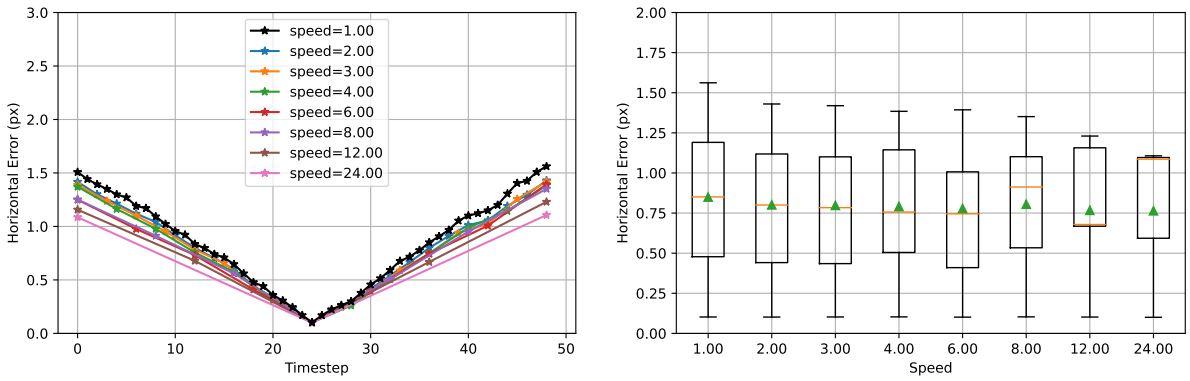


(a) $\mu(t)$, Horizontal Coordinate

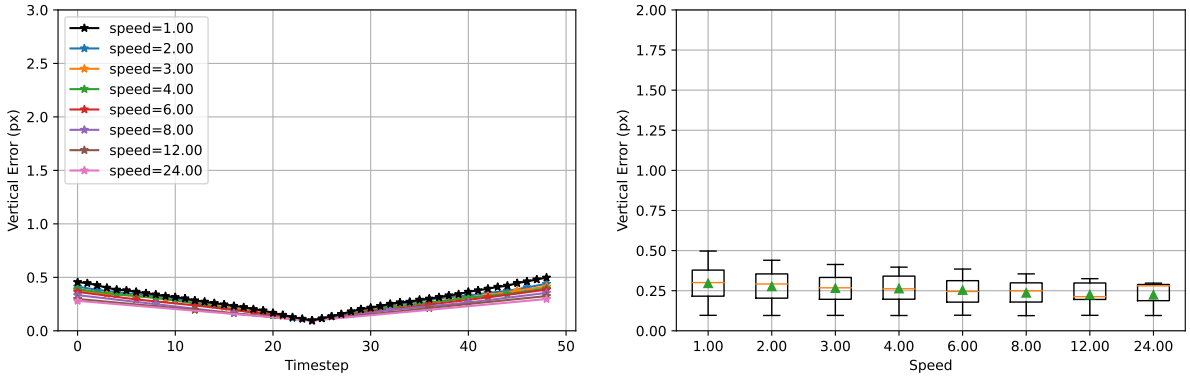


(b) $\mu(t)$, Vertical Coordinate

Figure B.9: **DTU Point Features Dataset: Speed affects mean error when using the Lucas-Kanade Tracker with diffuse lighting.** The left column contains plots of the horizontal and vertical coordinates of mean error $\mu(t)$ at each timestep and multiple speeds. Each dot corresponds to a processed frame. The right column plots the ordinate value of each line in the left figures as a box plot. As speed is increased, the slope of the horizontal components of $\mu(t)$ in the left plots (eq. (4.6)) decreases and the height of each box in the right plot decreases, i.e. the absolute magnitude of $\mu(t)$ slighty decreases. This trend indicates the existence of two speed-related components that affect $\mu(t)$: the difference between frames and the number of frames that have elapsed; the former has a much larger effect than the latter. The latter occurs because the exact point that the Lucas-Kanade Tracker tracks drifts with each frame. Fewer frames means that the tracked point has fewer opportunities to drift.

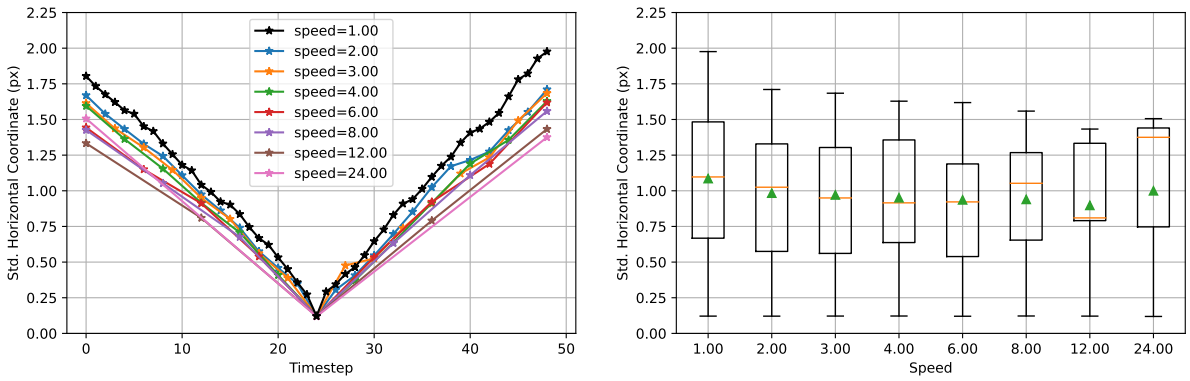


(a) $\kappa(t)$, Horizontal Coordinate

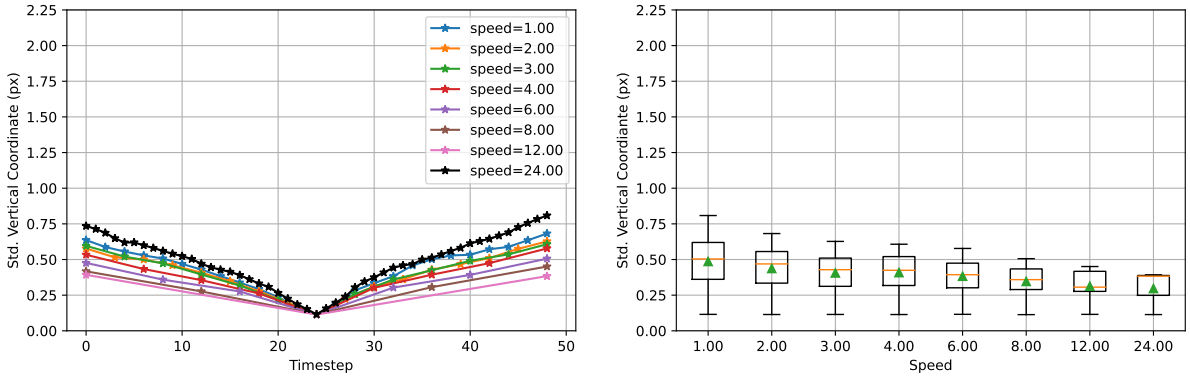


(b) $\kappa(t)$, Vertical Coordinate

Figure B.10: **DTU Point Features Dataset: Speed affects mean absolute error when using the Lucas-Kanade Tracker with diffuse lighting.** The left column contains plots of the horizontal (top row) and vertical (bottom row) coordinates of mean absolute error $\kappa(t)$ at each timestep and multiple speeds. Each dot corresponds to a processed frame. The right column plots the ordinate value of each line in the left figures as a box plot. As speed is increased, the mean absolute error at each timestep slightly decreases. This indicates the existence of two speed-related components that affect $\kappa(t)$: the difference between frames and the number of frames that have elapsed; the former has a much larger effect than the latter. The latter occurs because the exact point that the Lucas-Kanade Tracker tracks drifts with each frame. Fewer frames means that the tracked point has fewer opportunities to drift.

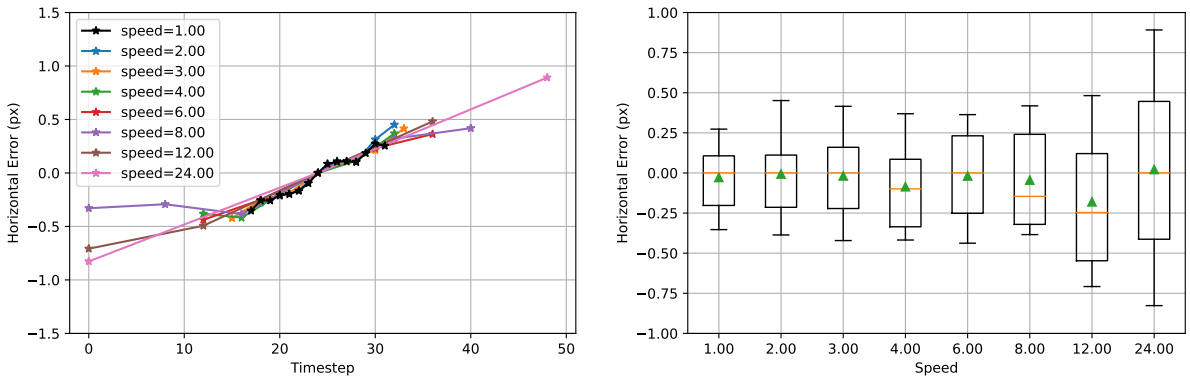


(a) $\Sigma(t)$, Horizontal Covariance

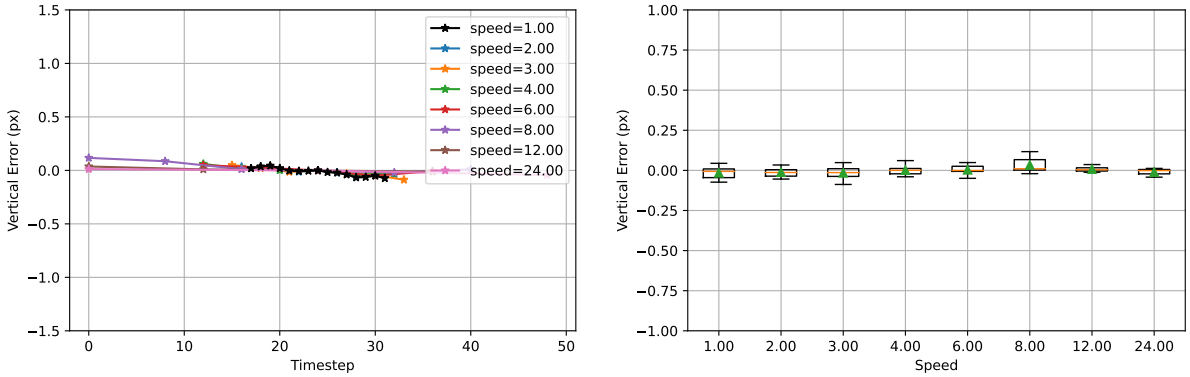


(b) $\Sigma(t)$, Vertical Coordinate

Figure B.11: **DTU Point Features Dataset: Speed affects covariance when using the Lucas-Kanade Tracker with diffuse lighting.** The left column contains plots of the square root of the horizontal (top row) and vertical (bottom row) coordiantes of $\Sigma(t)$ at each timestep and multiple speeds. Each dot cooresponds to a processed frame. The right column plots the ordinate value of each line in the left figures as a box plot. As speed is increased, the covariance of both the horizontal and vertical coordiantes slightly decreases; the lines in the left plot become slightly less steep and mean values of covariance get slightly smaller. This indicates the existence of two speed-related components to these statistics: the difference between frames and the number of frames that have elapsed; the former has a much larger effect than the latter. The latter occurs because the exact point that the Lucas-Kanade Tracker tracks drifts with each frame. Fewer frames means that the tracked point has fewer opportunities to drift.

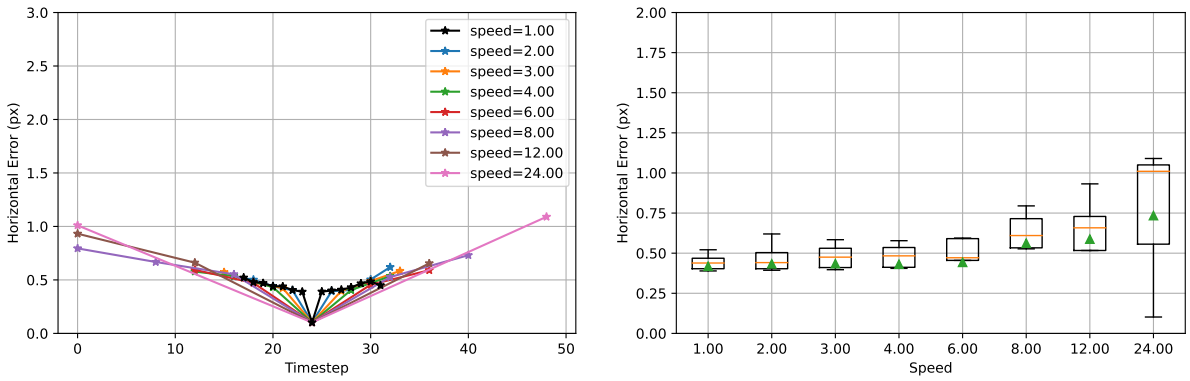


(a) $\mu(t)$, Horizontal Coordinate

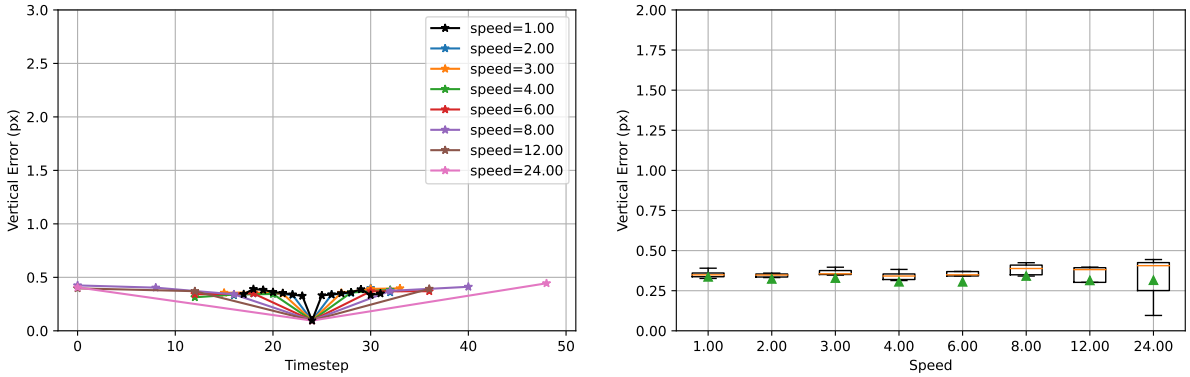


(b) $\mu(t)$, Vertical Coordinate

Figure B.12: **DTU Point Features Dataset: When using the Correspondence Tracker with diffuse lighting, mean error $\mu(t)$ is not affected by speed.** The left column contains plots of the horizontal coordinate (top row) and vertical coordinate (bottom row) of $\mu(t)$ at each timestep and multiple speeds. Each dot corresponds to a processed frame. The right column plots the ordinate value of each line in the left figures as a box plot. Both the line and box plots only contain timesteps that contain at least 100 tracked features (see Fig. B.6), leading to some asymmetry of the lines about the Key Frame. As speed is increased, there is no change in both the horizontal and vertical coordinates, as all lines in the left column plots are on top of one another. The boxes in the box plots of the horizontal coordinate are taller for higher speeds because the time cutoff for those speeds is longer than for the lower speeds, allowing more error to appear in the tracked features.

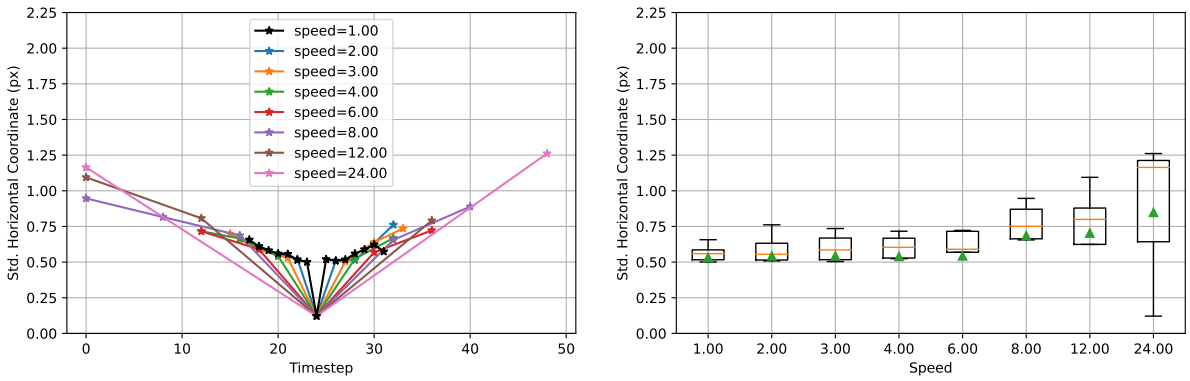


(a) $\kappa(t)$, Horizontal Coordinate

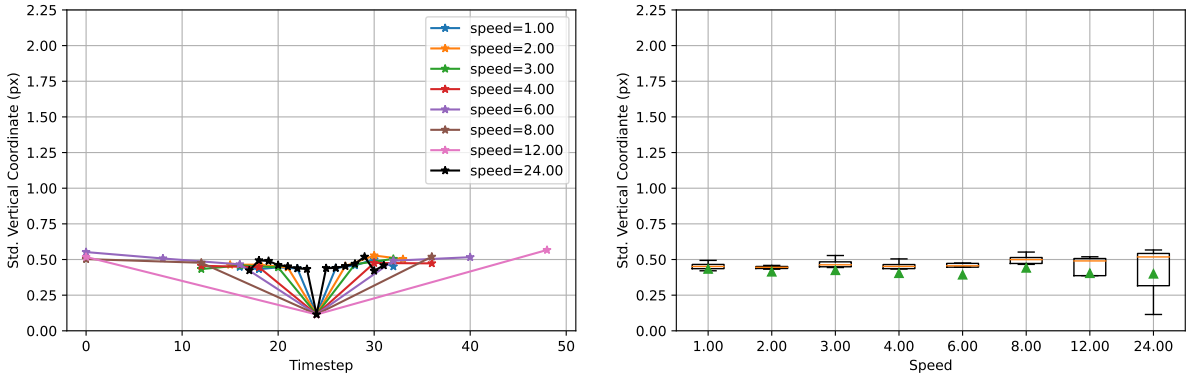


(b) $\kappa(t)$, Vertical Coordinate

Figure B.13: **DTU Point Features Dataset: When using the Correspondence Tracker with diffuse lighting, mean absolute error $\kappa(t)$ increases in the horizontal direction, but not the vertical direction, as speed is increased.** The left column contains plots of the horizontal coordinate (top row) and vertical coordinate (bottom row) of $\kappa(t)$ at each timestep and multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate value of each line in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. Both the line and box plots only contain timesteps that contain at least 100 tracked features (see Fig. B.6). The mean and median values of the horizontal coordinate of $\kappa(t)$ increases as speed is increased.

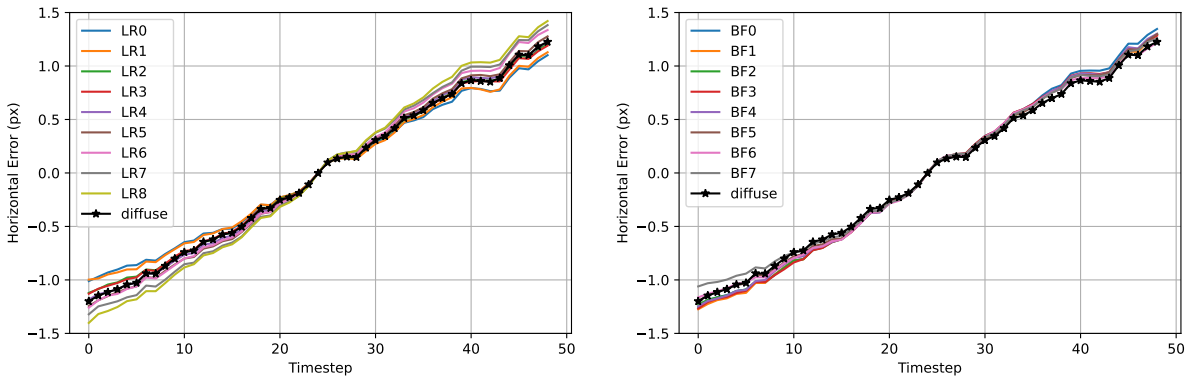


(a) $\Sigma(t)$, Horizontal Covariance

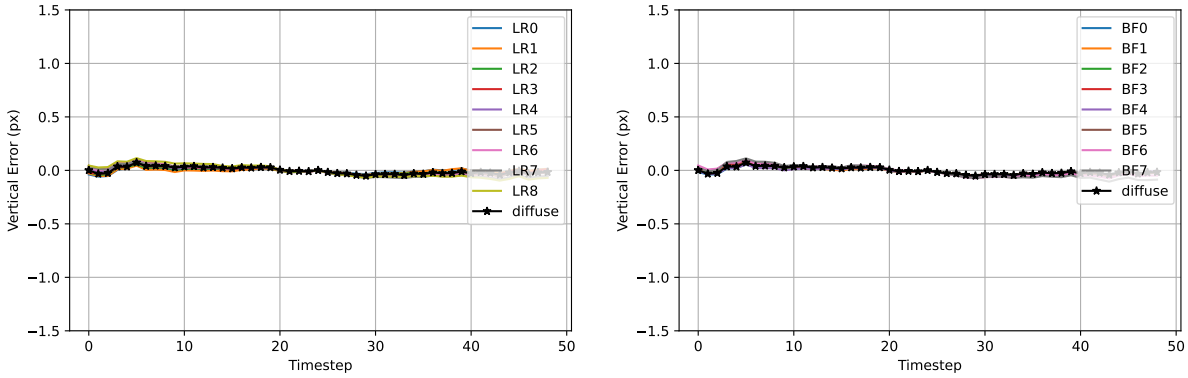


(b) $\Sigma(t)$, Vertical Coordinate

Figure B.14: **DTU Point Features Dataset: When using the Correspondence Tracker with diffuse lighting, covariance $\Sigma(t)$ increases in the horizontal direction, but not the vertical direction, as speed is increased.** The left column contains plots of the square root of the horizontal coordinate (top row) and vertical coordinate (bottom row) of $\Sigma(t)$ at each timestep and multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate value of each line in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. Both the line and box plots only contain timesteps that contain at least 100 tracked features (see Fig. B.6). The mean and median values of the horizontal value of $\Sigma(t)$ as speed is increased.

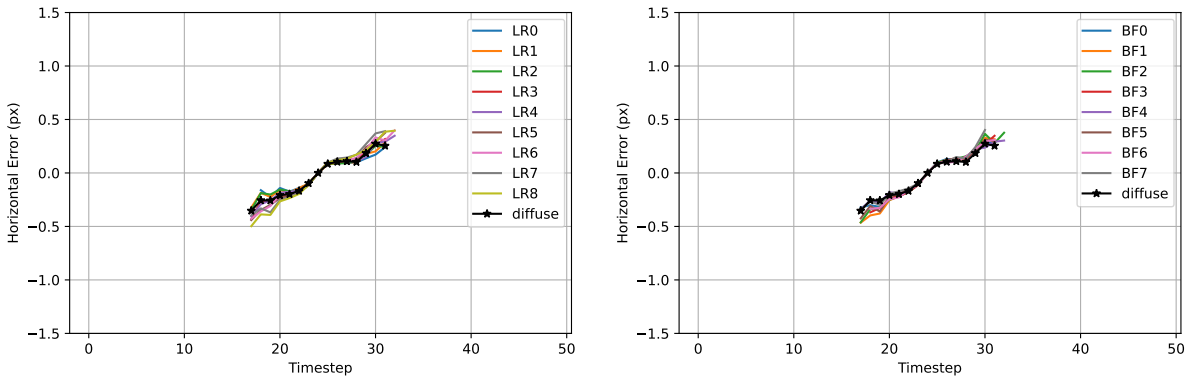


(a) $\mu(t)$, Horizontal Coordinate

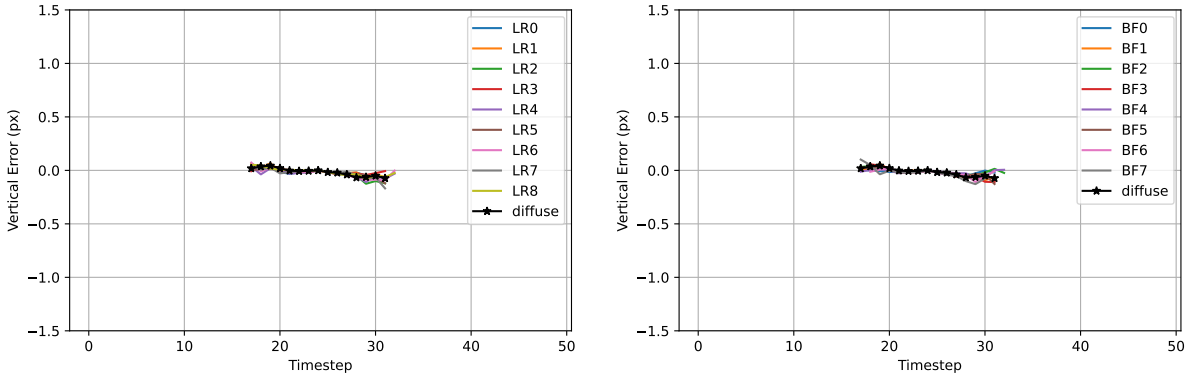


(b) $\mu(t)$, Vertical Coordinate

Figure B.15: **DTU Point Features Dataset: The existence of directional lighting does not change trends in mean error $\mu(t)$ when using the Lucas-Kande Tracker at nominal speed.** We compute $\mu(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Results for the horizontal coordinate are in the top row and results for the vertical coordinate are in the bottom row. Timesteps are limited to those that contain at least 100 features. The variation of $\mu(t)$ due to the existence of directional lighting is at most 10 percent the size of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

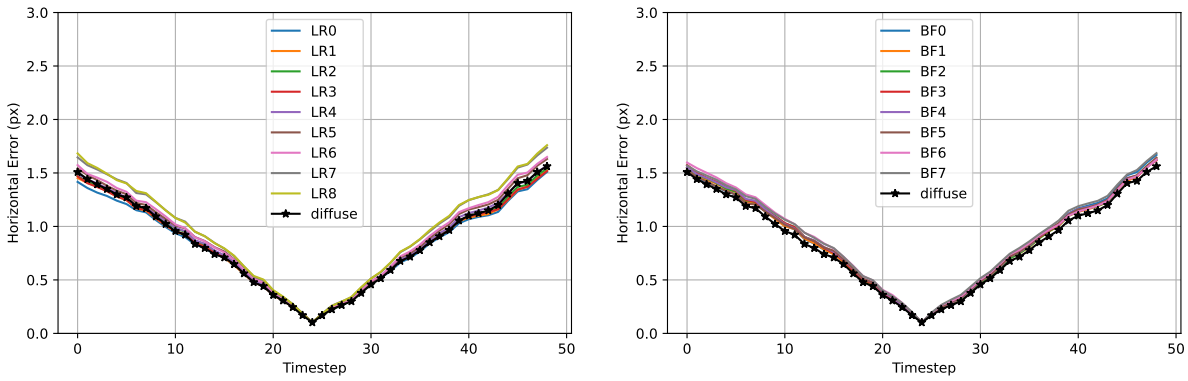


(a) $\mu(t)$, Horizontal Coordinate

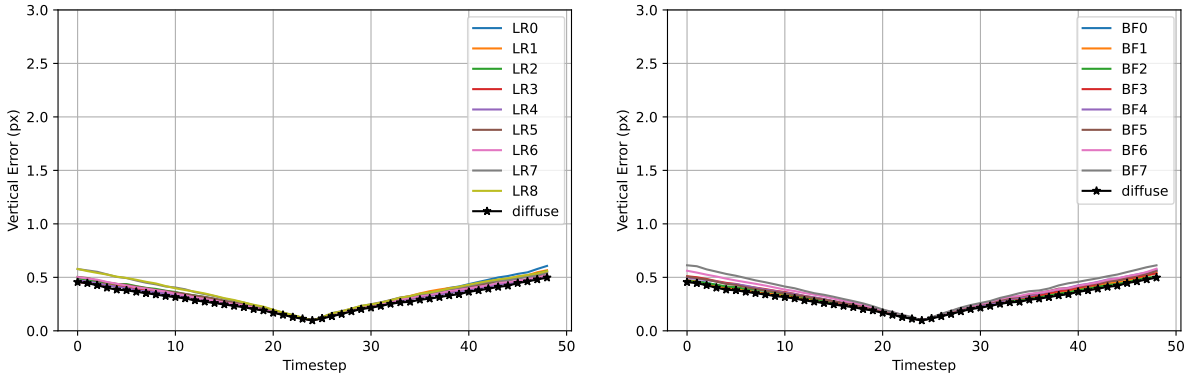


(b) $\mu(t)$, Vertical Coordinate

Figure B.16: **DTU Point Features Dataset: The existence of directional lighting does not change trends in mean error $\mu(t)$ when using the Correspondence Tracker at nominal speed.** We compute $\mu(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Results of the horizontal coordinate are shown in the top row and results for the vertical coordinate are shown in the bottom row. Timesteps are limited to those that contain at least 100 features. The variation of $\mu(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effects of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

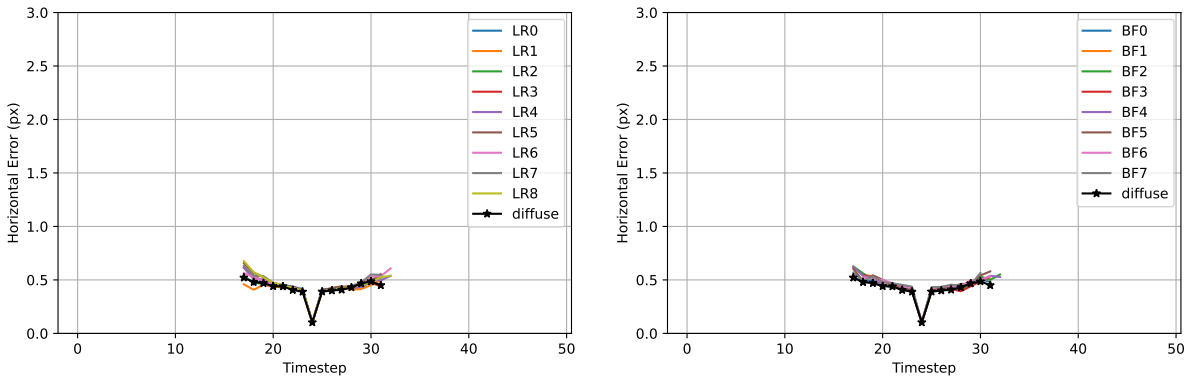


(a) $\kappa(t)$, Horizontal Coordinate

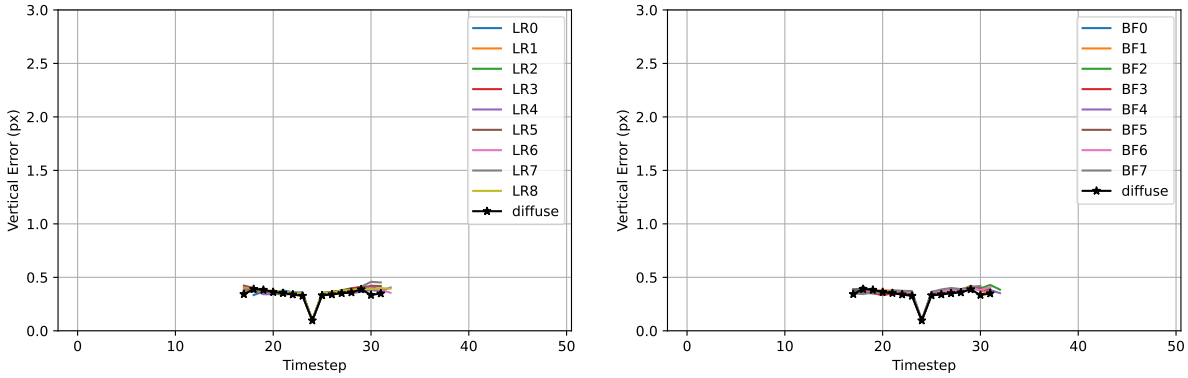


(b) $\kappa(t)$, Vertical Coordinate

Figure B.17: **DTU Point Features Dataset: The existence of directional lighting does not change trends in mean absolute error $\kappa(t)$ when using the Lucas-Kanade Tracker at nominal speed.** We compute $\mu(t)$ at each timestep using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Timesteps are limited to those that contain at least 100 features. The variation of $\kappa(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

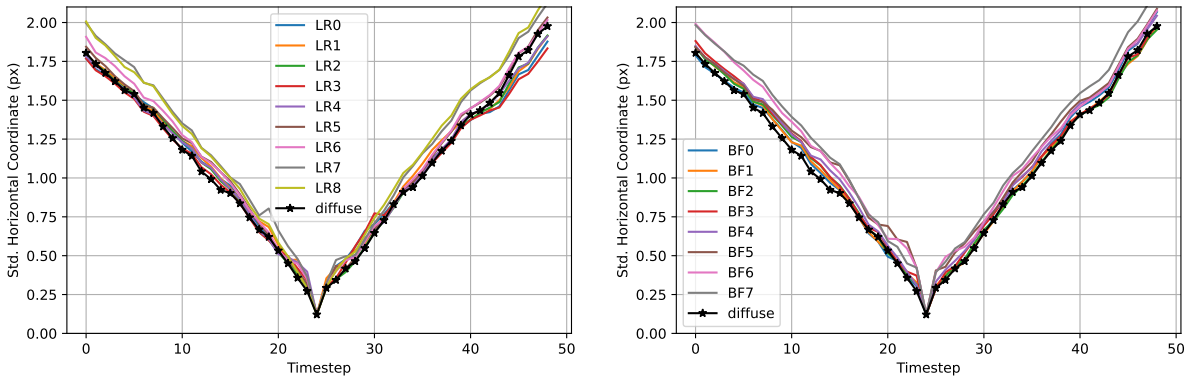


(a) $\kappa(t)$, Horizontal Coordinate

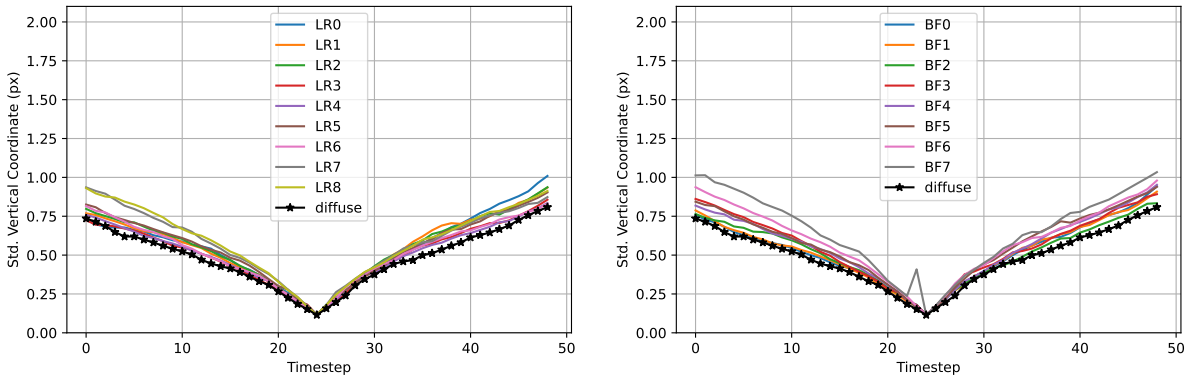


(b) $\kappa(t)$, Vertical Coordinate

Figure B.18: **DTU Point Features Dataset: The existence of directional lighting does not change trends in mean absolute error $\kappa(t)$ when using the Correspondence Tracker at nominal speed.** We compute $\kappa(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Timesteps are limited to those that contain at least 100 features. The variation of $\kappa(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

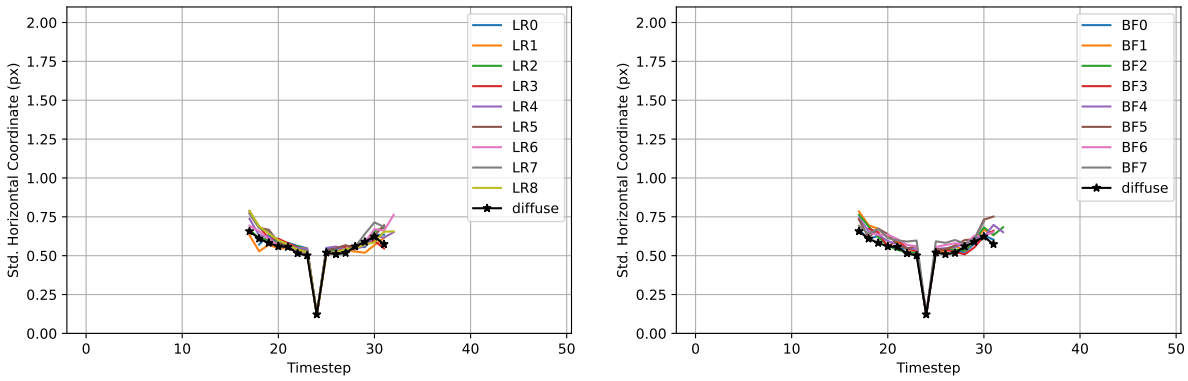


(a) $\Sigma(t)$, Horizontal Coordinate

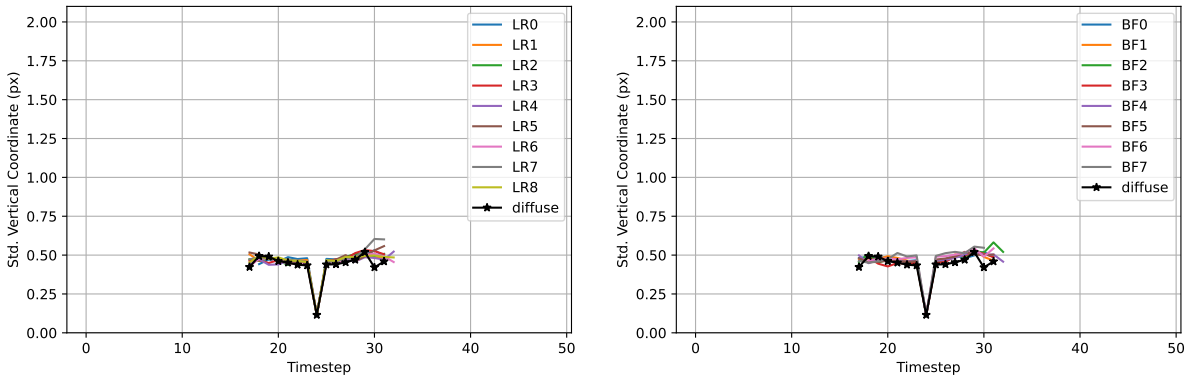


(b) $\Sigma(t)$, Vertical Coordinate

Figure B.19: **DTU Point Features Dataset: The existence of directional lighting does not change trends in covariance $\Sigma(t)$ when using the Lucas-Kanade Tracker at nominal speed.** We compute $\Sigma(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Timesteps are limited to those that contain at least 100 features. The variation of $\Sigma(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting. The blip in the bottom-right figure is due to one specific scene where the AGAST tracker finds very few features, causing a failure in tracking and outlier rejection, and then calculation of $\Sigma(t)$ downstream.

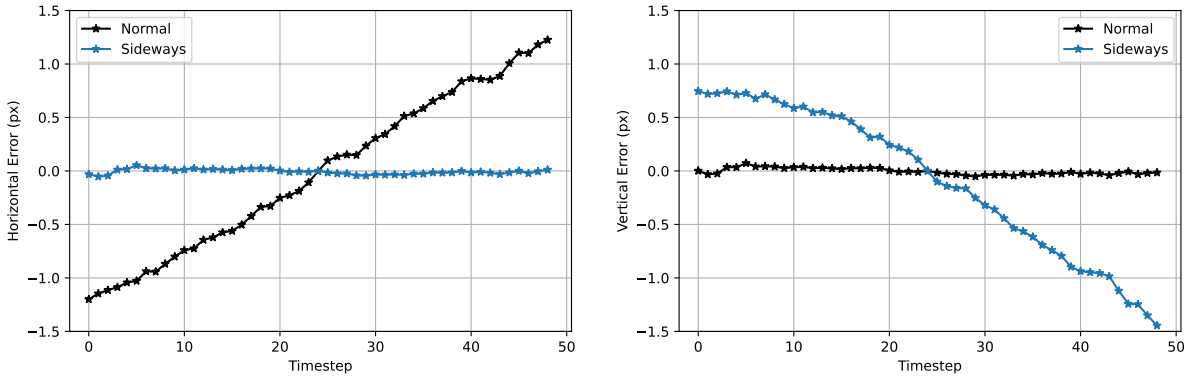


(a) $\Sigma(t)$, Horizontal Coordinate

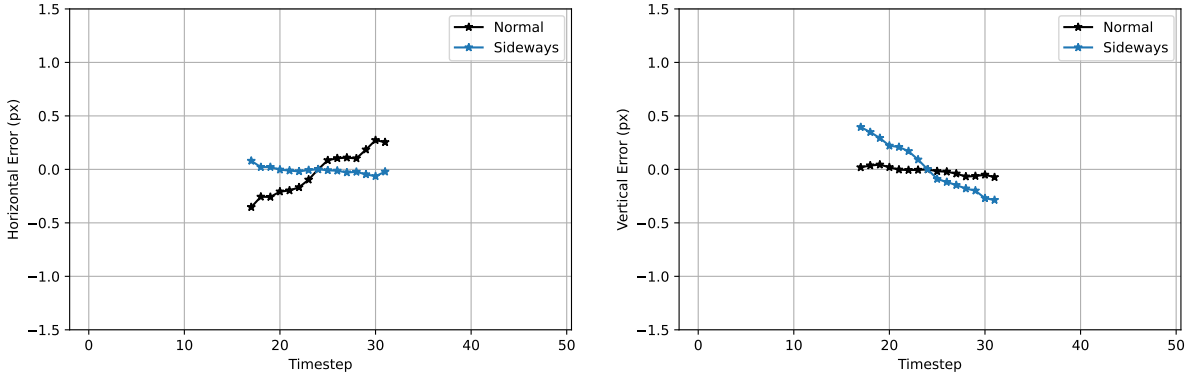


(b) $\Sigma(t)$, Vertical Coordinate

Figure B.20: **DTU Point Features Dataset: The existence of directional lighting does not change trends in covariance $\Sigma(t)$ when using the Correspondence Tracker at nominal speed.** We compute $\Sigma(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Timesteps are limited to those that contain at least 100 features. The variation of $\Sigma(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is small because changes between adjacent frames are small whether or not the scene contains directional lighting.

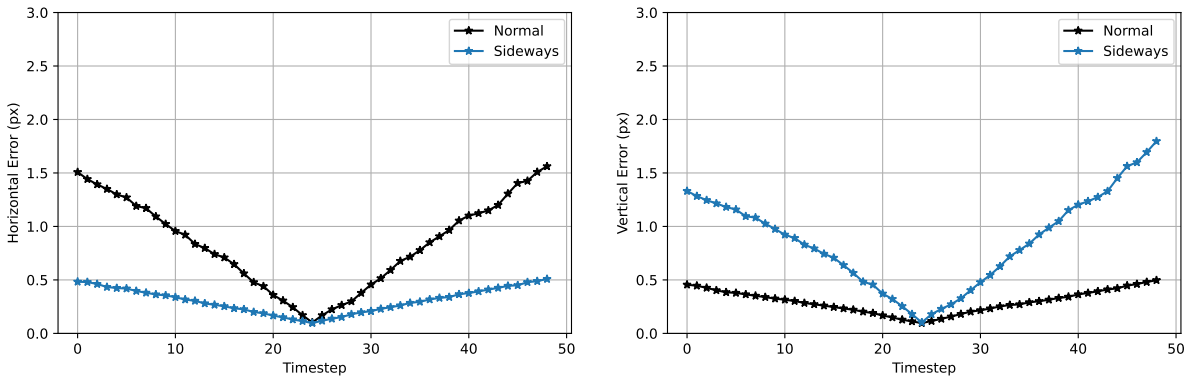


(a) Lucas-Kanade



(b) Correspondence

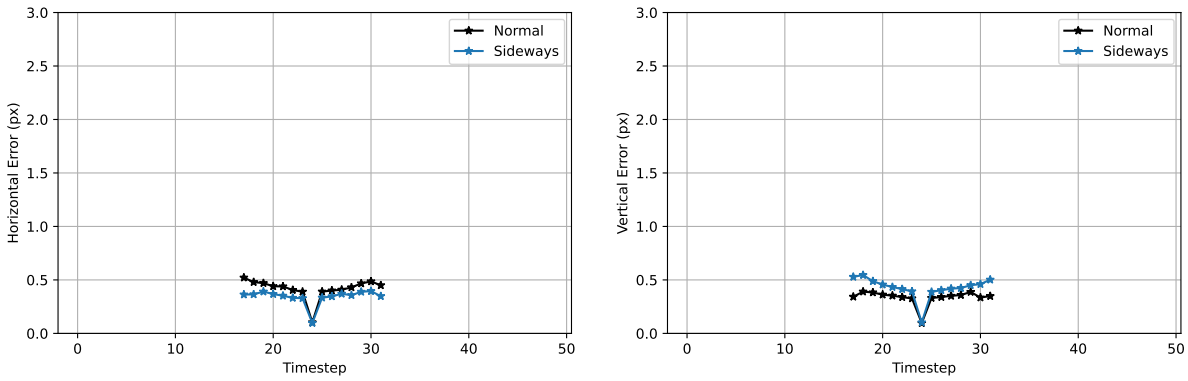
Figure B.21: **DTU Point Features Dataset: Mean errors are larger about the direction of motion for both the Lucas-Kanade and Correspondence Trackers.** In Figures B.9, B.12, B.15, and B.16, the horizontal component (left column) of $\mu(t)$ was always larger than the vertical component (right column). When images are rotated 90 degrees counterclockwise ("sideways"), the trend is reversed. Errors shown above are computed for the Lucas-Kanade Tracker at nominal speed and in diffuse lighting.



(a) $\kappa(t)$, Horizontal Coordinate

(b) $\kappa(t)$, Vertical Coordinate

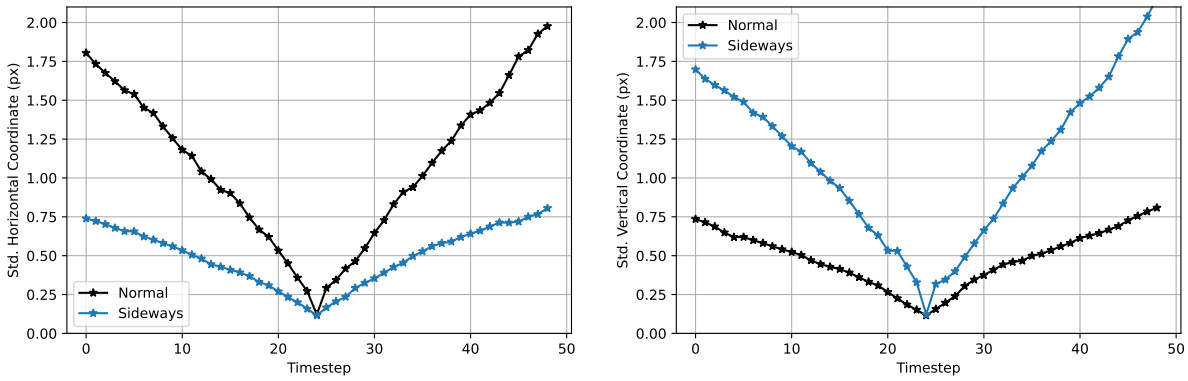
Figure B.22: **DTU Point Features Dataset: Mean absolute errors are larger about the direction of motion when using the Lucas-Kanade Tracker.** In Figures B.9, B.12, B.15, and B.16, the horizontal component of $\kappa(t)$ was always larger than the vertical component. When images are rotated 90 degress counterclockwise ("sideways"), the trend is reversed. Mean absolute errors shown above are computed for the Lucas-Kanade Tracker at nominal speed and in diffuse lighting.



(a) $\kappa(t)$, Horizontal Coordinate

(b) $\kappa(t)$, Vertical Coordinate

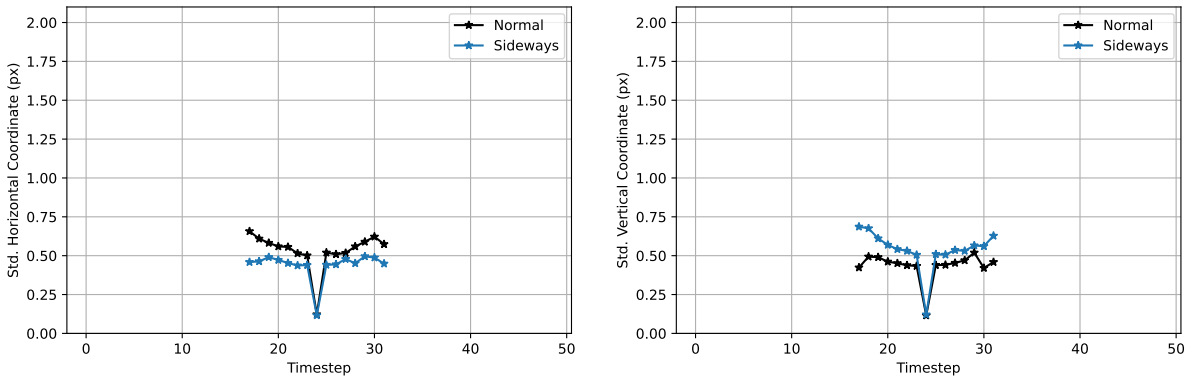
Figure B.23: **DTU Point Features Dataset: The direction of motion does not affect mean absolute error when using the Correspondence Tracker.** In Figures B.13, and B.18, the difference between the horizontal and vertical components of $\kappa(t)$ was a fraction of the size of $\kappa(t)$ in both components. When images are rotated 90 degrees counterclockwise ("sideways"), the trend is the same. Errors shown above are computed for the Correspondence Tracker at nominal speed and in diffuse lighting.



(a) $\Sigma(t)$, Horizontal Coordinate

(b) $\Sigma(t)$, Vertical Coordinate

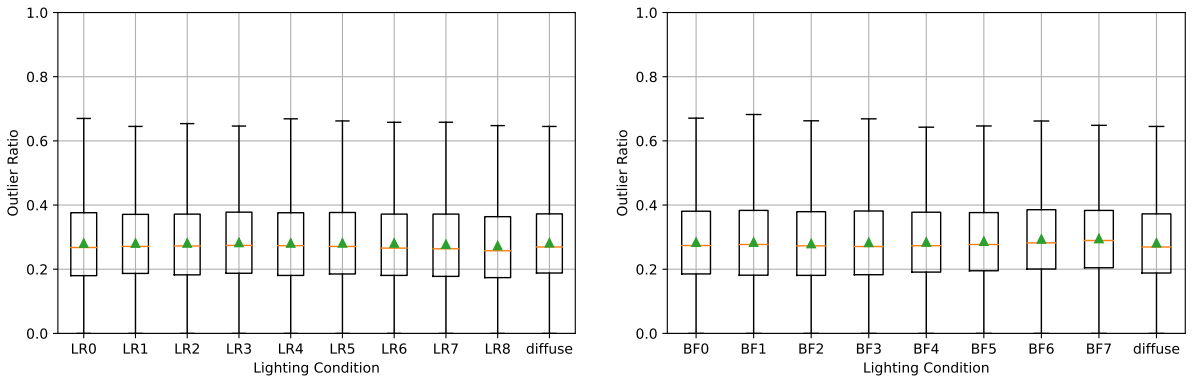
Figure B.24: **DTU Point Features Dataset: Covariances are larger about the direction of motion when using the Lucas-Kanade Tracker.** In Figures B.11, B.14, B.19, and B.20, the horizontal component of $\Sigma(t)$ was always larger than the vertical component. When images are rotated 90 degrees counterclockwise ("sideways"), the trend is reversed for both errors (top row) and covariance (bottom row). Errors above are computed for the Lucas-Kanade Tracker at nominal speed and in diffuse lighting.



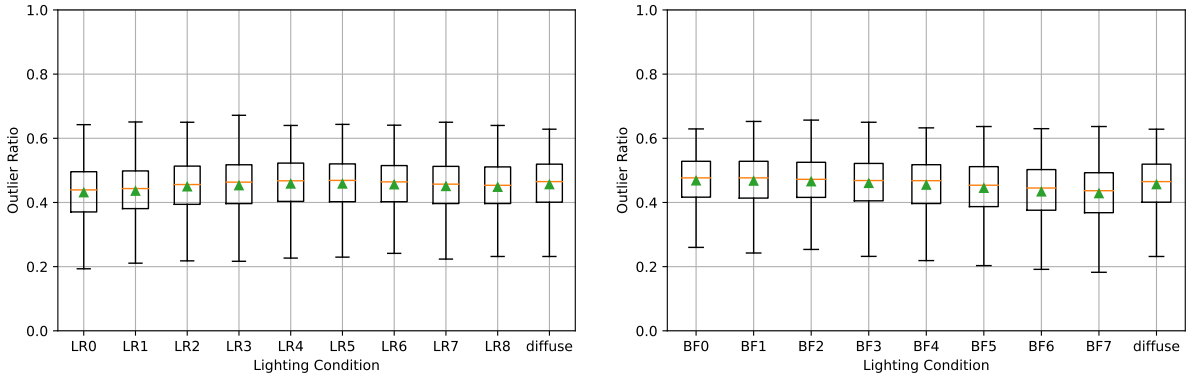
(a) $\Sigma(t)$, Horizontal Coordinate

(b) $\Sigma(t)$, Vertical Coordinate

Figure B.25: **DTU Point Features Dataset: The direction of motion does not affect covariance when using the Correspondence Tracker.** In Figures B.14, and B.20, the difference between the horizontal and vertical components of $\Sigma(t)$ was a fraction of the size of $\Sigma(t)$ in both components. When images are rotated 90 degrees counterclockwise ("sideways"), the trend is the same. Errors shown above are computed for the Correspondence Tracker at nominal speed and in diffuse lighting.

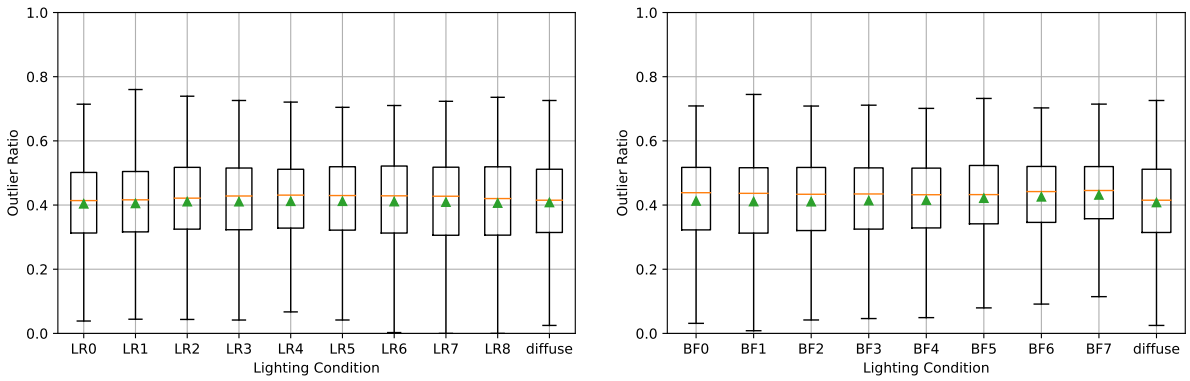


(a) Lucas-Kanade

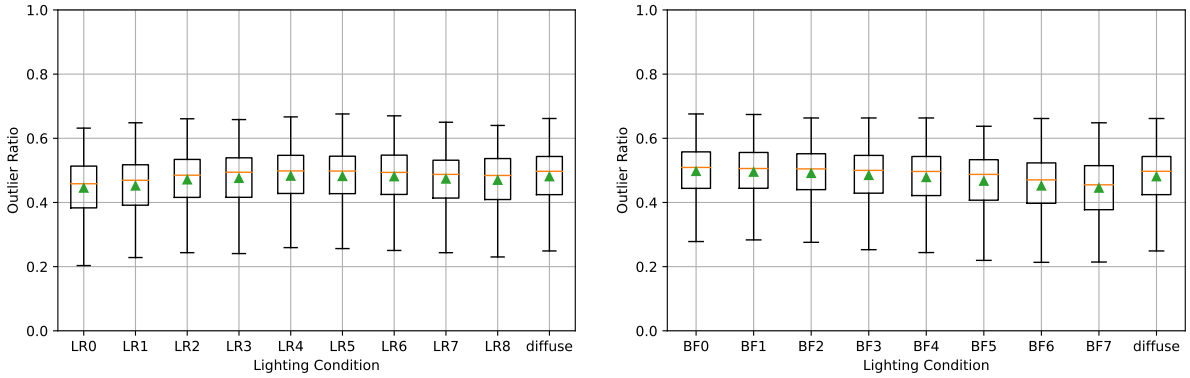


(b) Correspondence

Figure B.26: **At twice nominal speed, the existence of directional lighting does not affect outlier ratio.** In the box-and-whisker plots above, the orange line is the median, the green triangle is the mean, and the box extends from the first to the third quartiles. The whiskers extend up to 1.5x the length of the boxes. Each box-and-whisker plot is computed using features from all 60 scenes, one tracker, speed=2.00, and one of the lighting conditions in Figure 4.1. The distribution of outlier ratio is approximately the same for all lighting conditions.

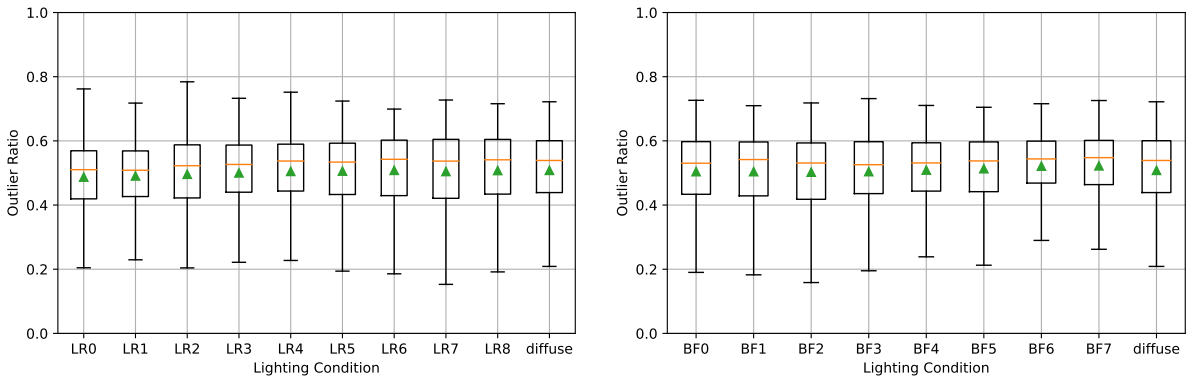


(a) Lucas-Kanade

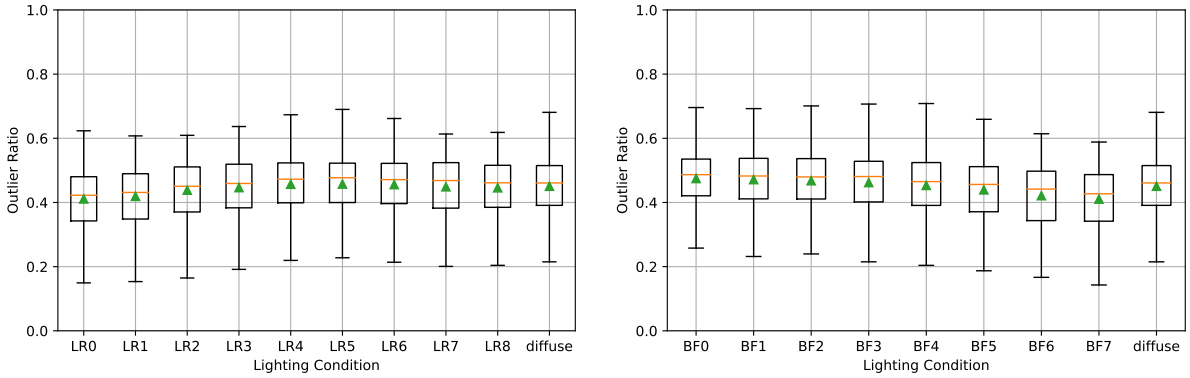


(b) Correspondence

Figure B.27: **At four times nominal speed, the existence of directional lighting does not affect outlier ratio.** In the box-and-whisker plots above, the orange line is the median, the green triangle is the mean, and the box extends from the first to the third quartiles. The whiskers extend up to 1.5x the length of the boxes. Each box-and-whisker plot is computed using features from all 60 scenes, one tracker, speed=4.00, and one of the lighting conditions in Figure 4.1. The distribution of outlier ratio is approximately the same for all lighting conditions.

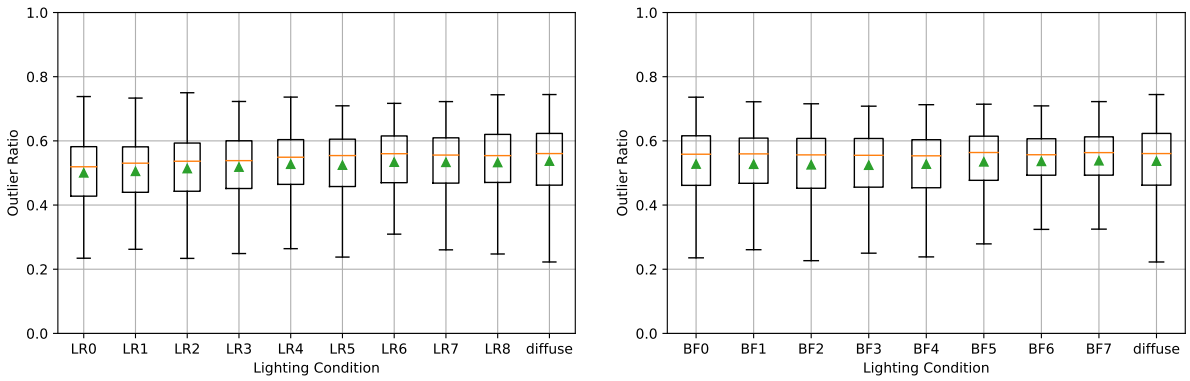


(a) Lucas-Kanade

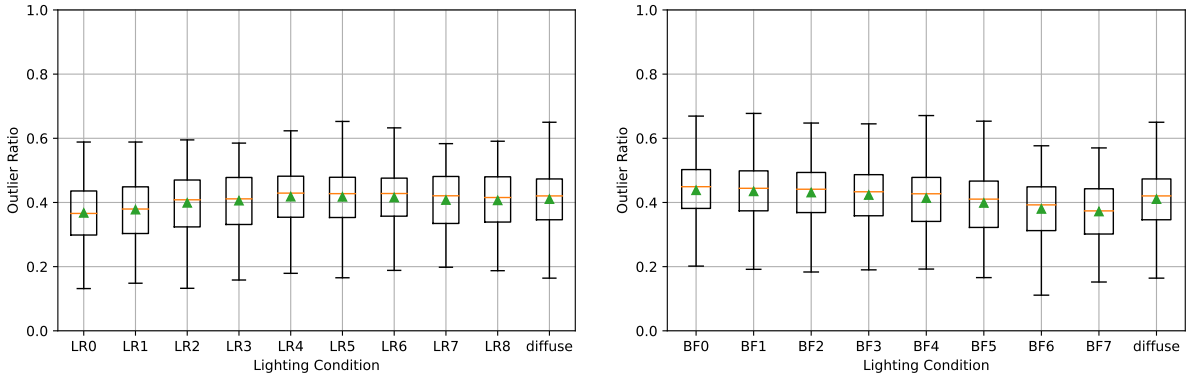


(b) Correspondence

Figure B.28: **At eight times nominal speed, the existence of directional lighting does not affect outlier ratio.** In the box-and-whisker plots above, the orange line is the median, the green triangle is the mean, and the box extends from the first to the third quartiles. The whiskers extend up to 1.5x the length of the boxes. Each box-and-whisker plot is computed using features from all 60 scenes, one tracker, speed=8.00, and one of the lighting conditions in Figure 4.1. The distribution of outlier ratio is approximately the same for all lighting conditions.

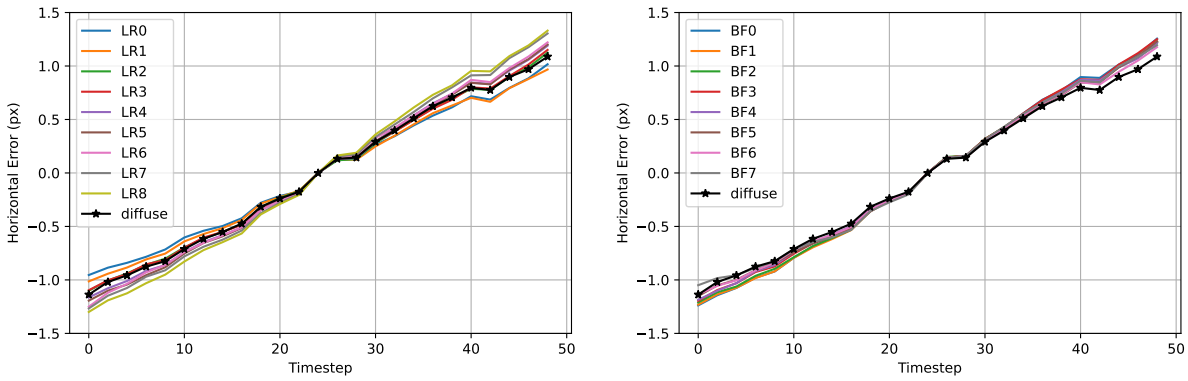


(a) Lucas-Kanade

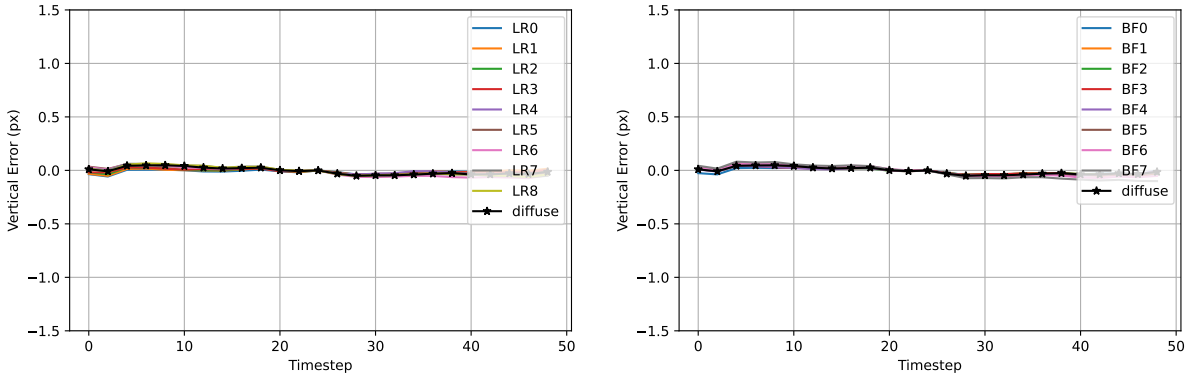


(b) Correspondence

Figure B.29: **At twelve times nominal speed, the existence of directional lighting does not affect outlier ratio.** In the box-and-whisker plots above, the orange line is the median, the green triangle is the mean, and the box extends from the first to the third quartiles. The whiskers extend up to 1.5x the length of the boxes. Each box-and-whisker plot is computed using features from all 60 scenes, one tracker, speed=12.00, and one of the lighting conditions in Figure 4.1. The distribution of outlier ratio is approximately the same for all lighting conditions.

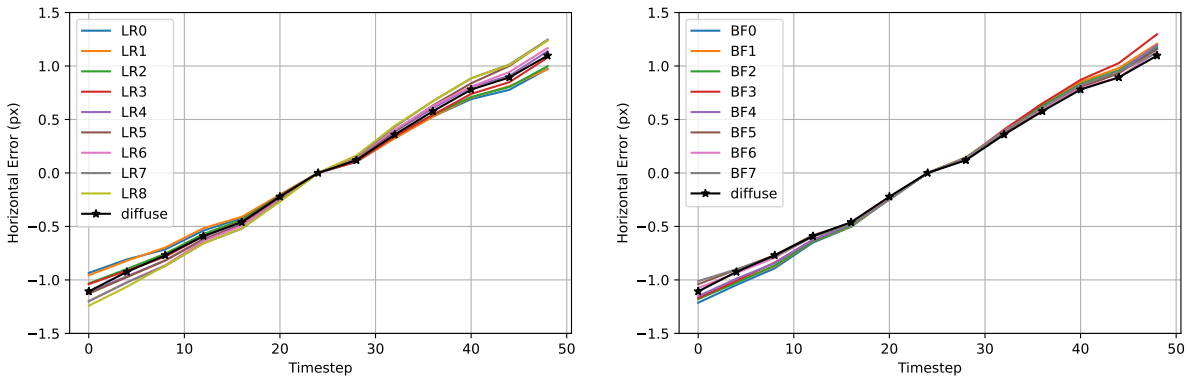


(a) Horizontal Coordinate

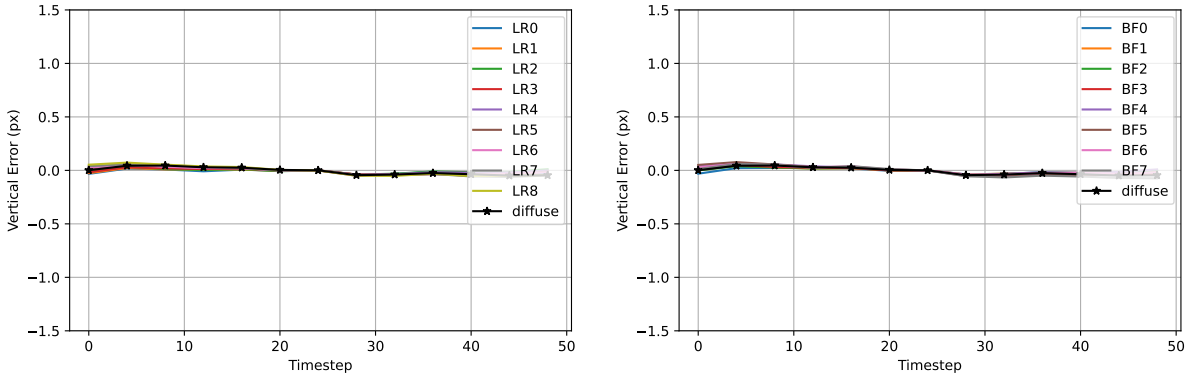


(b) Vertical Coordinate

Figure B.30: **DTU Point Features Dataset: At twice nominal speed, lighting condition does not change trends in mean error $\mu(t)$ when using the Lucas-Kanade Tracker.** We compute $\mu(t)$ at each timestep using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. The variation of $\mu(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

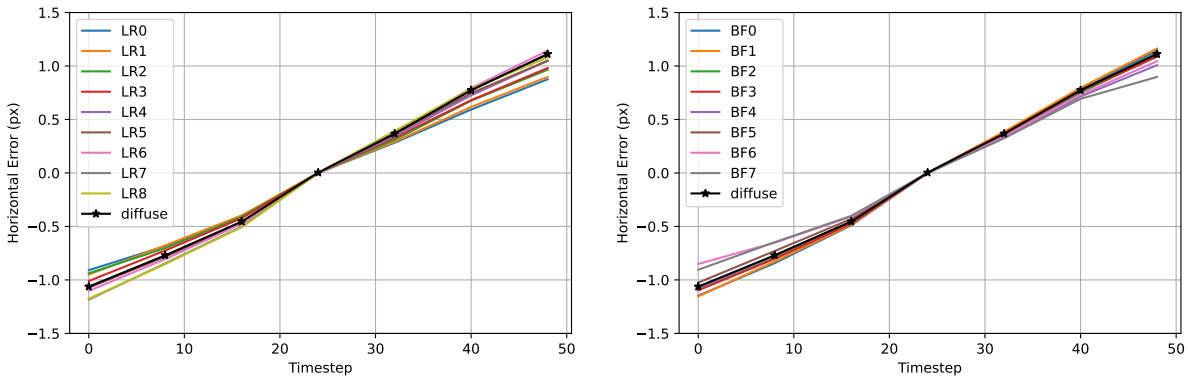


(a) Horizontal Coordinate

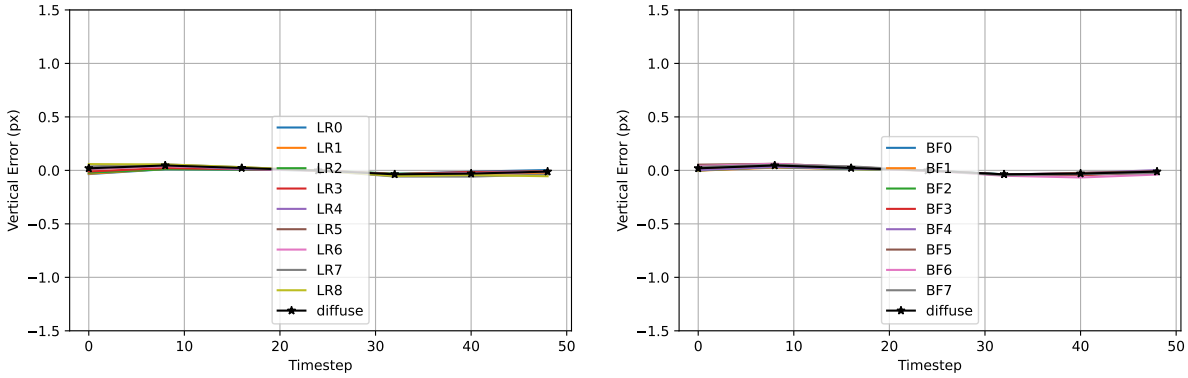


(b) Vertical Coordinate

Figure B.31: **DTU Point Features Dataset: At four times nominal speed, lighting condition does not change trends in mean error $\mu(t)$ when using the Lucas-Kanade Tracker.** We compute $\mu(t)$ at each timestep using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. The variation of $\mu(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

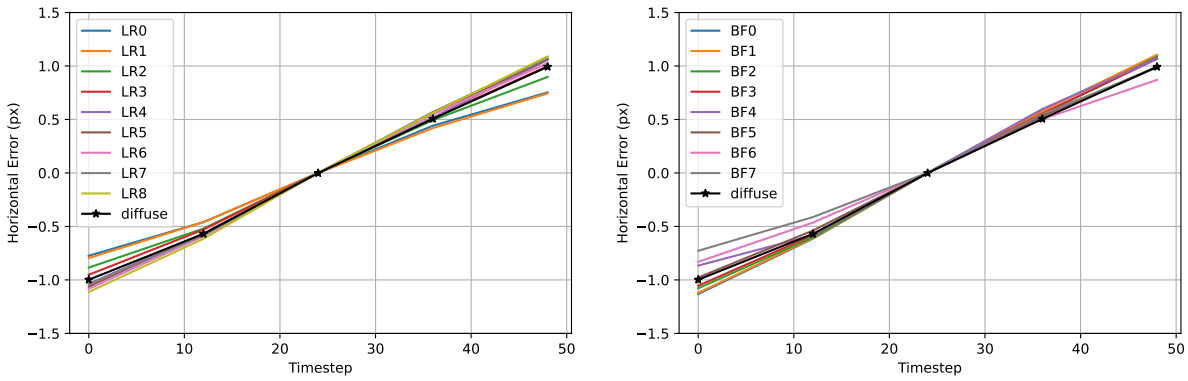


(a) Horizontal Coordinate

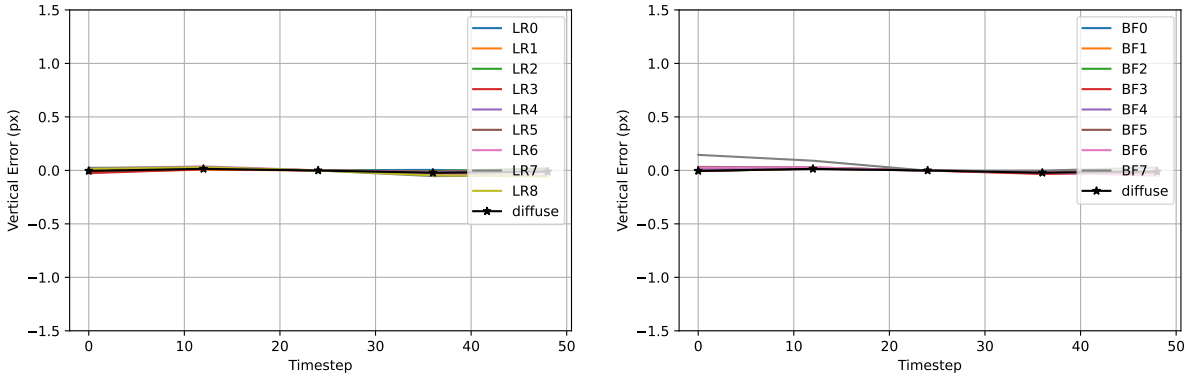


(b) Vertical Coordinate

Figure B.32: **DTU Point Features Dataset: At eight times nominal speed, lighting condition does not change trends in mean error $\mu(t)$ when using the Lucas-Kanade Tracker.** We compute $\mu(t)$ at each timestep using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. The variation of $\mu(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

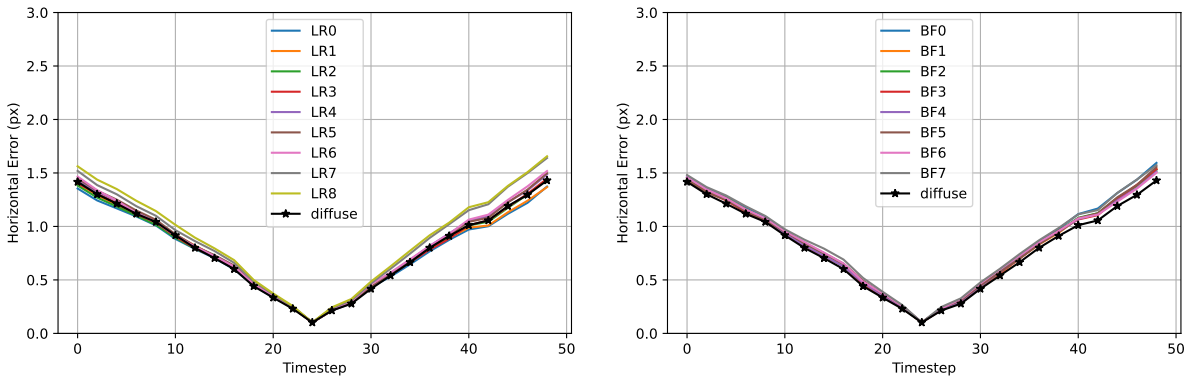


(a) Horizontal Coordinate

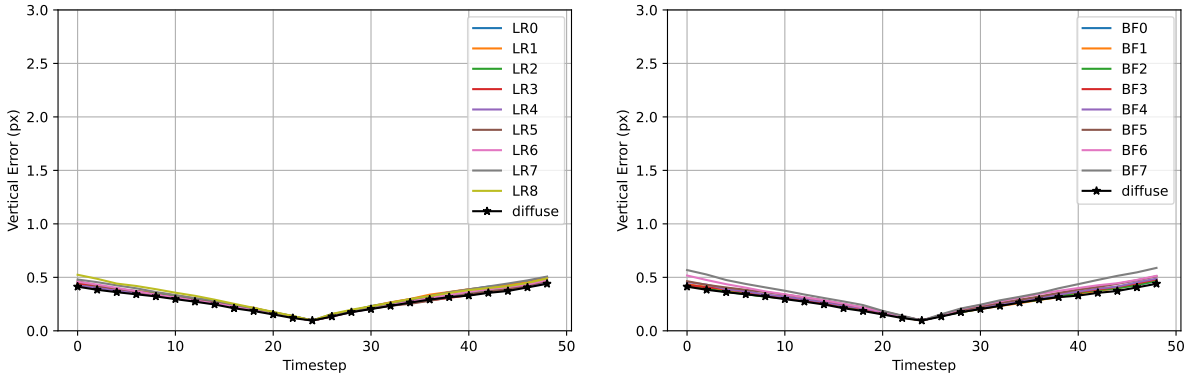


(b) Vertical Coordinate

Figure B.33: **DTU Point Features Dataset: At twelve times nominal speed, lighting condition does not change trends in mean error $\mu(t)$ when using the Lucas-Kanade Tracker.** We compute $\mu(t)$ at each timestep using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. The variation of $\mu(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

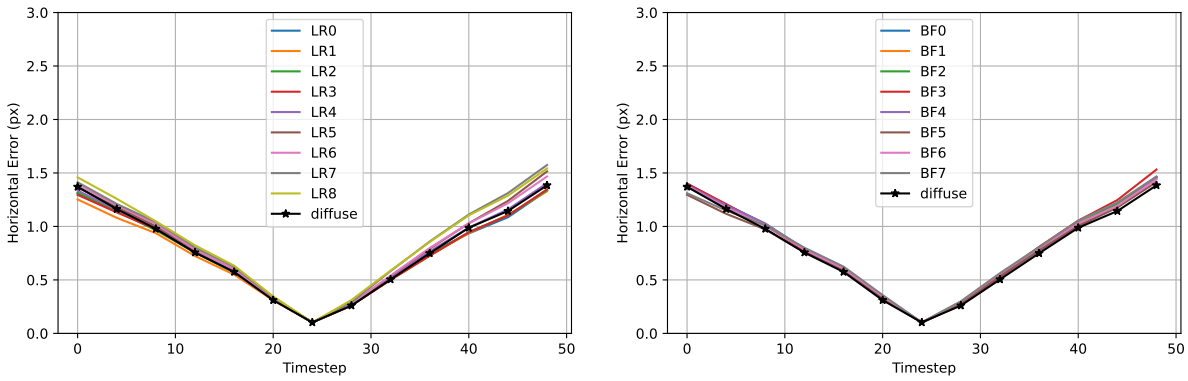


(a) Horizontal Coordinate

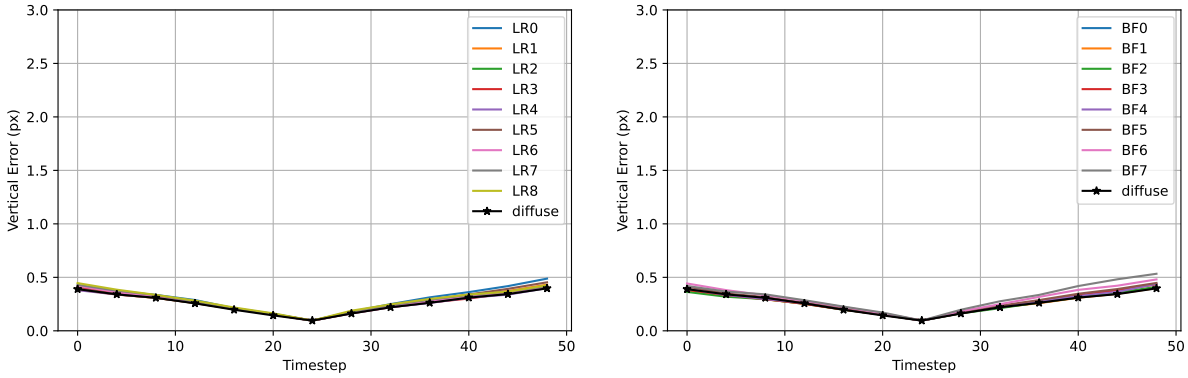


(b) Vertical Coordinate

Figure B.34: **DTU Point Features Dataset: At twice nominal speed, lighting condition does not change trends in mean absolute error $\kappa(t)$ when using the Lucas-Kanade Tracker.** We compute $\kappa(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. The variation of $\kappa(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

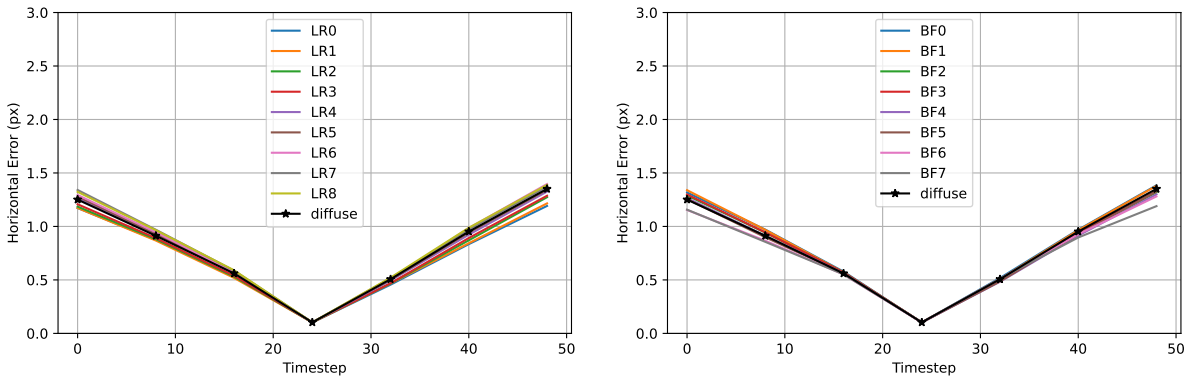


(a) Horizontal Coordinate

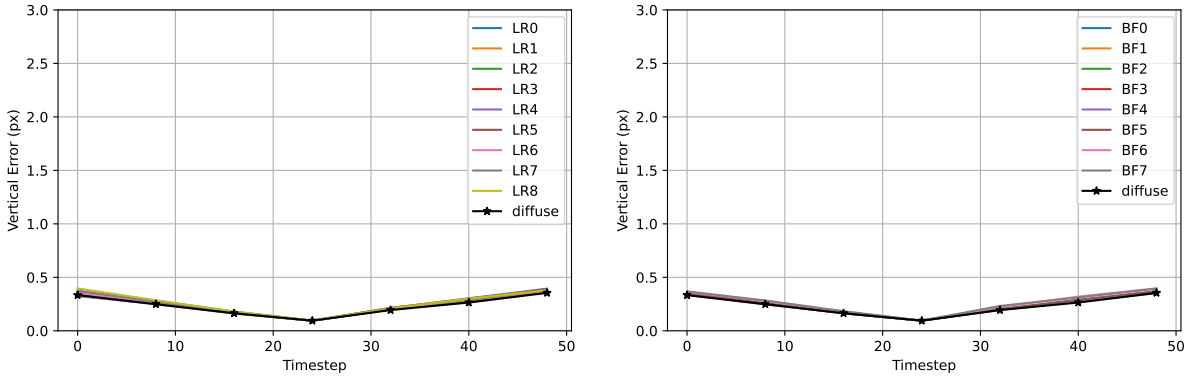


(b) Vertical Coordinate

Figure B.35: **DTU Point Features Dataset: At four times nominal speed, lighting condition does not change trends in mean absolute error $\kappa(t)$ when using the Lucas-Kanade Tracker.** We compute $\kappa(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. The variation of $\kappa(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

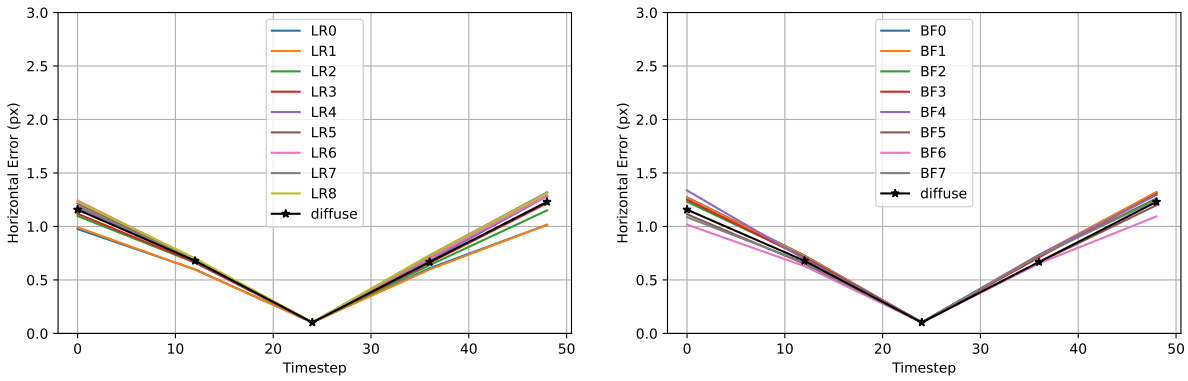


(a) Horizontal Coordinate

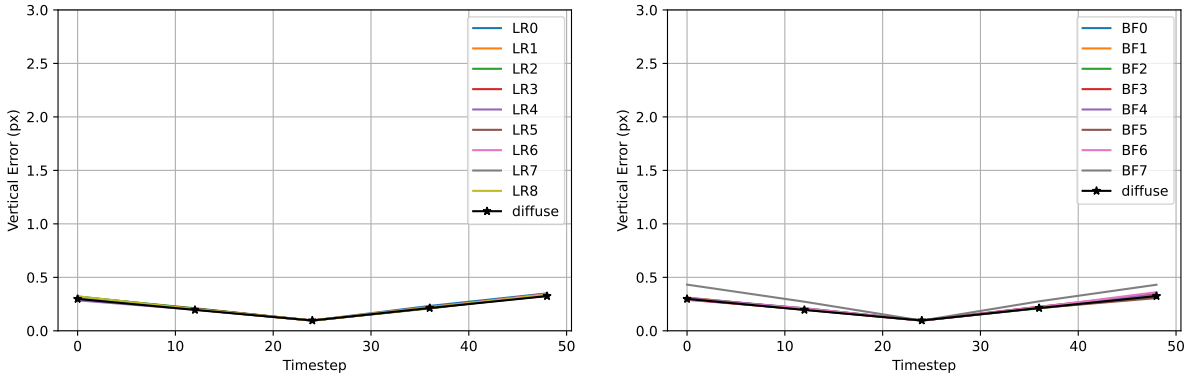


(b) Vertical Coordinate

Figure B.36: **DTU Point Features Dataset: At eight times nominal speed, lighting condition does not change trends in $\kappa(t)$ when using the Lucas-Kanade Tracker.** We compute $\kappa(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. The variation of $\kappa(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is small because changes between adjacent frames are small whether or not the scene contains directional lighting.

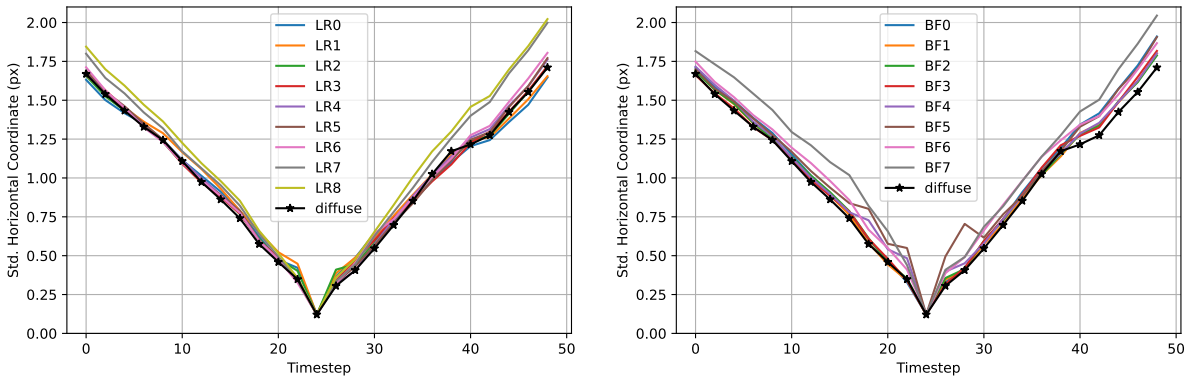


(a) Horizontal Coordinate

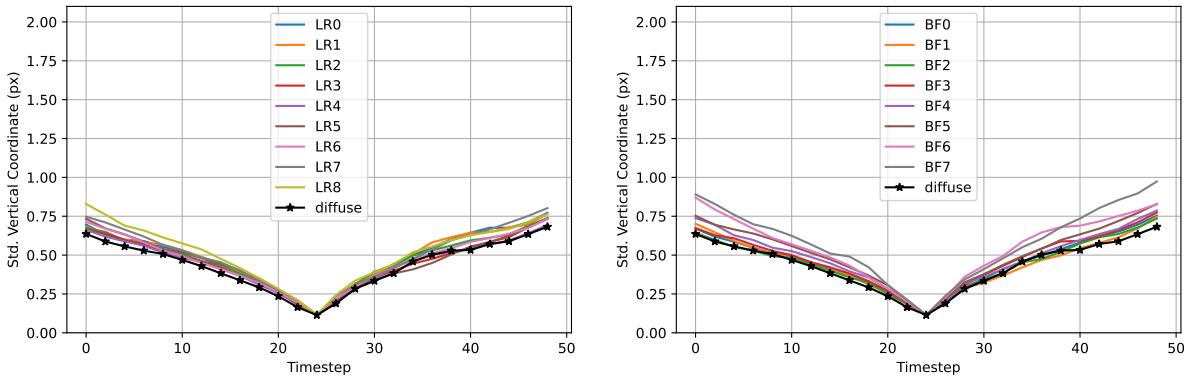


(b) Vertical Coordinate

Figure B.37: **DTU Point Features Dataset: At twelve times nominal speed, lighting condition does not change trends in mean absolute error $\kappa(t)$ when using the Lucas-Kanade Tracker.** We compute $\kappa(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. The variation of $\kappa(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

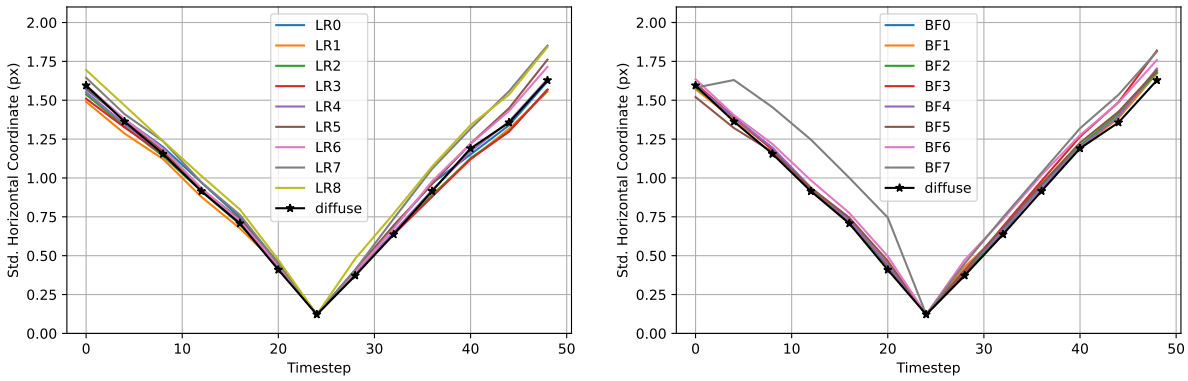


(a) Horizontal Coordinate

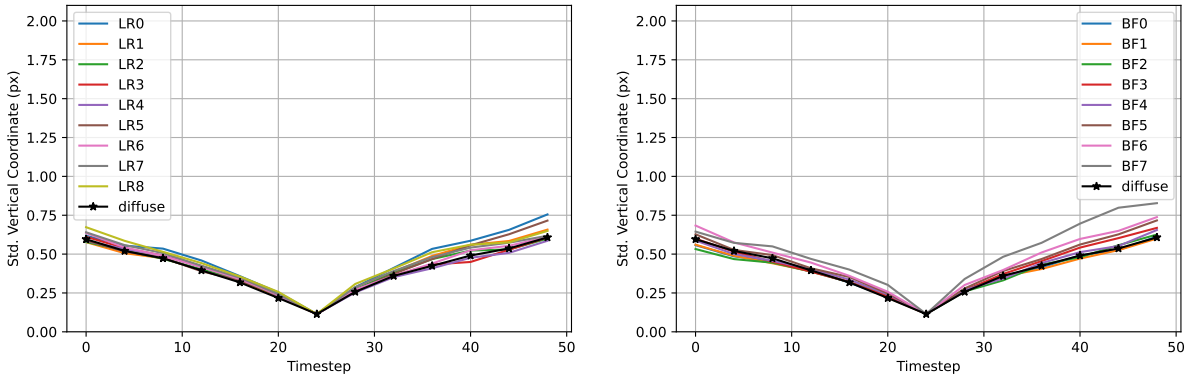


(b) Vertical Coordinate

Figure B.38: **DTU Point Features Dataset: At twice nominal speed, lighting condition does not change trends in covariance $\Sigma(t)$ when using the Lucas-Kanade Tracker.** We compute $\Sigma(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. The variation of $\Sigma(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

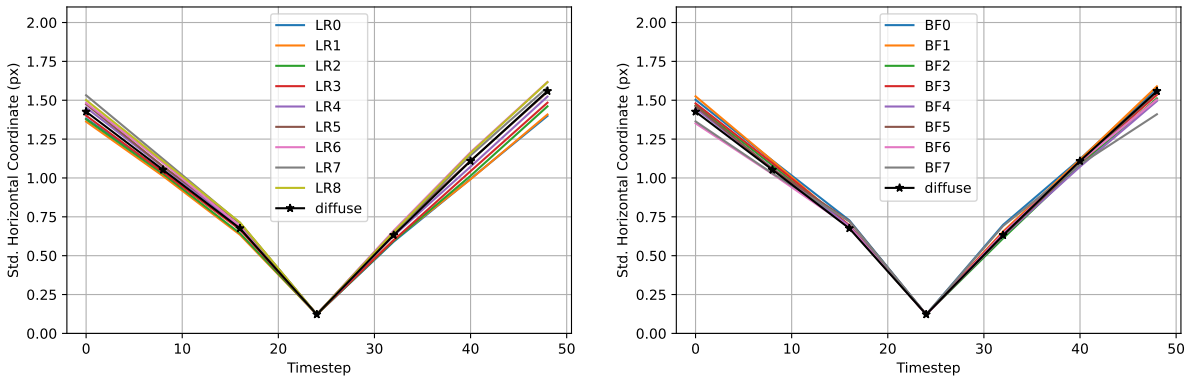


(a) Horizontal Coordinate

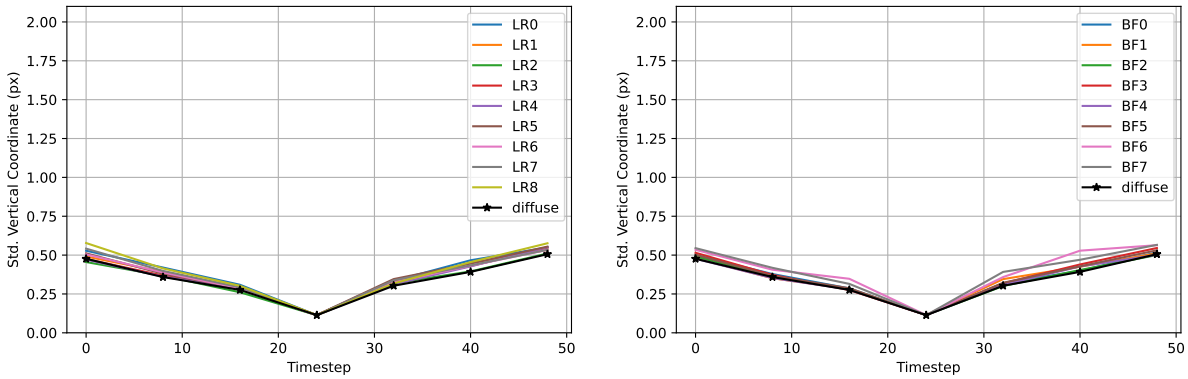


(b) Vertical Coordinate

Figure B.39: **DTU Point Features Dataset: At four times nominal speed, lighting condition does not change trends in covariance $\Sigma(t)$ when using the Lucas-Kanade Tracker.** We compute $\Sigma(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. The variation of $\Sigma(t)$ due to the existence of directional lighting is less than 10 percent of the variation common to all plotted lines for all but one lighting condition. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting. The larger-than average covariance for lighting condition BF7 is caused by a single scene where feature tracking fails.

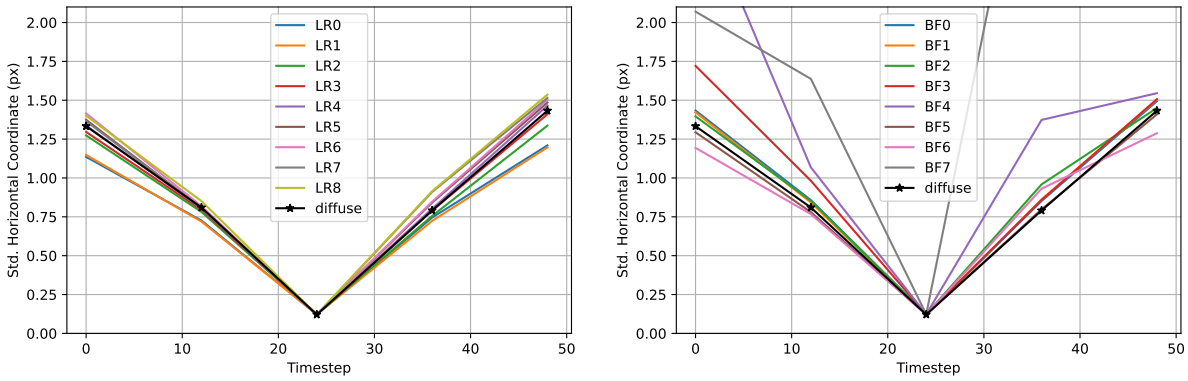


(a) Horizontal Coordinate

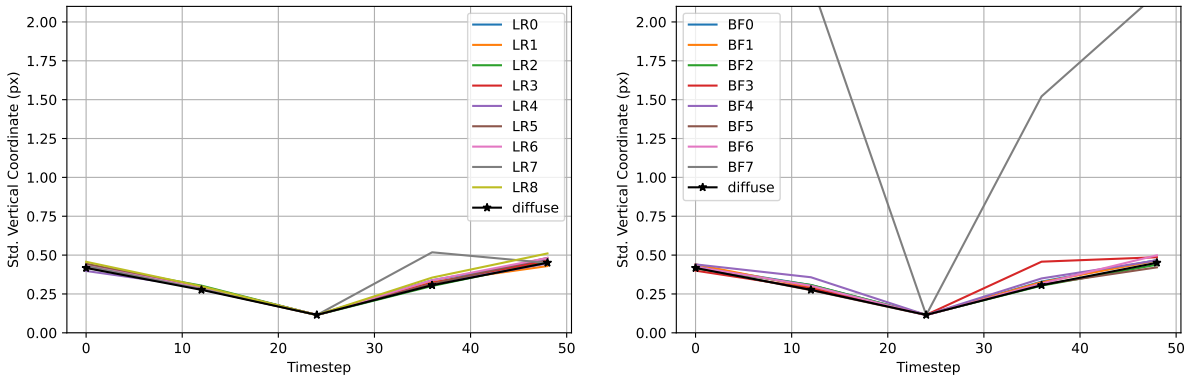


(b) Vertical Coordinate

Figure B.40: **DTU Point Features Dataset: At eight times nominal speed, lighting condition does not change trends in covariance $\Sigma(t)$ when using the Lucas-Kanade Tracker.** We compute $\Sigma(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. The variation of $\Sigma(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

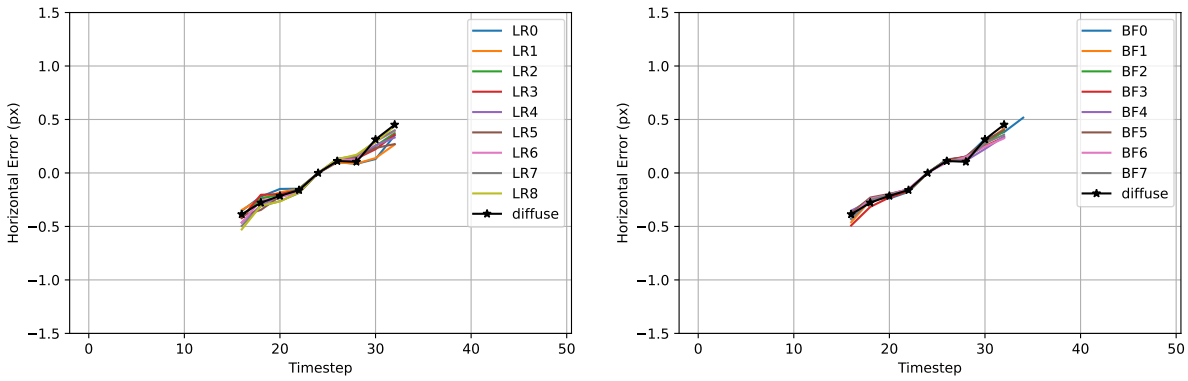


(a) Horizontal Coordinate

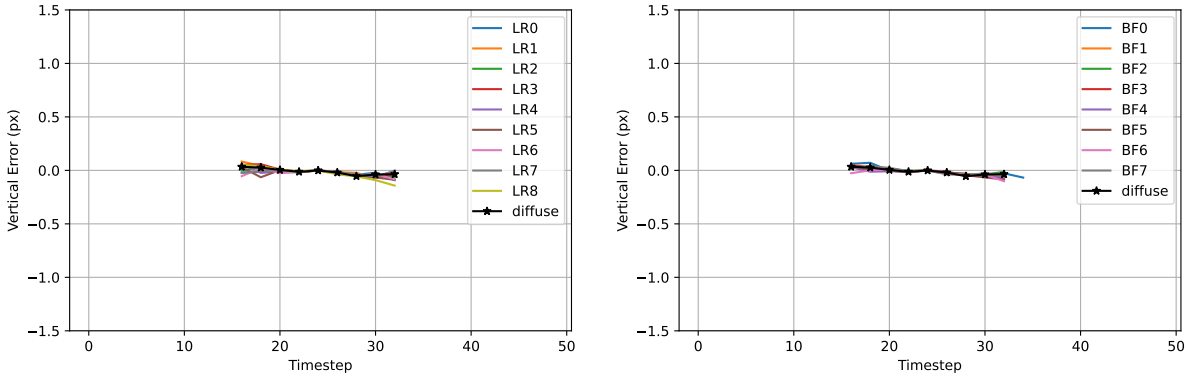


(b) Vertical Coordinate

Figure B.41: **DTU Point Features Dataset: At twelve times nominal speed, lighting condition does not change trends in covariance $\Sigma(t)$ when using the Lucas-Kanade Tracker.** We compute $\Sigma(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. At twelve times nominal speed, tracking failures cause large covariances to appear for some lighting conditions. Otherwise, the variation of $\Sigma(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

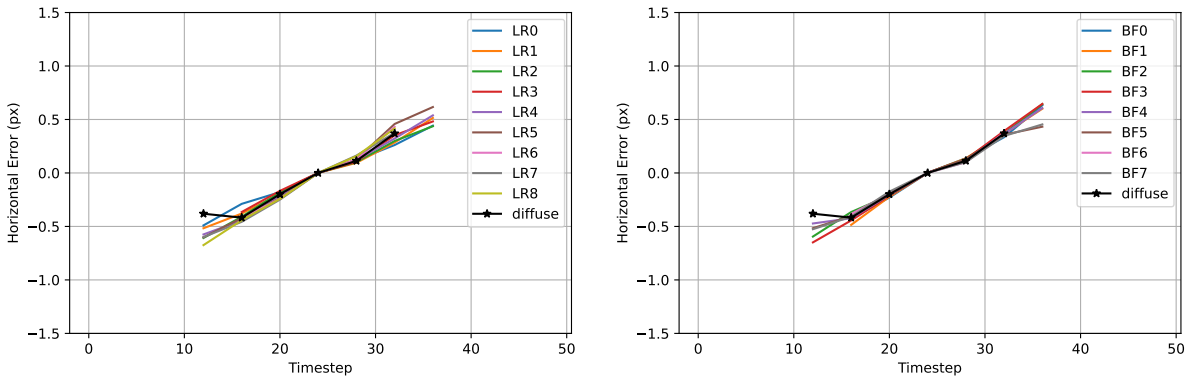


(a) Horizontal Coordinate

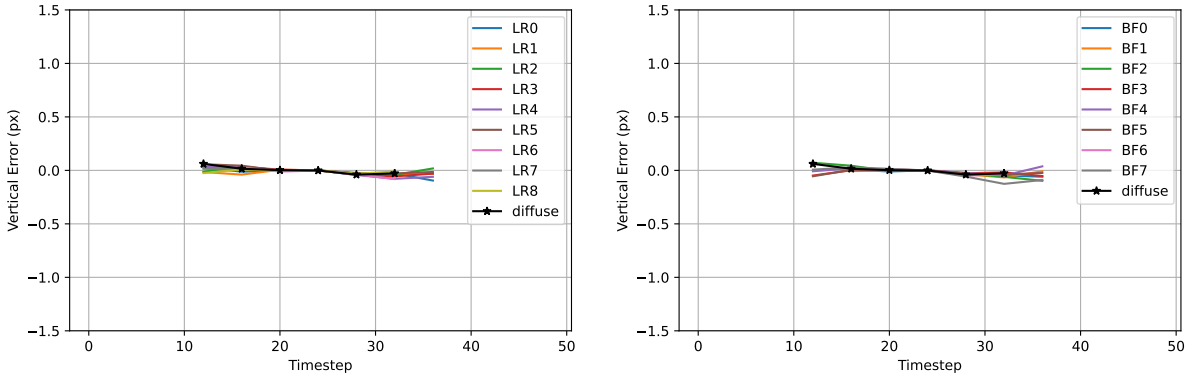


(b) Vertical Coordinate

Figure B.42: **DTU Point Features Dataset: At twice nominal speed, lighting condition does not change trends in mean error $\mu(t)$ when using the Correspondence Tracker.** We compute $\mu(t)$ at each timestep using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Lines are limited to timesteps containing at least 100 features. The variation of $\mu(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

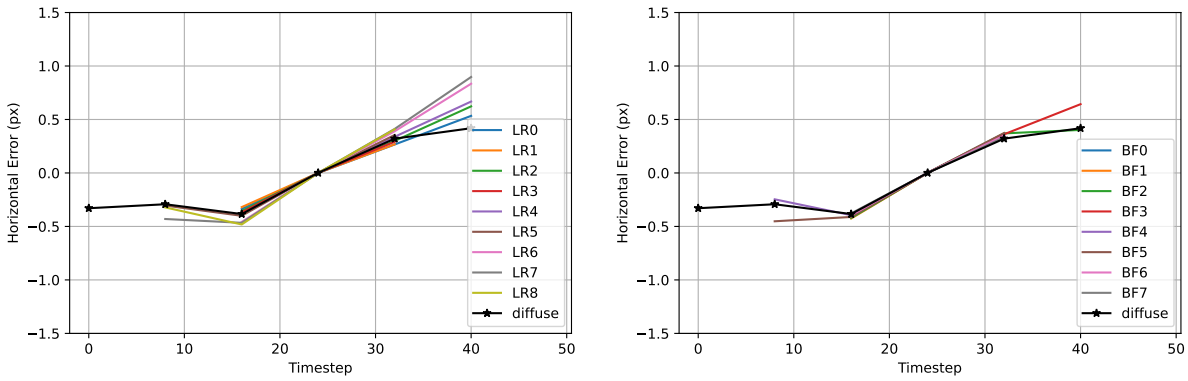


(a) Horizontal Coordinate

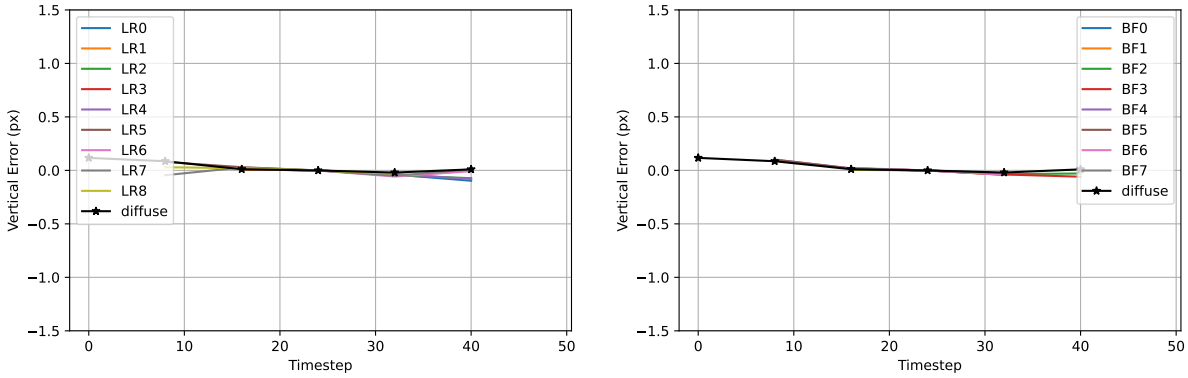


(b) Vertical Coordinate

Figure B.43: **DTU Point Features Dataset: At four times nominal speed, lighting condition does not change trends in mean error $\mu(t)$ when using the Correspondence Tracker.** We compute $\mu(t)$ at each timestep using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Lines are limited to timesteps containing at least 100 features. The variation of $\mu(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

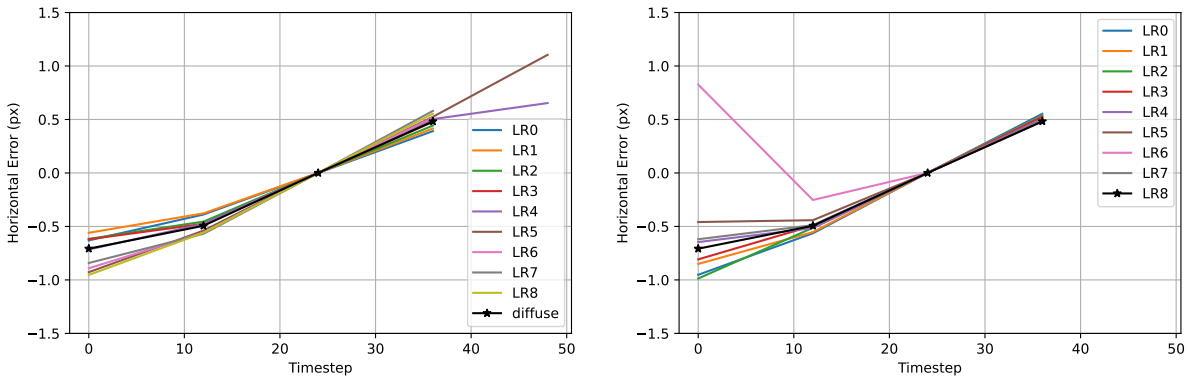


(a) Horizontal Coordinate

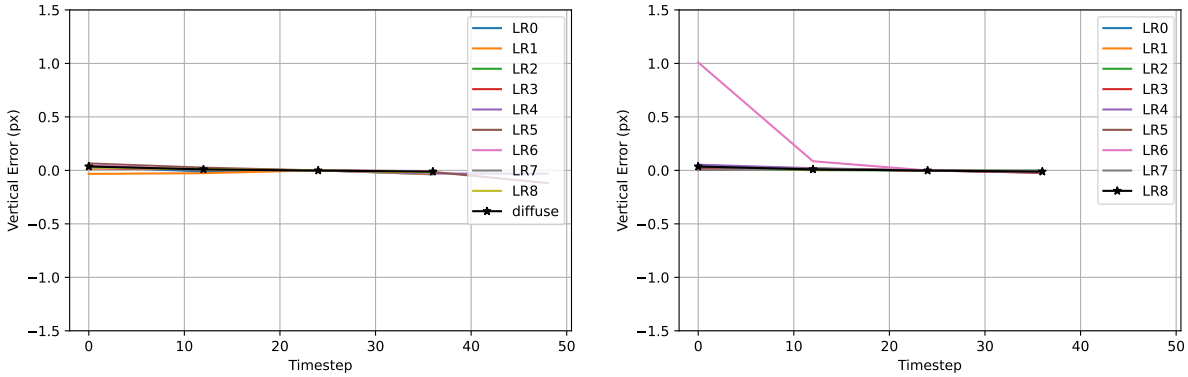


(b) Vertical Coordinate

Figure B.44: **DTU Point Features Dataset: At eight times nominal speed, lighting condition does not change trends in mean error $\mu(t)$ when using the Correspondence Tracker.** We compute $\mu(t)$ at each timestep using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Lines are limited to timesteps containing at least 100 features. The variation of $\mu(t)$ due to the existence of directional lighting is smaller than the variation common to all plotted lines. The effect of directional lighting is small because changes between adjacent frames are small whether or not the scene contains directional lighting.

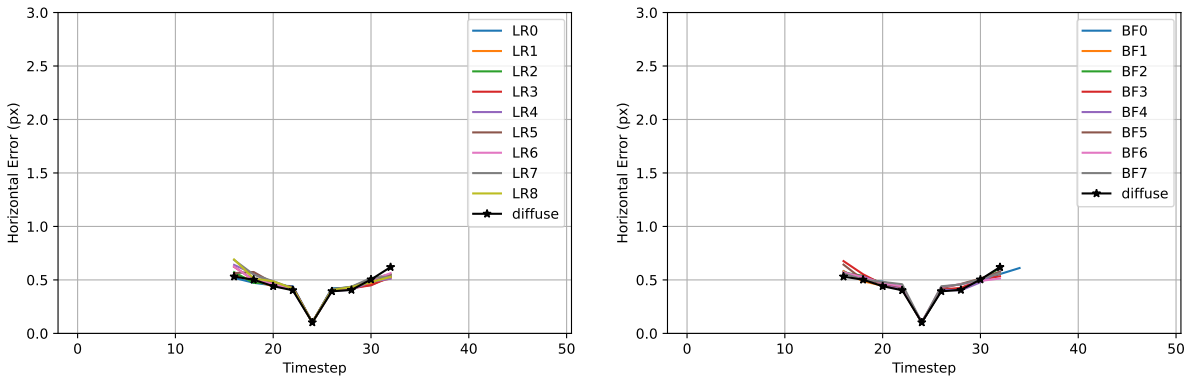


(a) Horizontal Coordinate

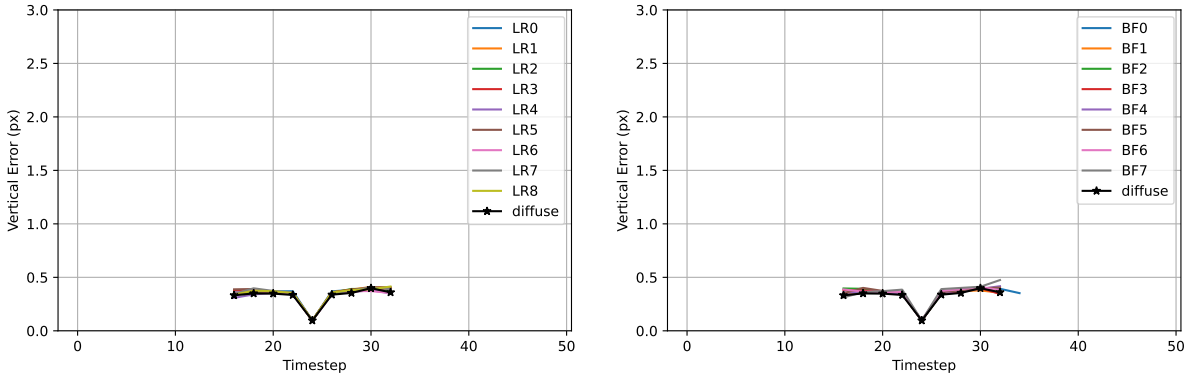


(b) Vertical Coordinate

Figure B.45: **DTU Point Features Dataset: At twelve times nominal speed, lighting condition does not change trends in mean error $\mu(t)$ when using the Correspondence Tracker.** We compute $\mu(t)$ at each timestep using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Lines are limited to timesteps containing at least 100 features. With the exception of one lighting condition, the variation of $\mu(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting. The large variation in lighting condition LR6 is caused by tracking failures.

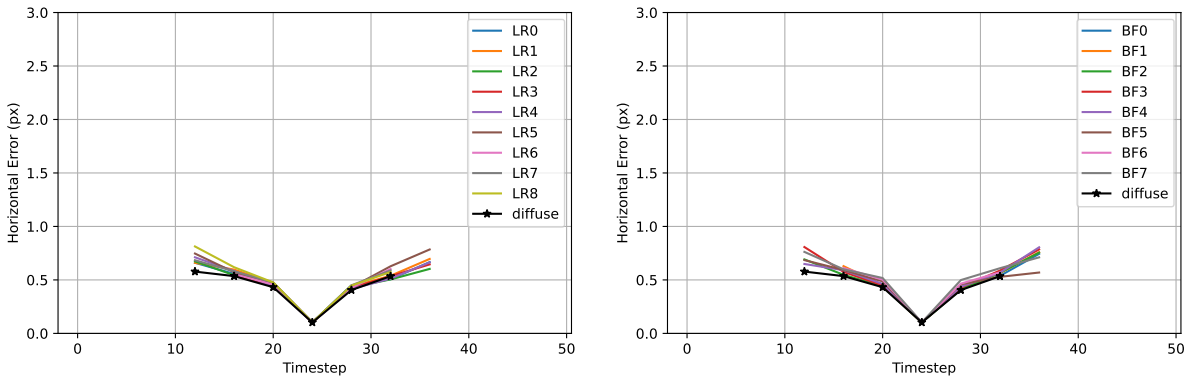


(a) Horizontal Coordinate

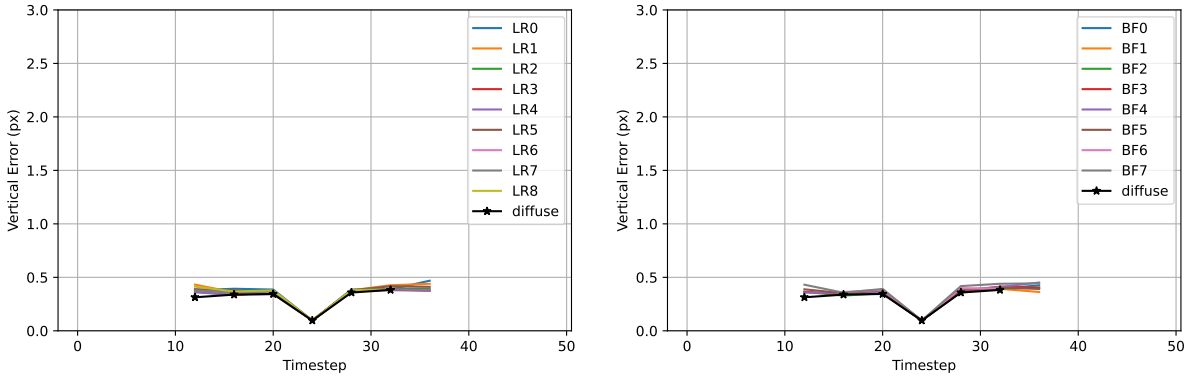


(b) Vertical Coordinate

Figure B.46: **DTU Point Features Dataset: At twice nominal speed, lighting condition does not change trends in mean absolute error $\kappa(t)$ when using the Correspondence Tracker.** We compute $\kappa(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Lines are limited to timesteps containing at least 100 features. The variation of $\kappa(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

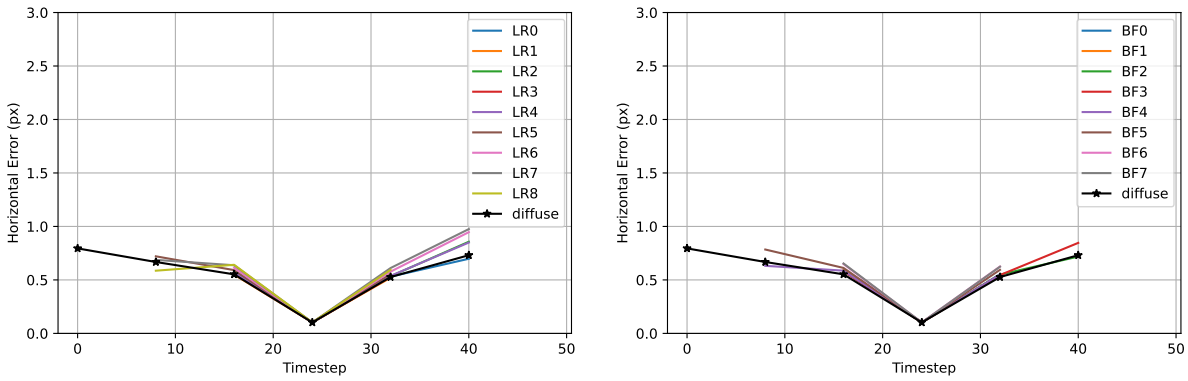


(a) Horizontal Coordinate

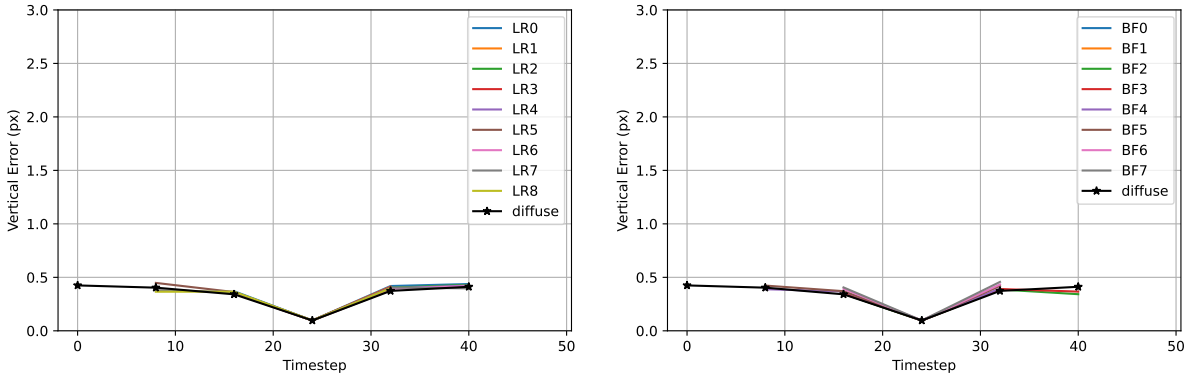


(b) Vertical Coordinate

Figure B.47: **DTU Point Features Dataset: At four times nominal speed, lighting condition does not change trends in mean absolute error $\kappa(t)$ remains independent of lighting condition when using the Correspondence Tracker.** We compute $\kappa(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Lines are limited to timesteps containing at least 100 features. There are no significant differences between lines. The effect of directional lighting is small because changes from frame-to-frame are small.

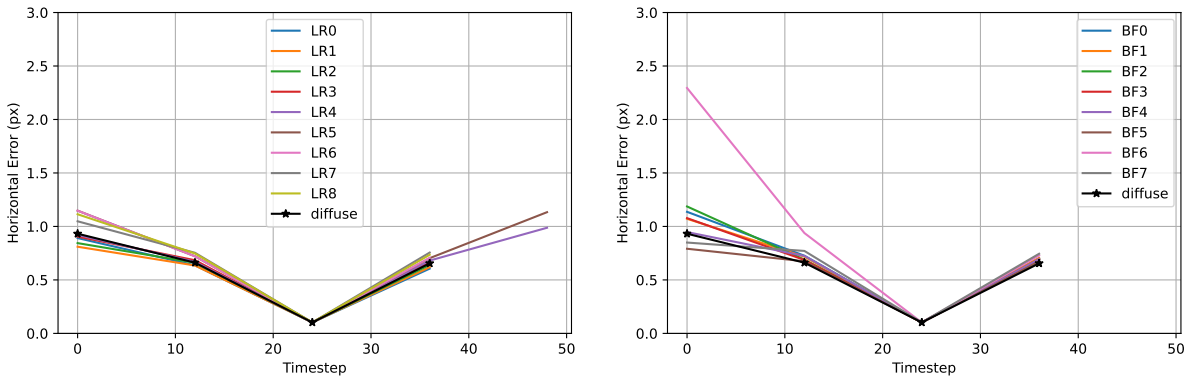


(a) Horizontal Coordinate

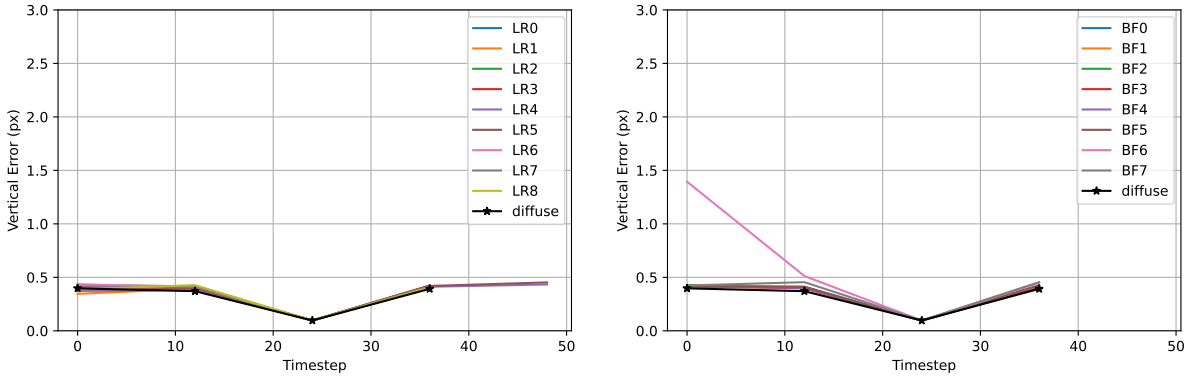


(b) Vertical Coordinate

Figure B.48: **DTU Point Features Dataset: At eight times nominal speed, lighting condition does not change trends in mean absolute error $\kappa(t)$ when using the Correspondence Tracker.** We compute $\kappa(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Lines are limited to timesteps containing at least 100 features. The variation of $\kappa(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

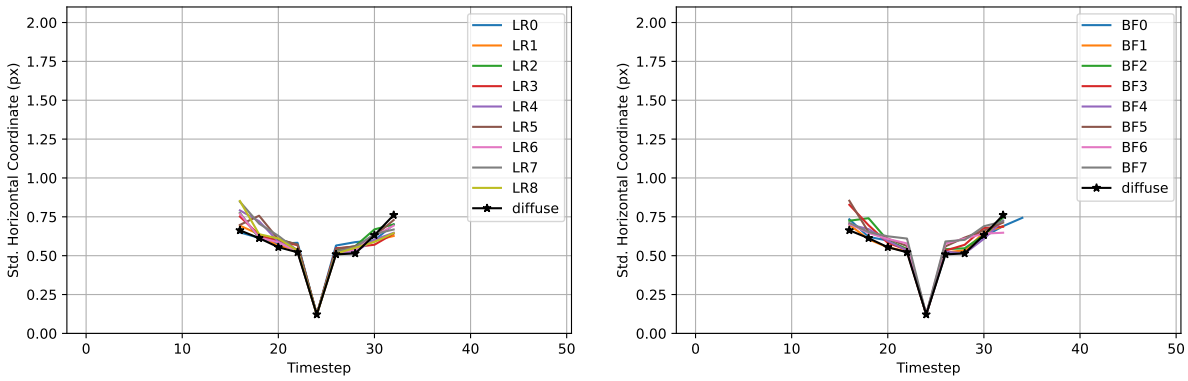


(a) Horizontal Coordinate

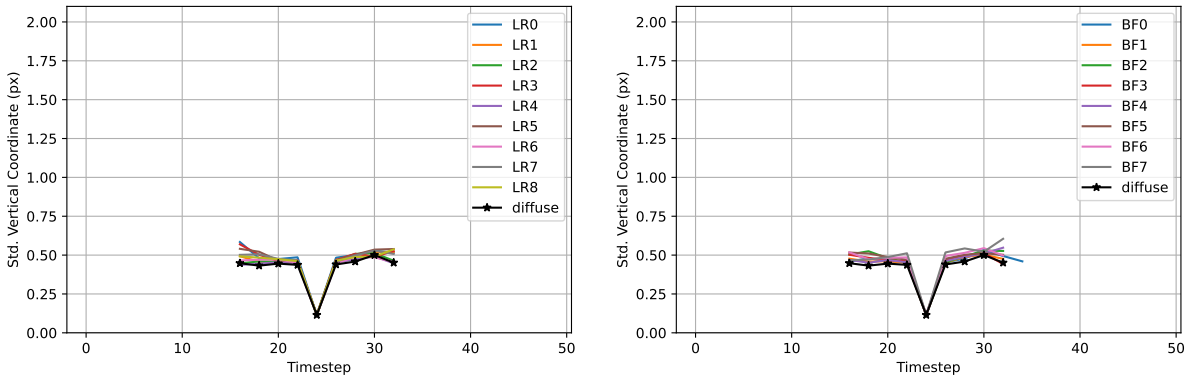


(b) Vertical Coordinate

Figure B.49: **DTU Point Features Dataset: At twelve times nominal speed, lighting condition does not change trends in mean absolute error $\kappa(t)$ when using the Correspondence Tracker.** We compute $\kappa(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Lines are limited to timesteps containing at least 100 features. With the exception of lighting condition BF6, the variation of $\kappa(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

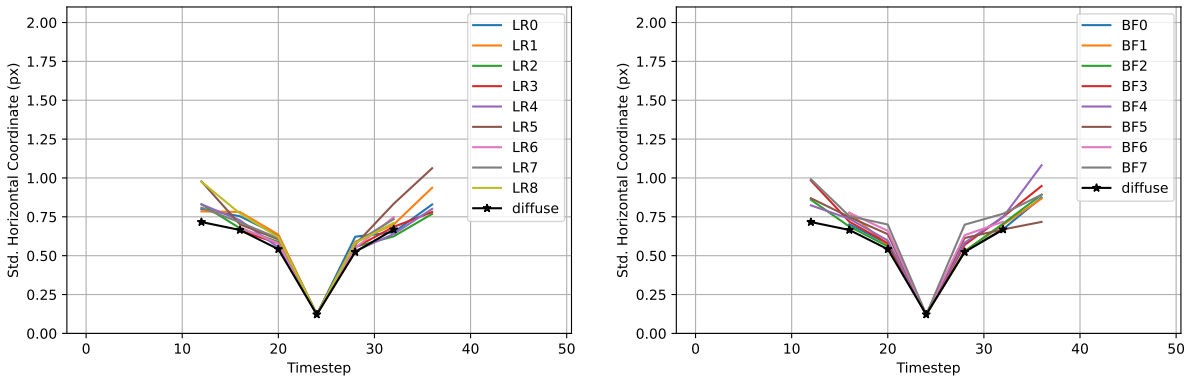


(a) Horizontal Coordinate

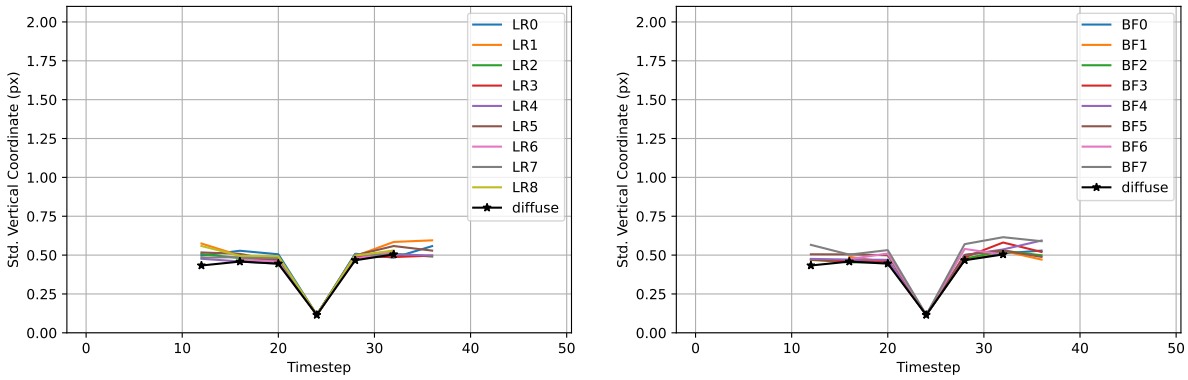


(b) Vertical Coordinate

Figure B.50: **DTU Point Features Dataset: At twice nominal speed, lighting condition does not change trends in covariance $\Sigma(t)$ when using the Correspondence Tracker.** We compute $\Sigma(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Timesteps are limited to those with at least 100 features. The variation of $\Sigma(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

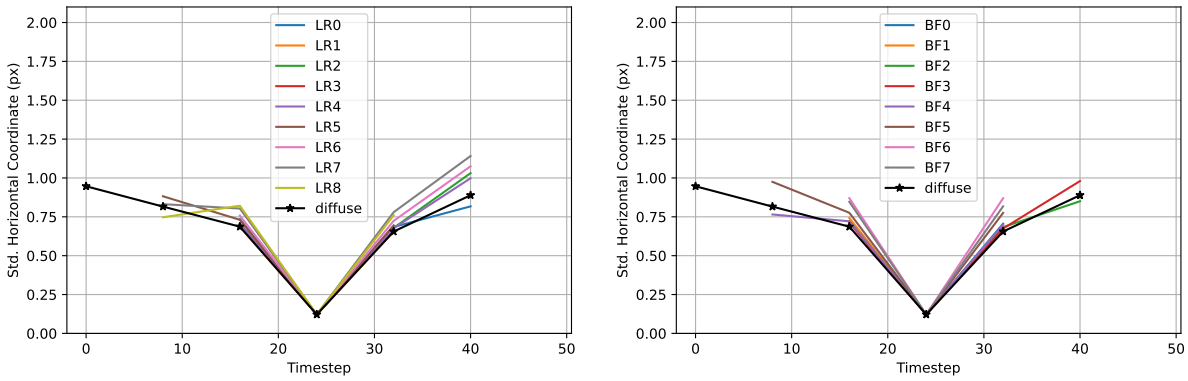


(a) Horizontal Coordinate

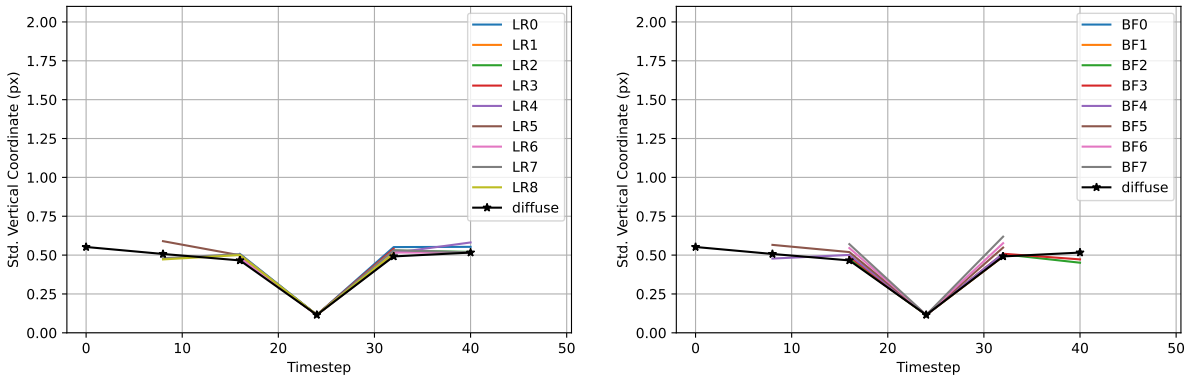


(b) Vertical Coordinate

Figure B.51: **DTU Point Features Dataset: At four times nominal speed, lighting condition does not change trends in covariance $\Sigma(t)$ when using the Correspondence Tracker.** We compute $\Sigma(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Timesteps are limited to those with at least 100 features. The variation of $\Sigma(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

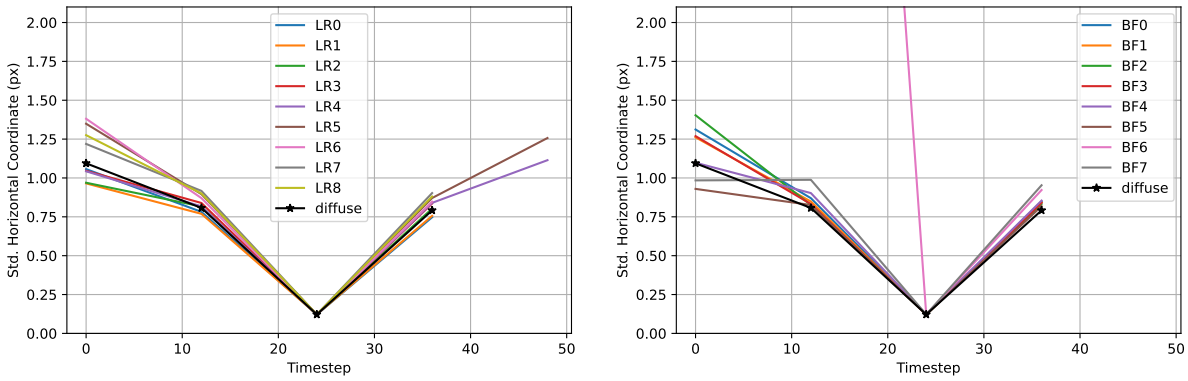


(a) Horizontal Coordinate

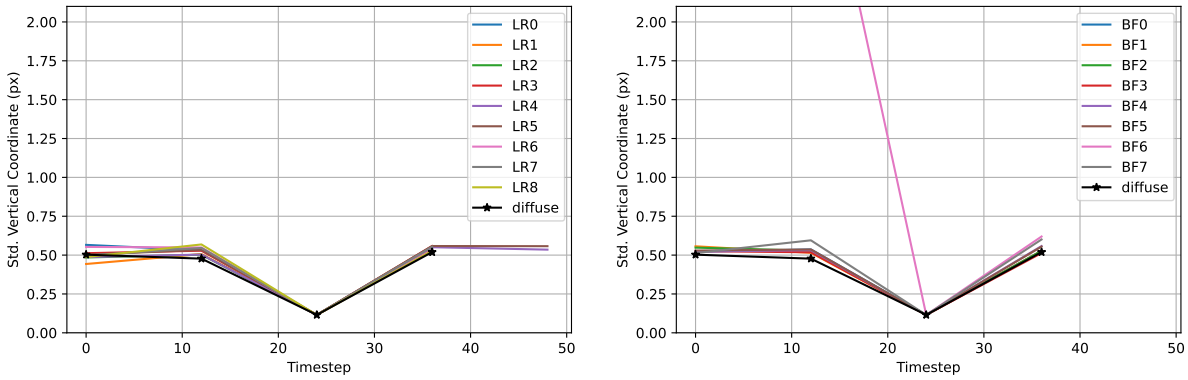


(b) Vertical Coordinate

Figure B.52: **DTU Point Features Dataset: At eight times nominal speed, lighting condition does not change trends in covariance $\Sigma(t)$ when using the Correspondence Tracker.** We compute $\Sigma(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Timesteps are limited to those with at least 100 features. The variation of $\Sigma(t)$ due to the existence of directional lighting is at most 10 percent of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.



(a) Horizontal Coordinate



(b) Vertical Coordinate

Figure B.53: **DTU Point Features Dataset: At twelve times nominal speed, lighting condition does not change trends in covariance $\Sigma(t)$ when using the Correspondence Tracker.** We compute $\Sigma(t)$ using diffuse lighting (black lines) and each of the directional lighting conditions listed in Figure 4.1 using all tracks from all 60 scenes. Timesteps are limited to those with at least 100 features. With the exception of feature track failures in lighting condition BF6, the variation of $\Sigma(t)$ due to the existence of directional lighting is a fraction of the variation common to all plotted lines. The effect of directional lighting is relatively small because changes between adjacent frames are small whether or not the scene contains directional lighting.

B.2 Supporting Figures for KITTI Vision Suite

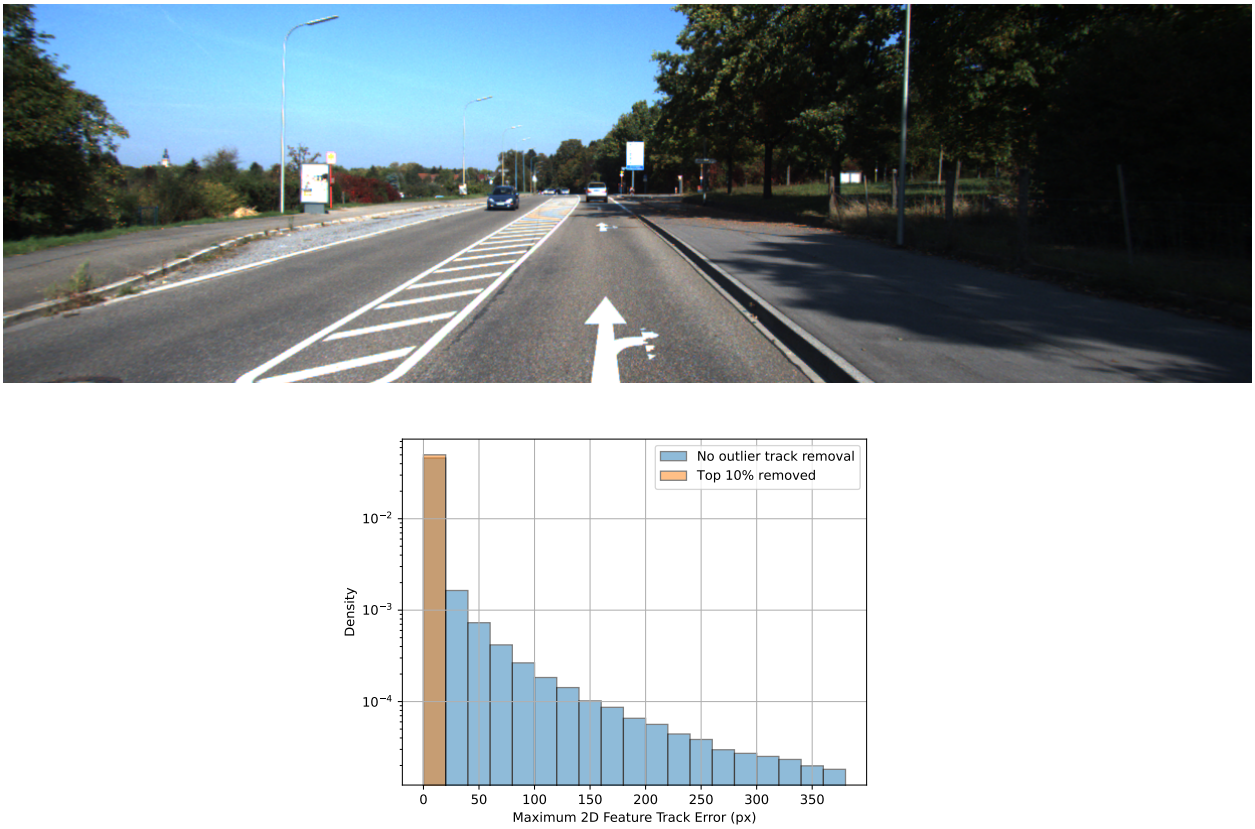


Figure B.54: **KITTI Dataset: We will throw out the 10% of tracks from each scene with the most error.** The bottom figure plots the histogram density of the maximum L2 error of all feature tracks of a single scene in log scale. The corresponding scene is pictured on top. The outlier errors are caused by noisy data in the depth image collection process.

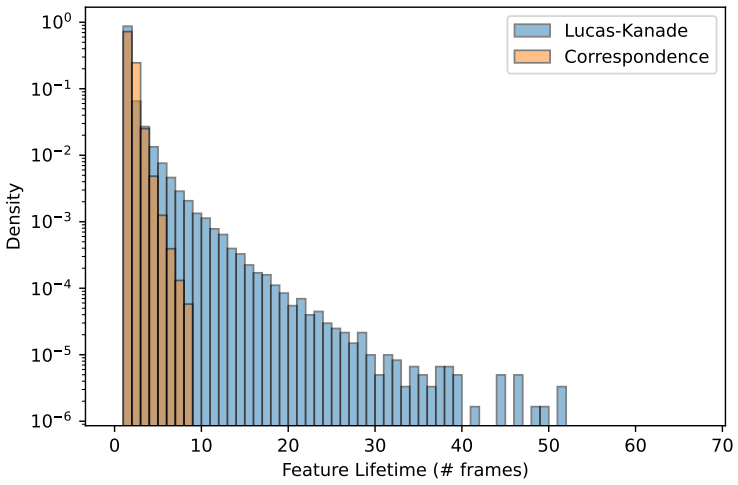
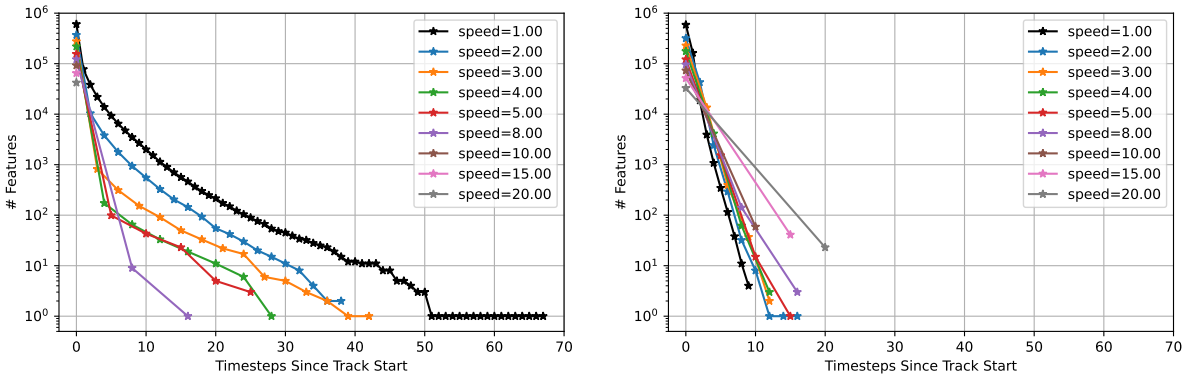


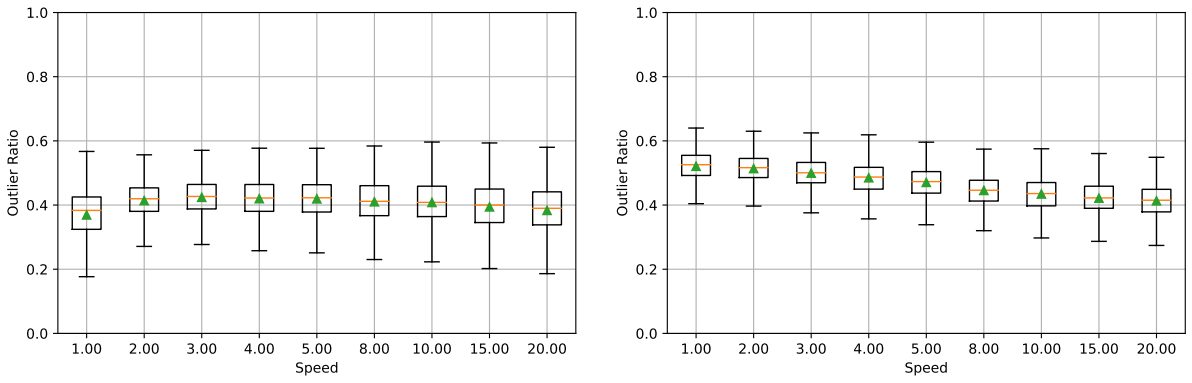
Figure B.55: **KITTI Dataset: Most features live for less than five frames.** The distribution of feature lifetimes is plotted as a log-scale histogram for both the Lucas-Kanade and Correspondence-Based Tracker at nominal speed. The Lucas-Kanade Tracker produces a long tail of features with longer lifetimes. Features with long-lifetimes are those far away from the car's camera, in the center of the image.



(a) Lucas-Kanade

(b) Correspondence

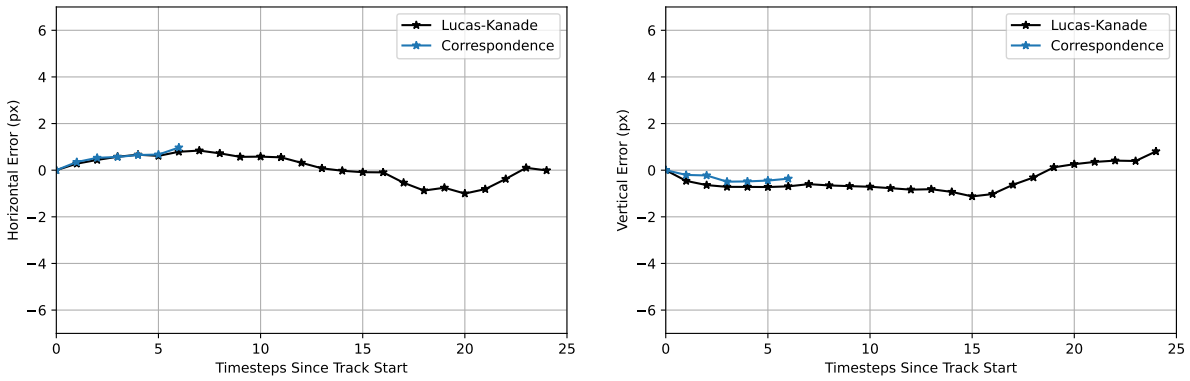
Figure B.56: Feature lifetime is plotted on the horizontal axis. The vertical axis, in log scale, shows the number of features in all 28 scenes that were tracked for at least that many frames. In both plots the number of features drops very fast. Note that for speeds greater than 8.00, the Lucas-Kanade tracker fails to match any features past one frame. **In subsequent analyses on the KITTI dataset, we only compute mean errors and covariances at timesteps with at least 100 features. We also only analyze speeds 1.00, 2.00, and 3.00 because higher speeds would otherwise be limited to $\leq$ two timesteps.**



(a) Lucas-Kanade

(b) Correspondence

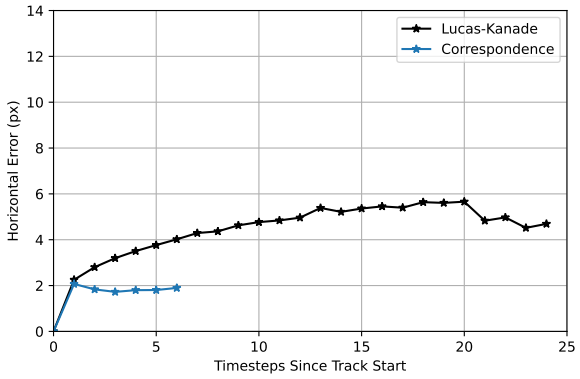
Figure B.57: **KITTI Dataset: Outlier ratios are above 40 percent.** Outlier ratios per frame are shown as box-and-whisker plots for the Lucas-Kanade tracker on the left and the Correspondence tracker on the right. For the Lucas-Kanade tracker, outlier ratios remain a constant 40 percent. For the correspondence tracker, outlier ratios are higher, around 50 percent, for lower speeds and then decrease. The decreases exists not because of improvements in feature matching with higher speeds, but because fewer features are matched at all.



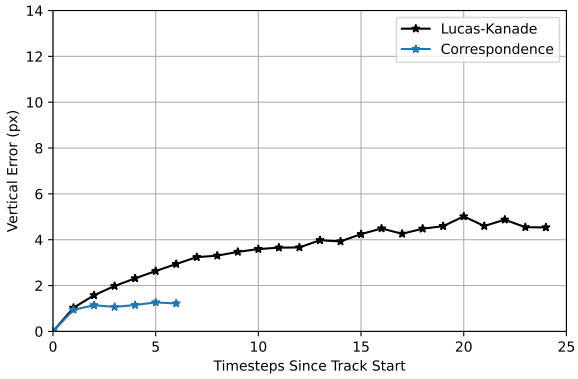
(a) $\nu(t)$, Horizontal Coordinate

(b) $\nu(t)$, Vertical Coordinate

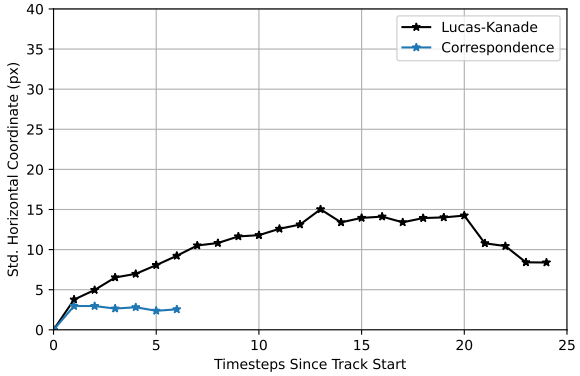
Figure B.58: **KITTI Dataset: The zero-mean assumption approximately holds for both the Lucas-Kanade Tracker and the Correspondence Tracker at nominal speed.** Lines shown are horizontal (left) and vertical (right) coordinates of mean error $\nu(t)$ calculated using tracks averaged over all scenes; calculation is cutoff at 24 frames for the Lucas-Kanade Tracker and 6 frames for the Correspondence Tracker so that averages can be computed with at least 100 features. Mean errors remain at roughly zero.



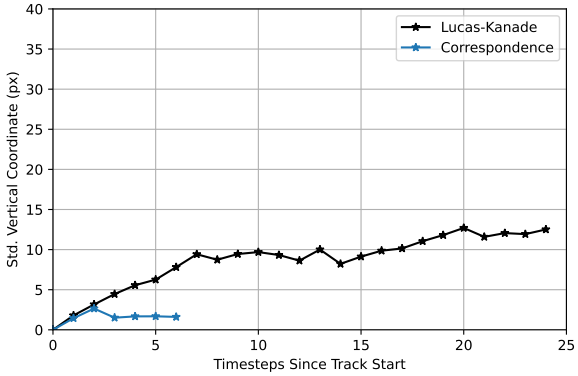
(a) $\eta(t)$, Horizontal Coordinate



(b) $\eta(t)$, Vertical Coordinate

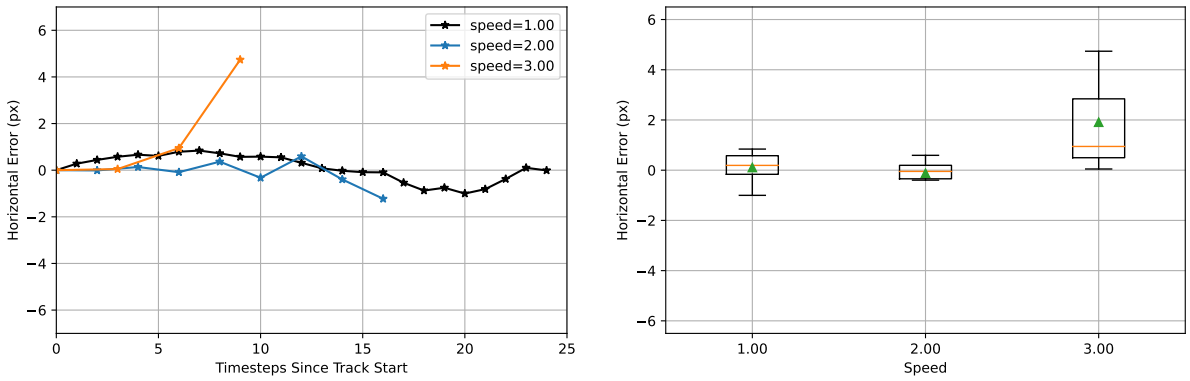


(c) $\Phi(t)$, Horizontal Coordinate

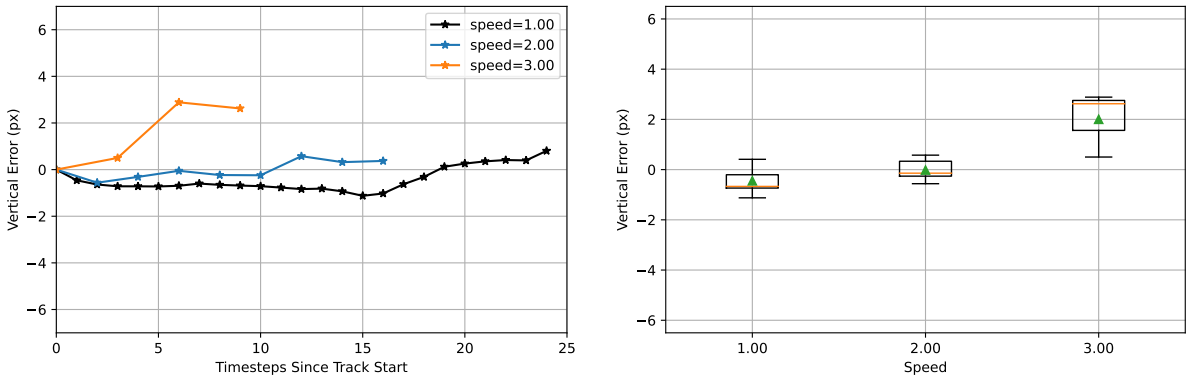


(d) $\Phi(t)$, Vertical Coordinate

Figure B.59: **KITTI Dataset: The Lucas-Kanade Tracker drifts more than the Correspondence Tracker in all directions.** Lines shown are horizontal (left column) and vertical (right column) coordinates of mean absolute error $\eta(t)$ (top row) and covariance $\Phi(t)$ (bottom row) calculated using tracks averaged over all scenes; calculation is cutoff at 24 frames for Lucas-Kanade Tracker and 6 frames for the Correspondence Tracker so that averages can be computed with at least 100 features. Both mean absolute error and covariance are roughly constant when using the Correspondence Tracker. On the other hand, both drift slightly upwards and then level off when using the Lucas-Kanade Tracker.

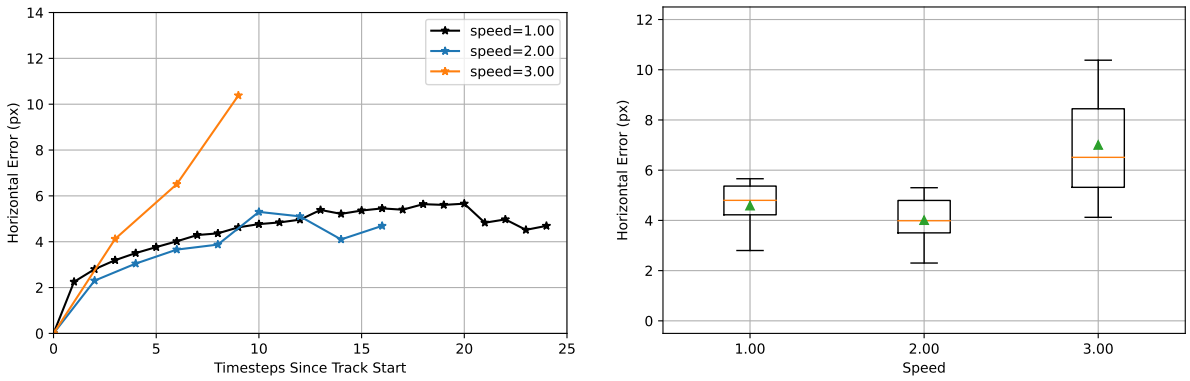


(a) $\nu(t)$, Horizontal Coordinate

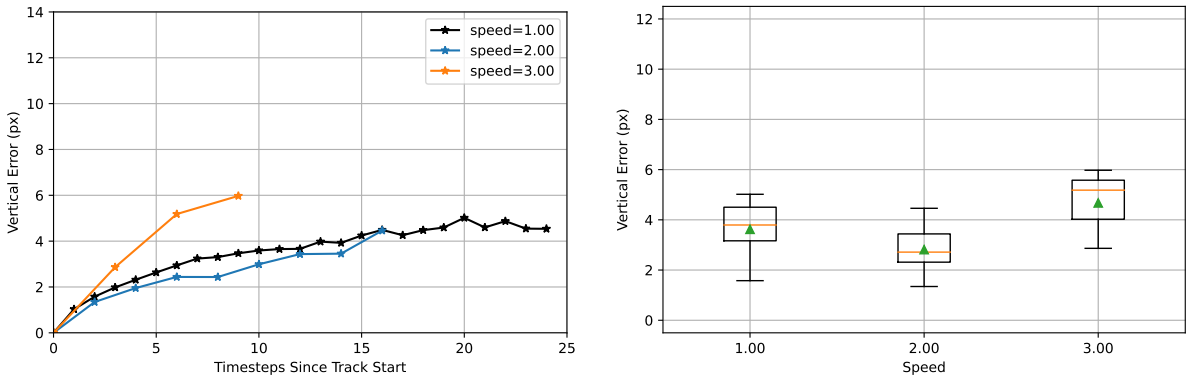


(b) $\nu(t)$, Vertical Coordinate

Figure B.60: **KITTI Dataset: Mean tracking errors increase with speed when using the Lucas-Kanade Tracker.** The left column contains plots of the horizontal (top row) and vertical (bottom row) components of the mean tracking error $\nu(t)$ at each timestep t after initial feature detection at multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate values of each line for $t > 0$ in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. The mean and median values of the horizontal and vertical coordinates of $\nu(t)$ increases by about two pixels when speed is increased from 2.00 to 3.00. There is no such increase in $\nu(t)$ when speed is increased from 1.00 to 2.00.

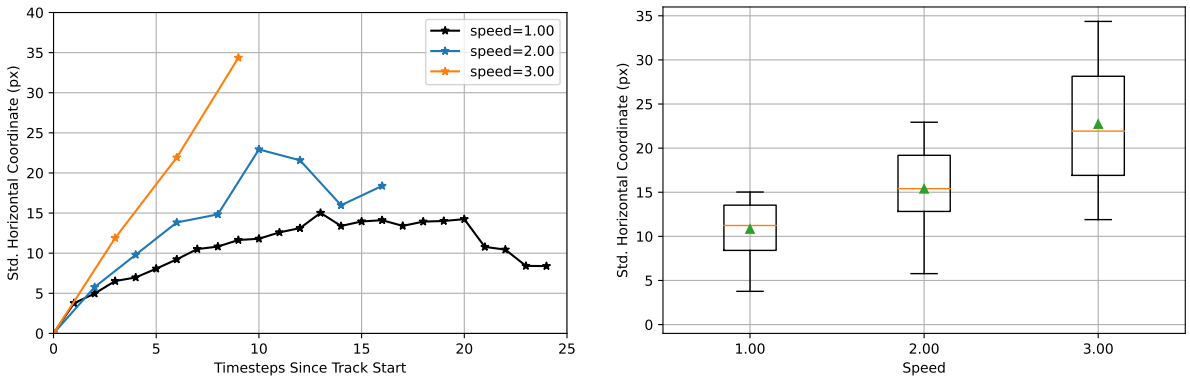


(a) $\eta(t)$, Horizontal Coordinate

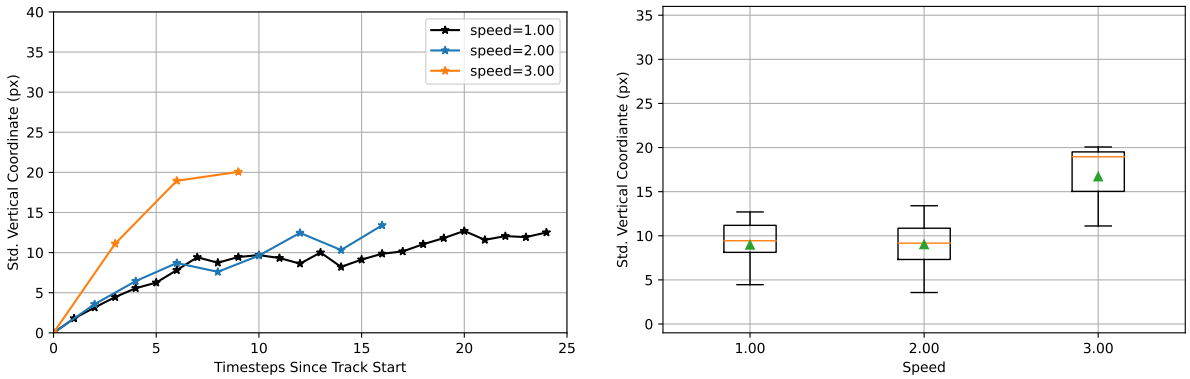


(b) $\eta(t)$, Vertical Coordinate

Figure B.61: **KITTI Dataset: Mean absolute errors increase with speed when using the Lucas-Kanade Tracker.** The left column contains plots of the horizontal (top row) and vertical (bottom row) components of the mean absolute error $\eta(t)$ at each timestep t after initial feature detection at multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate values of each line for $t > 0$ in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. The mean and median values of $\eta(t)$ jump when speed is increased from 2.00 to 3.00. Left column plots show that $\eta(t)$ is approximately unchanged when speed is increased from 1.00 to 2.00. Since the box plot for speed=1.00 contains more points at larger values of t than the box plot for speed=2.00, the mean and median values in the box plot decrease.

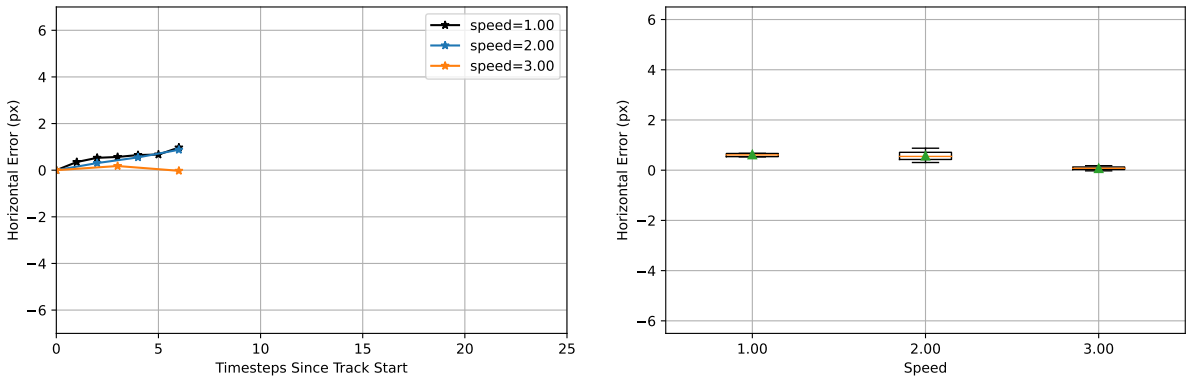


(a) $\Phi(t)$, Horizontal Coordinates

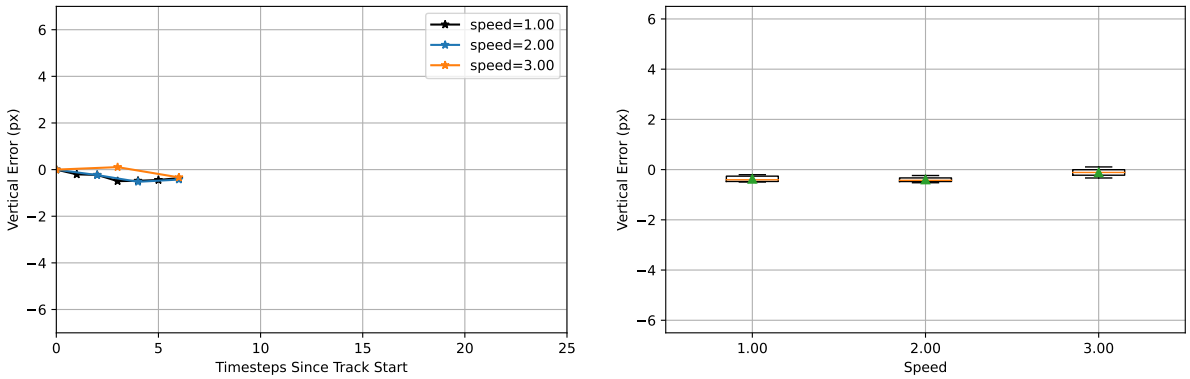


(b) $\Phi(t)$, Vertical Coordinates

Figure B.62: **KITTI Dataset: Covariances increase with speed when using the Lucas-Kanade Tracker.** The left column contains plots of the horizontal (top row) and vertical (bottom row) components of the covariance $\Phi(t)$ at each timestep t after initial feature detection at multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate values of each line for $t > 0$ in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. We see a linear increase in covariance in the horizontal coordinate with speed. The increase in the vertical coordinate follows the same trend noted in Figures B.60 and B.61.

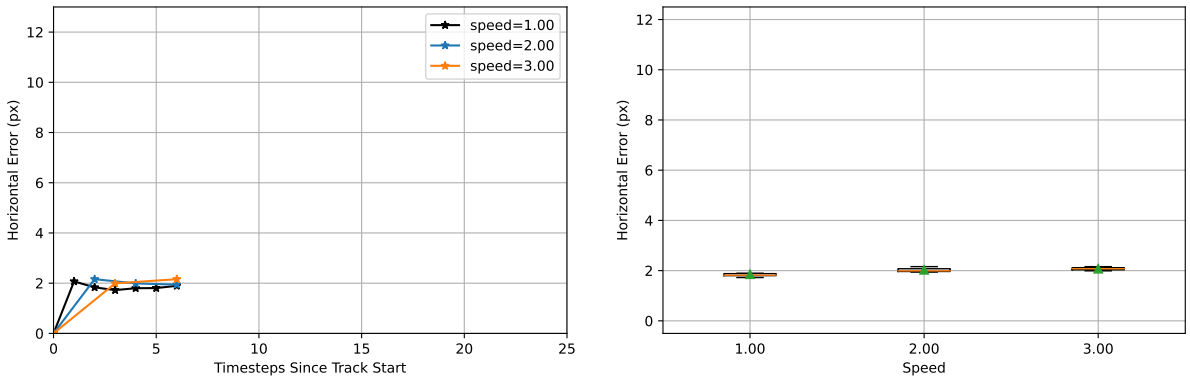


(a) $\nu(t)$, Horizontal Coordinate

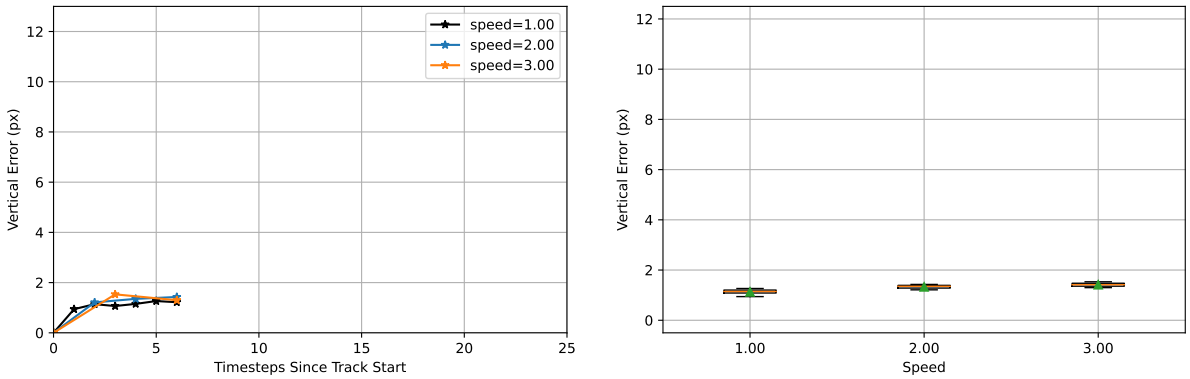


(b) $\nu(t)$, Vertical Coordinate

Figure B.63: **KITTI Dataset: Mean errors are unaffected by speed when using the Correspondence Tracker.** The left column contains plots of the horizontal (top row) and vertical (bottom row) components of the mean tracking error $\nu(t)$ at each timestep t after initial feature detection at multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate values of each line for $t > 0$ in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. Compared to the results for the Lucas-Kanade Tracker in Figure B.60, mean errors do not change when speed is increased from 1.00 to 3.00.

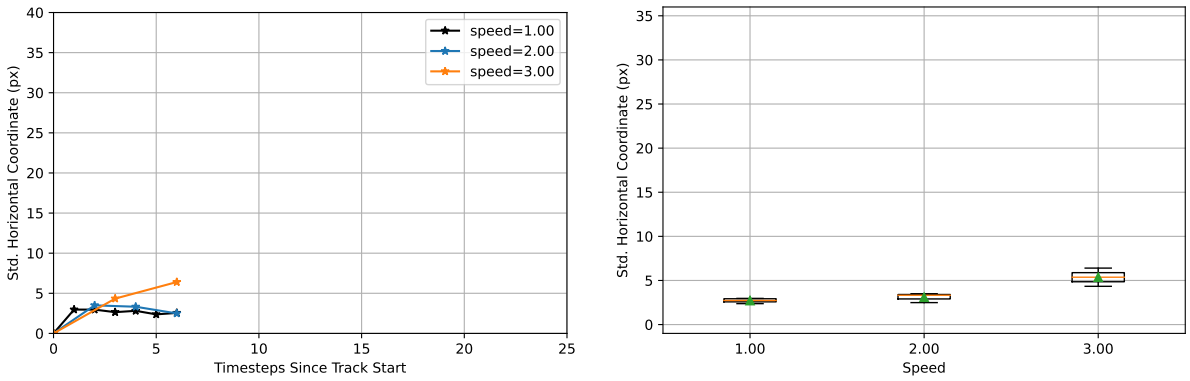


(a) $\eta(t)$, Horizontal Coordinate

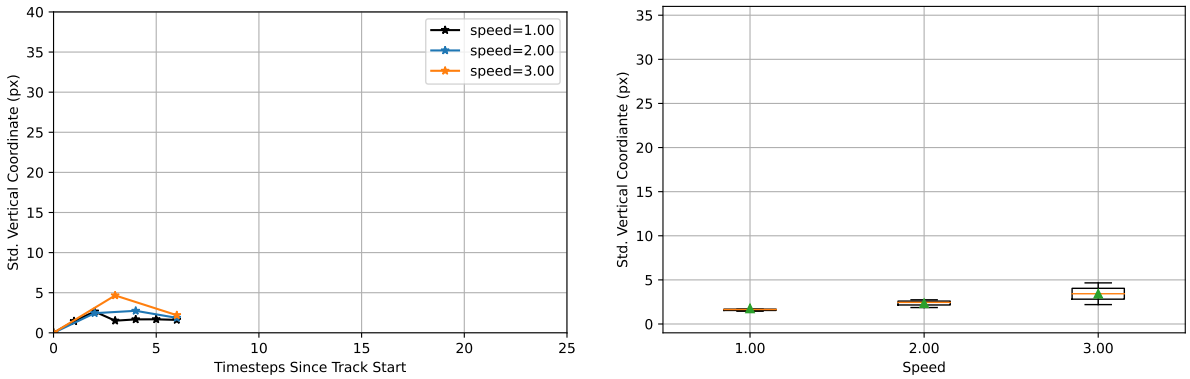


(b) $\eta(t)$, Vertical Coordinate

Figure B.64: **KITTI Dataset: Mean absolute errors are unaffected by speed when using the Correspondence Tracker.** The left column contains plots of the horizontal (top row) and vertical (bottom row) components of the mean absolute error $\eta(t)$ at each timestep t after initial feature detection at multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate values of each line for $t > 0$ in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. Compared to the results for the Lucas-Kanade Tracker in Figure B.61, mean errors do not change when speed is increased from 1.00 to 3.00.



(a) $\Phi(t)$, Horizontal Coordinates



(b) $\Phi(t)$, Vertical Coordinates

Figure B.65: **KITTI Dataset: Covariance is unaffected by speed when using the Correspondence Tracker.** The left column contains plots of the horizontal (top row) and vertical (bottom row) components of the mean absolute error $\eta(t)$ at each timestep t after initial feature detection at multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate values of each line for $t > 0$ in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. Compared to the results for the Lucas-Kanade Tracker in Figure B.62, covariances do not change when speed is increased from 1.00 to 2.00. Covariances show an increase of about 2 pixels when speed is increased from 2.00 to 3.00, however.

B.3 Supporting Figures for Gazebo Linear Dataset

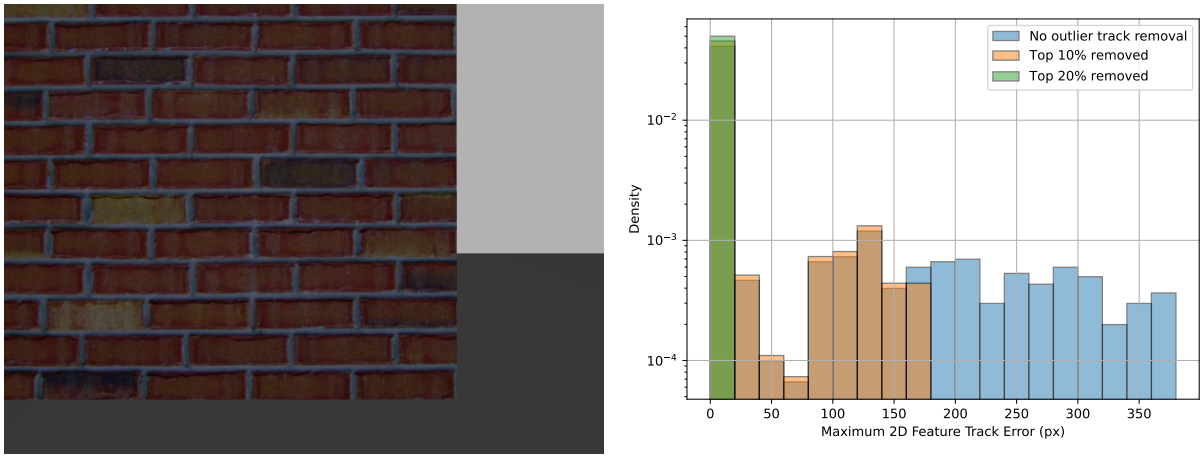


Figure B.66: **Gazebo Linear Dataset: We will throw out the 20% of tracks with the most error instead of the 10% of tracks.** The right figure plots the histogram density of the maximum L2 error of all feature tracks of one scene in log scale. The corresponding scene is pictured on the left. The large errors that still remain after removing the 10% of tracks with the most errors are caused by track propagation along smooth edges when the AGAST feature detector does not select perfect corners, as well as the asynchronous collection of RGB and depth images in the Gazebo simulator. The errors caused by track propagation along smooth edges are unlikely to occur in real world data, where backgrounds and textures are less ideal.

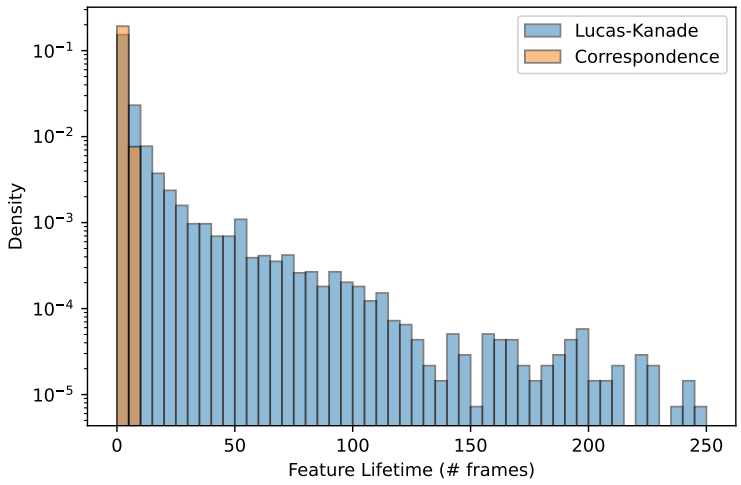
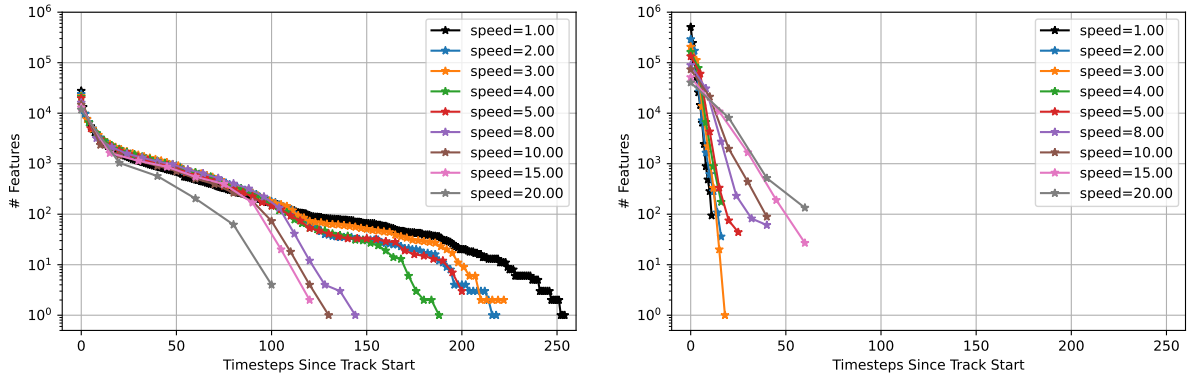


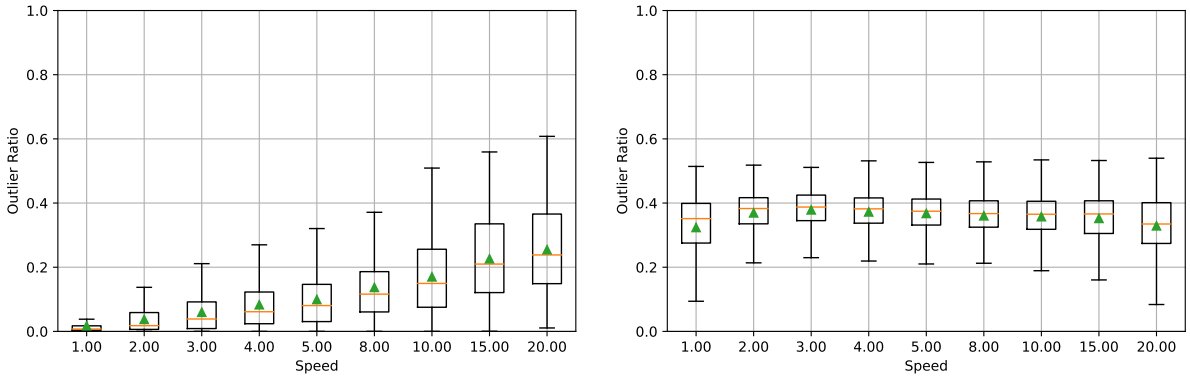
Figure B.67: **Gazebo Linear Dataset: Feature Lifetime is usually $\leq$ five frames.** The distribution of feature lifetimes is plotted as a log-scale histogram for both the Lucas-Kanade and Correspondence Tracker at nominal speed. Many features live for less $\leq$ five frames, especially when the Correspondence Tracker is used. However, Lucas-Kanade produces a long tail of features with longer lifetimes.



(a) Lucas-Kanade

(b) Correspondence

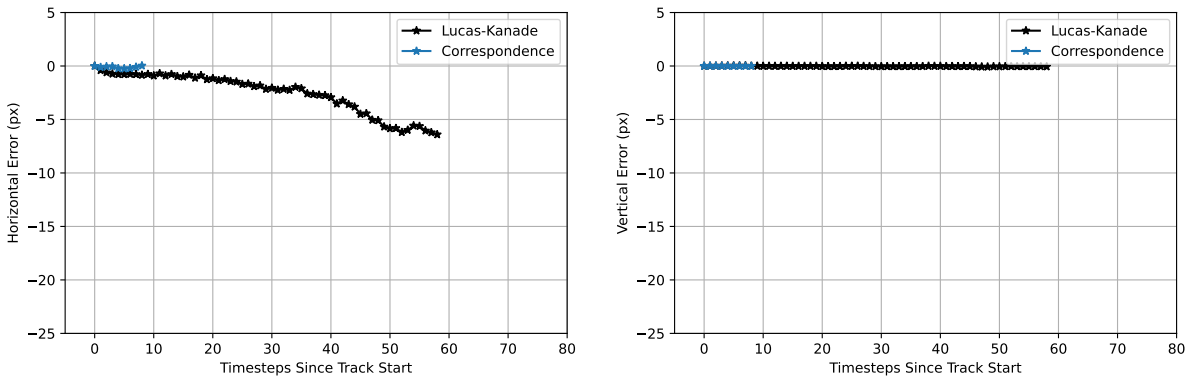
Figure B.68: Feature Lifetime is plotted on the horizontal axis. The vertical axis, in log scale, shows the number of features in all 11 scenes that lived at least that long for every tested speed. The number of features drops very fast, especially when the Correspondence Tracker is used. **In subsequent analyses, we only compute means errors and covariances at timesteps with at least 500 features on the Gazebo Linear Dataset.**



(a) Lucas-Kanade

(b) Correspondence

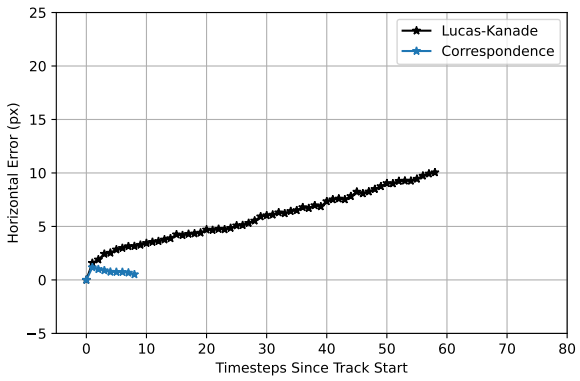
Figure B.69: **Gazebo Linear Dataset: Outlier Ratios are a function of speed when using the Lucas-Kanade Tracker and constant for the Correspondence Tracker.** Outlier ratios per frame are shown as box-and-whisker plots for tested speeds for the Lucas-Kanade tracker on the left and the Correspondence Tracker on the right. Mean values are shown as green triangles and median values are shown as orange lines. For lower speeds, the Lucas-Kanade tracker produces fewer outliers. Outlier ratios then increase with speed. On the other hand, the outlier ratio for the Correspondence Tracker remains constant, at around 40 percent.



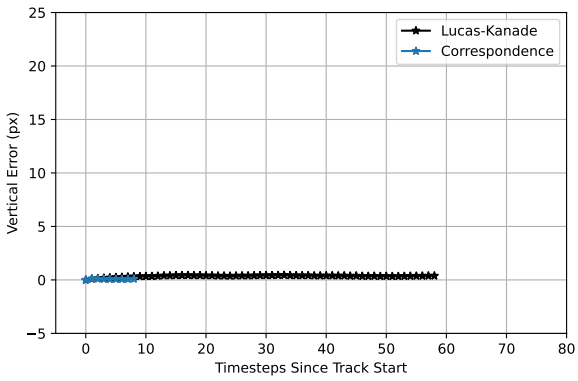
(a) $\nu(t)$, Horizontal Coordinate

(b) $\nu(t)$, Vertical Coordinate

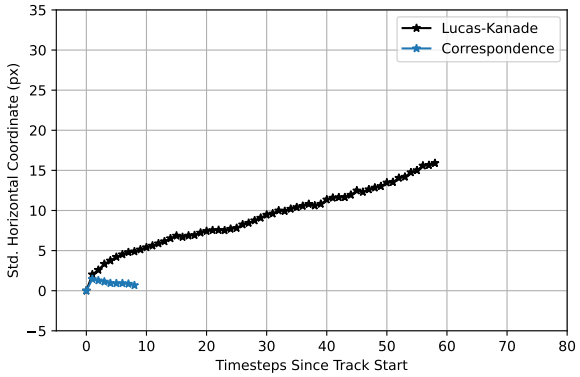
Figure B.70: **Gazebo Linear Dataset: The Lucas-Kanade Tracker slowly accumulates negative error in the horizontal direction at nominal speed. The Correspondence Tracker has zero mean error.** Lines shown are horizontal (left) and vertical (right) coordinates of mean error $\nu(t)$ calculated using tracks averaged over all scenes; calculation is cut off at 58 frames for the Lucas-Kanade Tracker and 9 frames for the Correspondence Tracker so that averages can be computed with at least 500 features.



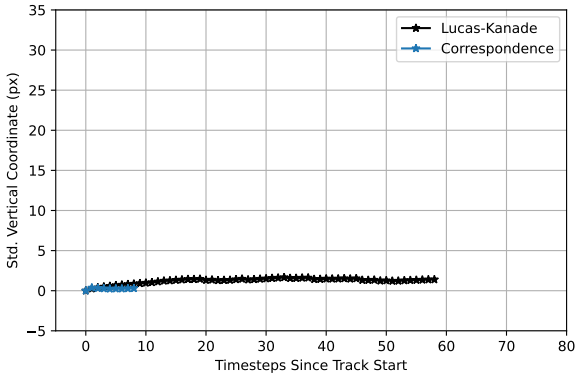
(a) $\eta(t)$, Horizontal Coordinate



(b) $\eta(t)$, Vertical Coordinate

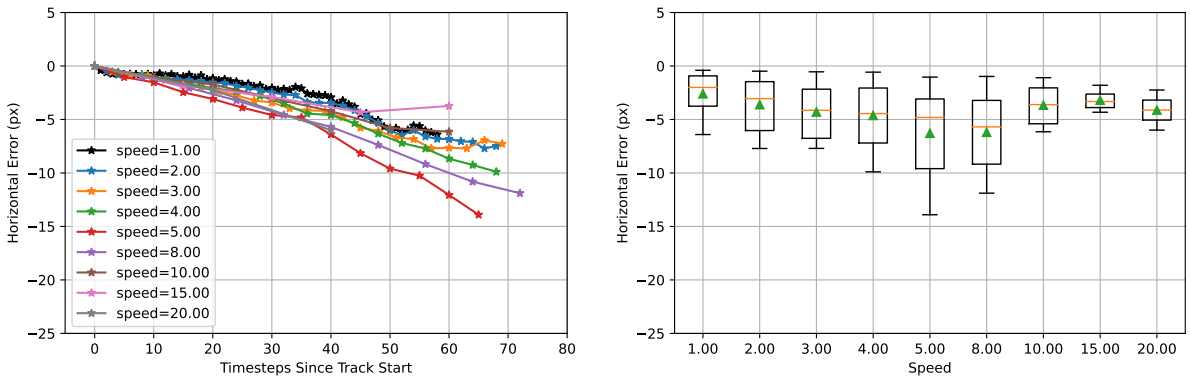


(c) $\Phi(t)$, Horizontal Coordinate

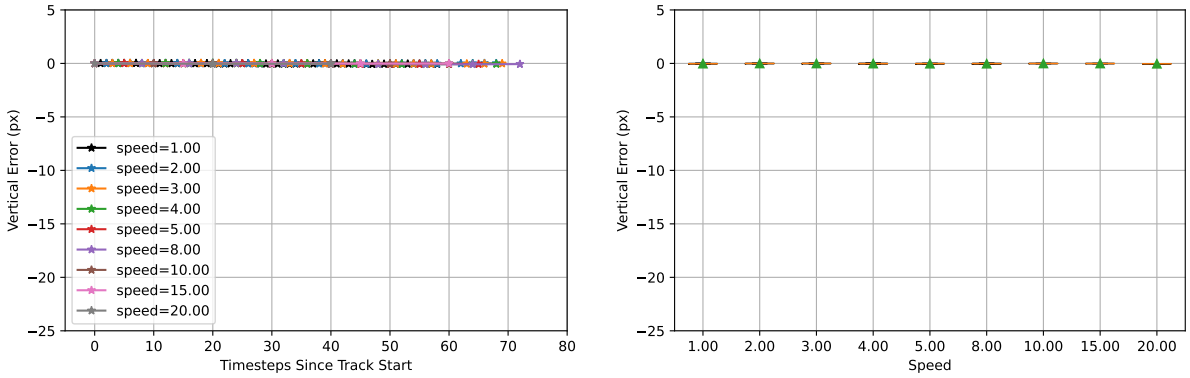


(d) $\Phi(t)$, Vertical Coordinate

Figure B.71: **Gazebo Linear Dataset: The Lucas-Kanade tracker drifts considerably more than the Correspondence Tracker, but only in the horizontal direction.** Lines shown are horizontal (left column) and vertical (right column) coordinates of mean absolute error $\eta(t)$ (top row) and covariance $\Phi(t)$ (bottom row) calculated using tracks averaged over all scenes; calculation is cut off at 58 frames for Lucas-Kanade Tracker and 9 frames for the Correspondence Tracker so that averages can be computed with at least 500 features. Both mean absolute error and covariance are constant when using the Correspondence Tracker. On the other hand, the horizontal coordinate of $\eta(t)$ and $\Phi(t)$ drifts upwards when using the Lucas-Kanade Tracker.

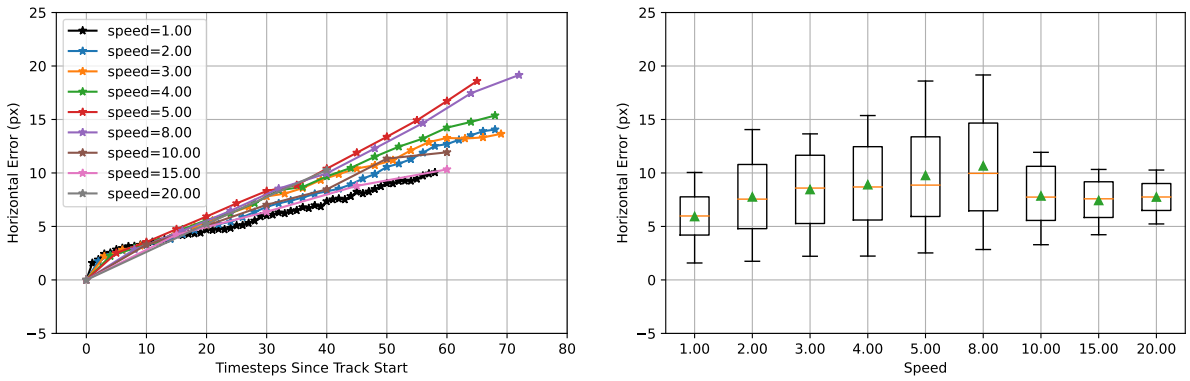


(a) $\nu(t)$, Horizontal Coordinate

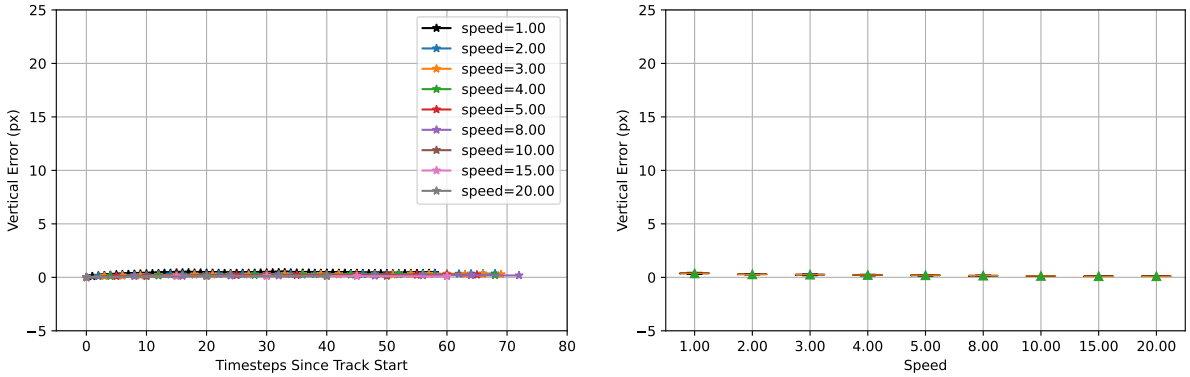


(b) $\nu(t)$, Vertical Coordinate

Figure B.72: **Gazebo Linear Dataset: Mean errors increase with speed when using the Lucas-Kanade Tracker.** The left column contains plots of the horizontal (top row) and vertical (bottom row) components of the mean tracking error $\nu(t)$ at each timestep t after initial feature detection at multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate values of each line for $t > 0$ in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. The top-right shows that mean errors in the horizontal coordinate become more negative as speed is increased from 1.00 to 8.00. The mean error then decreases for speeds=10.00 (brown line), 15.00 (pink line), and 20.00 (gray line), showing that both the number of elapsed frames, and the speed are both factors that affect $\nu(t)$. For all speeds, mean error is close to zero in the vertical coordinate.

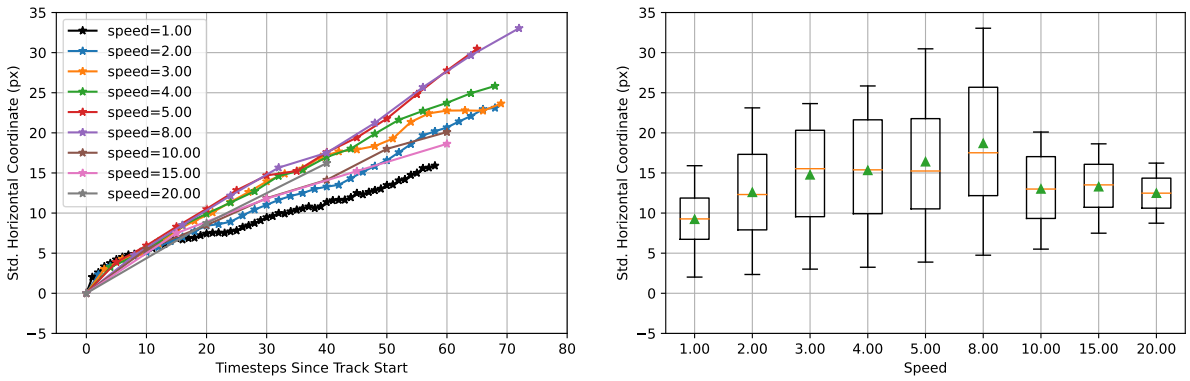


(a) $\eta(t)$, Horizontal Coordinate

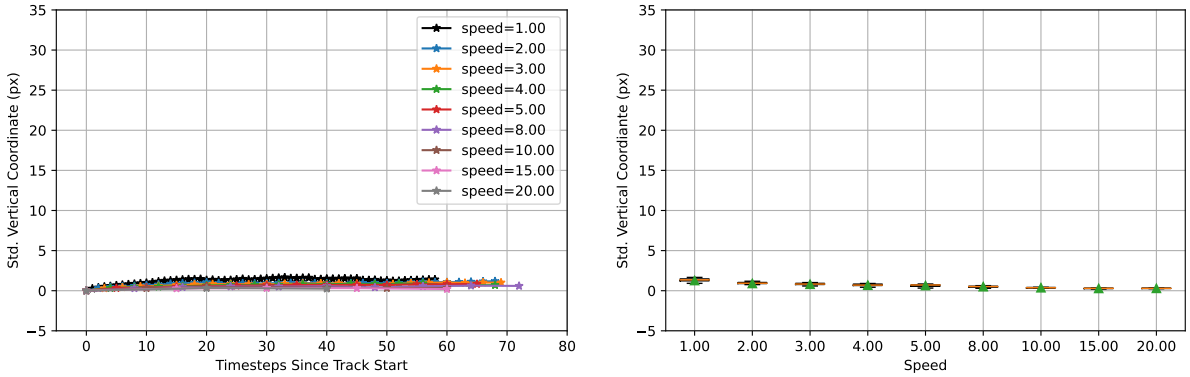


(b) $\eta(t)$, Vertical Coordinate

Figure B.73: **Gazebo Linear Dataset: Mean absolute errors increase with speed when using the Lucas-Kanade Tracker.** The left column contains plots of the horizontal (top row) and vertical (bottom row) components of the mean absolute error $\eta(t)$ at each timestep t after initial feature detection at multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate values of each line for $t > 0$ in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. Mean absolute errors in the horizontal coordinate increase as speed is increased from 1.00 to 8.00. $\eta(t)$ then decreases for speeds=10.00 (brown line), 15.00 (pink line), and 20.00 (gray line), showing that both the number of elapsed frames, and the speed are both factors that affect $\nu(t)$. For all speeds, mean absolute error is close to zero in the vertical coordinate.

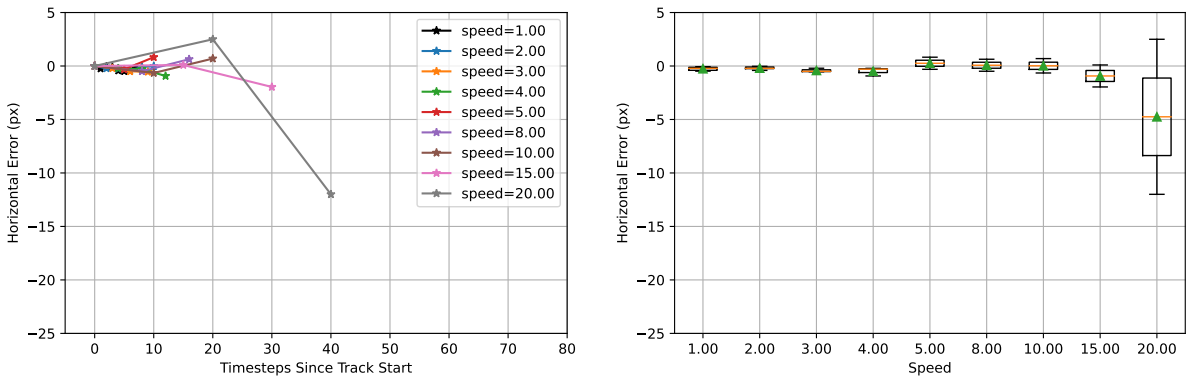


(a) $\Phi(t)$, Horizontal Coordinate

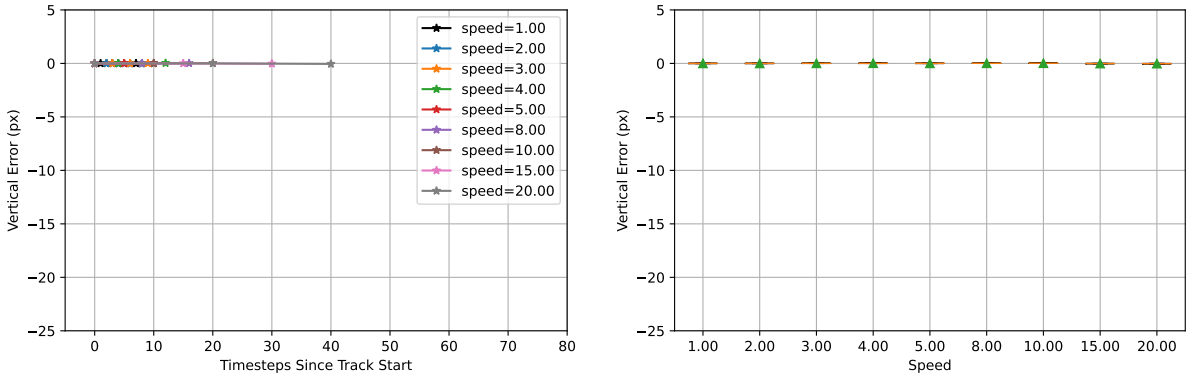


(b) $\Phi(t)$, Vertical Coordinate

Figure B.74: **Gazebo Linear Dataset: Covariance increases with speed when using the Lucas-Kanade Tracker.** The left column contains plots of the horizontal (top row) and vertical (bottom row) components of the covariance $\Phi(t)$ at each timestep t after initial feature detection at multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate values of each line for $t > 0$ in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. Covariance increases in the horizontal coordinate increase as speed is increased from 1.00 to 8.00. The covariance then decreases for speeds=10.00 (brown line), 15.00 (pink line), and 20.00 (gray line), showing that both the number of elapsed frames, and the speed are both factors that affect $\Phi(t)$. For all speeds, covariance is close to zero in the vertical coordinate.

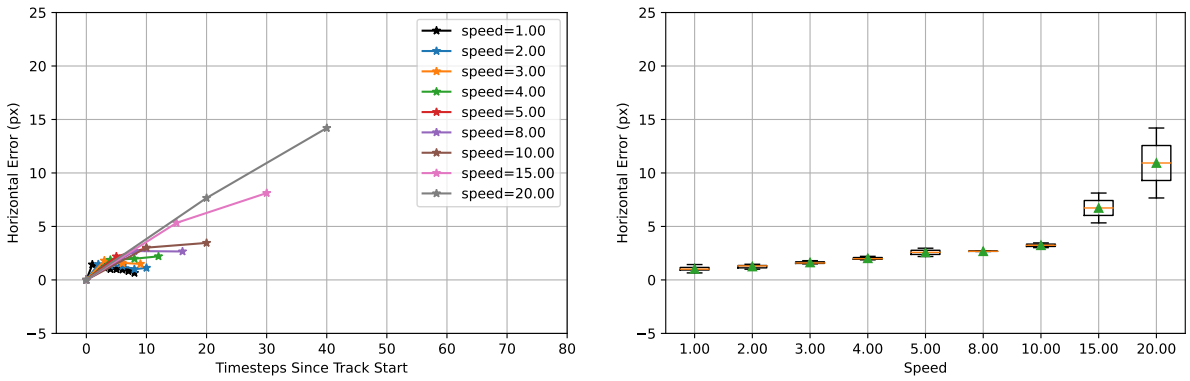


(a) $\nu(t)$, Horizontal Coordinate

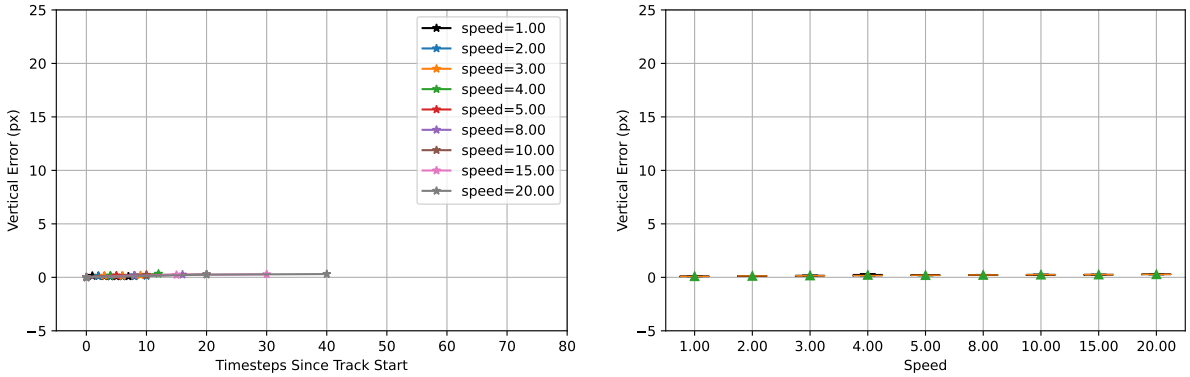


(b) $\nu(t)$, Vertical Coordinate

Figure B.75: **Gazebo Linear Dataset: Mean errors are unaffected by speed when using the Correspondence Tracker until tracking failure occurs.** The left column contains plots of the horizontal (top row) and vertical (bottom row) components of the mean tracking error $\nu(t)$ at each timestep t after initial feature detection at multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate values of each line for $t > 0$ in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. In the horizontal coordinate, mean errors remain near zero as speed is increased from 1.00 to 15.00. Mean errors are larger when speed=20.00. The mean error is close to zero in the vertical coordinate.

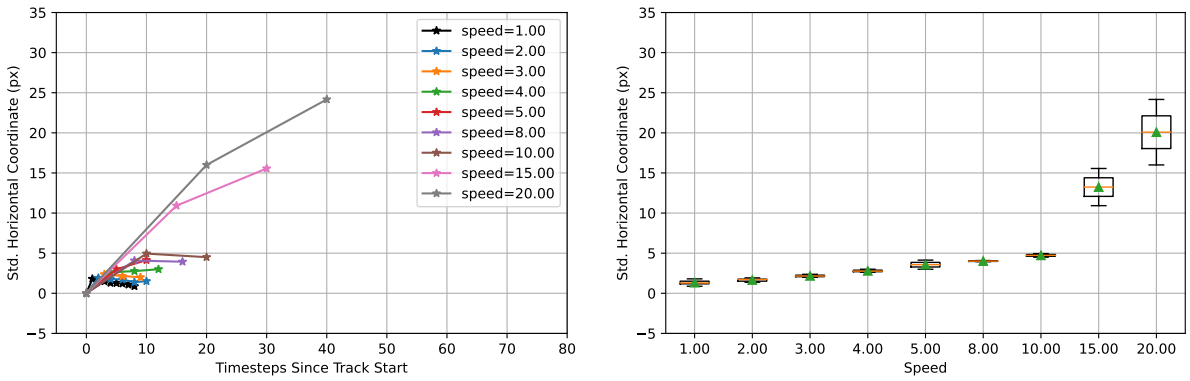


(a) $\eta(t)$, Horizontal Coordinate

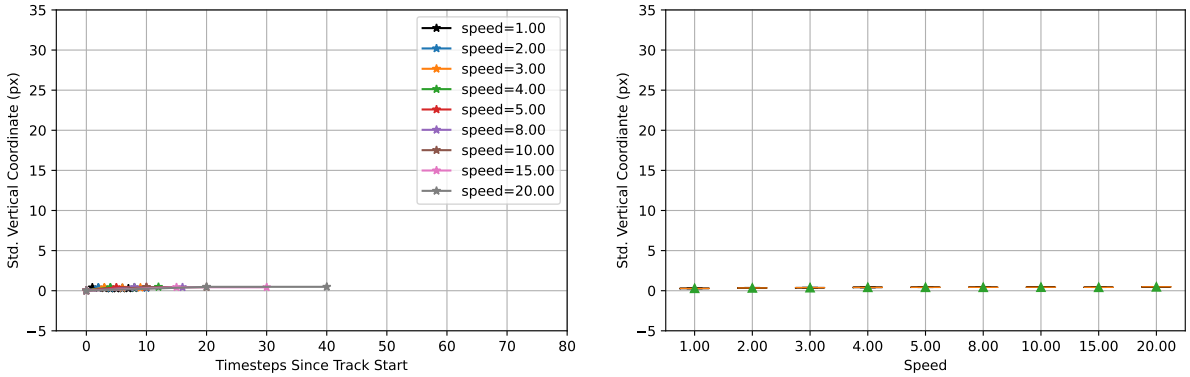


(b) $\eta(t)$, Vertical Coordinate

Figure B.76: **Gazebo Linear Dataset: Mean absolute errors increase with speed when using the Correspondence Tracker.** The left column contains plots of the horizontal (top row) and vertical (bottom row) components of the mean absolute error $\eta(t)$ at each timestep t after initial feature detection at multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate values of each line for $t > 0$ in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. In the horizontal coordinate, mean absolute errors increase slowly with speed at first; increases are larger from speed=10.00 to speed=15.00 and speed=15.00 to speed=20.00. The mean absolute error is approximately 0 in the vertical coordinate.

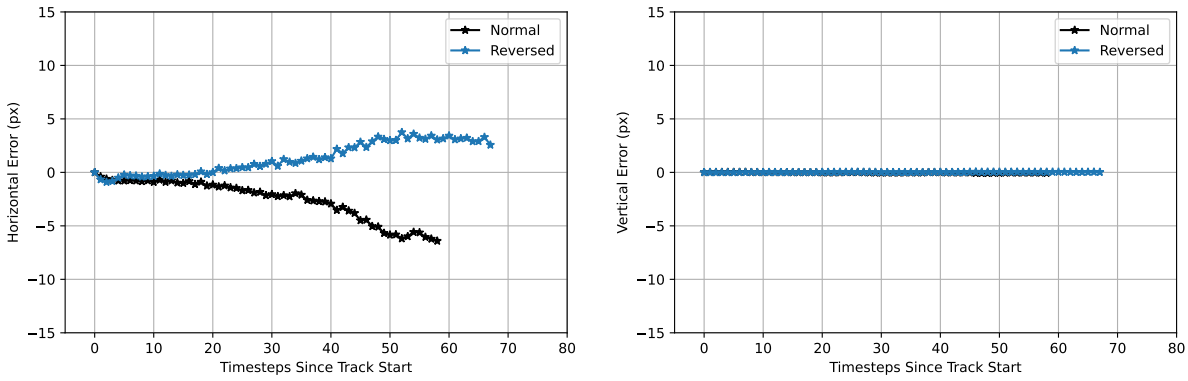


(a) $\Phi(t)$, Horizontal Coordinate



(b) $\Phi(t)$, Vertical Coordinate

Figure B.77: **Gazebo Linear Dataset: Covariance increases speed when using the Correspondence Tracker.** The left column contains plots of the horizontal (top row) and vertical (bottom row) components of the mean tracking error $\Phi(t)$ at each timestep t after initial feature detection at multiple speeds. Each dot corresponds to a processed frame; lines for higher speeds contain data from fewer frames and therefore show fewer dots. The right column plots the ordinate values of each line for $t > 0$ in the left figures as a box plot: means are shown as green triangles and medians are shown as orange lines. In the horizontal coordinate, covariance increases slowly with speed at first; increases are larger from speed=10.00 to speed=15.00 and speed=15.00 to speed=20.00. The covariance is close to zero in the vertical coordinate.



(a) $\nu(t)$, Horizontal Coordinate

(b) $\nu(t)$, Vertical Coordinate

Figure B.78: **Gazebo Linear Dataset: The Lucas-Kanade Tracker drifts opposite the direction of motion.** Lines above contain $\nu(t)$ computed from tracks using the Lucas-Kanade Tracker. In the black lines, the quadrotor is flying horizontally from left to right, as is the case in the rest of the experiments on the Gazebo Linear Dataset. In the blue lines, the quadrotor is flying horizontally from right to left while observing the same scene; the scene is not mirror-imaged, so the features tracked in the two trajectories are not identical. Once again, there is nearly no mean error in the vertical direction. However, mean horizontal error is positive instead of negative.

REFERENCES

[AAC15] A. Aghar-Mohammadi, S. Agarwal, S. Chakravorty, and N. Amato. "Simultaneous Localization and Planning for Physical Mobile Robots via Enabling Dynamic Replanning in Belief Space." *CoRR*, **abs/1510.07380**, 2015.

[ABA20] Javier Antoran, Umang Bhatt, Tameem Adel, Adrian Weller, and José Miguel Hernández-Lobato. "Getting a CLUE: A Method for Explaining Uncertainty Estimates." September 2020.

[ABJ20] Anastasios Nikolas Angelopoulos, Stephen Bates, Michael Jordan, and Jitendra Malik. "Uncertainty Sets for Image Classifiers using Conformal Prediction." September 2020.

[ACH18] M. S. Ahn, H. Chae, and D. W. Hong. "Stable, Autonomous, Unknown Terrain Locomotion for Quadrupeds Based on Visual Feedback and Mixed-Integer Convex Optimization." In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3791–3798, 2018.

[ADS12] Henrik Aanæs, Anders Lindbjerg Dahl, and Kim Steenstrup Pedersen. "Interesting Interest Points." *International Journal of Computer Vision*, **97**(1):18–35, March 2012.

[AGH19] J. Andersson, J. Gillis, G. Horn, J. Rawlings, and M. Diehl. "CasADi – A software framework for nonlinear optimization and optimal control." *Mathematical Programming Computation*, **11**(1):1–36, 2019.

[ALF11] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. "S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems." pp. 254–257. Springer Berlin Heidelberg, 2011.

[AT65] Karl-Johan Astrom and Bohlin Torsten. "Numerical Identification of Linear Dynamic Systems from Normal Operating Records." *IFAC Proceedings Volumes*, **2**(2):96–111, September 1965.

[AZA21] Adel Ahmadyan, Liangkai Zhang, Artsiom Ablavatski, Jianing Wei, and Matthias Grundmann. "Objectron: A Large Scale Dataset of Object-Centric Videos in the Wild with Pose Annotations." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[BB18] Axel Barrau and Silvère Bonnabel. "Invariant Kalman Filtering." *Annual Review of Control, Robotics, and Autonomous Systems*, **1**(1):237–257, 2018. _eprint: https://doi.org/10.1146/annurev-control-060117-105010.

[BBB19] Martin Brossard, Axel Barrau, and Silvère Bonnabel. "Exploiting Symmetries to Design EKFs With Consistency Properties for Navigation and SLAM." *IEEE Sensors Journal*, **19**(4):1572–1579, February 2019. Conference Name: IEEE Sensors Journal.

[BBO17] Michael Bloesch, Michael Burri, Sammy Omari, Marco Hutter, and Roland Siegwart. "Iterated extended Kalman filter based visual-inertial odometry using direct photometric feedback." *The International Journal of Robotics Research*, **36**(10):1053–1072, September 2017. Publisher: SAGE Publications Ltd STM.

[BCR21] Rina Foygel Barber, Emmanuel J. Candès, Aaditya Ramdas, and Ryan J. Tibshirani. "Predictive inference with the jackknife+." *The Annals of Statistics*, **49**(1):486–507, February 2021. Publisher: Institute of Mathematical Statistics.

[BHH13] M. Bloesch, M. Hutter, M. Hoepflinger, S. Leutenegger, C. Gehring, D. Remy, and R. Siegwart. "State estimation for legged robots-consistent fusion of leg kinematics and IMU." *Robotics*, **17**:17–24, 2013.

[BLV17] Vassileios Balntas, Karel Lenc, Andrea Vedaldi, and Krystian Mikolajczyk. "HPatches: A Benchmark and Evaluation of Handcrafted and Learned Local Descriptors." pp. 5173–5182, 2017.

[BNG06] Tim Bailey, Juan Nieto, Jose Guivant, Michael Stevens, and Eduardo Nebot. "Consistency of the EKF-SLAM algorithm." In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3562–3568. IEEE, 2006.

[CCC16] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J. Leonard. "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age." *IEEE Transactions on Robotics*, **32**(6):1309–1332, December 2016.

[CGD09] J. Civera, O. G. Grasa, A. J. Davison, and J. M. M. Montiel. "1-point RANSAC for EKF-based Structure from Motion." In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3498–3504, October 2009.

[CWW17] Ronald Clark, Sen Wang, Hongkai Wen, Andrew Markham, and Niki Trigoni. "VINet: Visual-Inertial Odometry as a Sequence-to-Sequence Learning Problem." *Proceedings of the AAAI Conference on Artificial Intelligence*, **31**(1), February 2017. Number: 1.

[DB06] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I." *IEEE Robotics & Automation Magazine*, **13**(2):99–110, 2006.

[DBB21] Nikita Durasov, Timur Bagautdinov, Pierre Baque, and Pascal Fua. "Masksembles for Uncertainty Estimation." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13539–13548, June 2021.

[DL20] Andrea De Maio and Simon Lacroix. "Simultaneously Learning Corrections and Error Models for Geometry-Based Visual Odometry Methods." *IEEE Robotics and Automation Letters*, **5**(4):6536–6543, October 2020.

[ECH18] M. Everett, Y. Chen, and J. How. "Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning." pp. 3052–3059, 10 2018.

[ESC14] Jakob Engel, Thomas Schöps, and Daniel Cremers. "LSD-SLAM: Large-Scale Direct Monocular SLAM." In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pp. 834–849, Cham, 2014. Springer International Publishing.

[FBH18] J. Fisac, A. Bajcsy, S. Herbert, D. Fridovich-Keil, S. Wang, C. Tomlin, and A. Dragan. "Probabilistically Safe Robot Planning with Confidence-Based Human Predictions." 06 2018.

[FCD17] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. "On-Manifold Preintegration for Real-Time Visual–Inertial Odometry." *IEEE Transactions on Robotics*, **33**(1):1–21, February 2017.

[FS18] Xiaohan Fei and Stefano Soatto. "Visual-Inertial Object Detection and Mapping." In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pp. 318–334, Cham, 2018. Springer International Publishing.

[GDS20] Fredrik K. Gustafsson, Martin Danelljan, and Thomas B. Schon. "Evaluating Scalable Bayesian Deep Learning Methods for Robust Computer Vision." pp. 318–319, 2020.

[GEL20] Patrick Geneva, Kevin Eckenhoff, Woosik Lee, Yulin Yang, and Guoquan Huang. "OpenVINS: A Research Platform for Visual-Inertial Estimation." In *Proc. of the IEEE International Conference on Robotics and Automation*, Paris, France, 2020.

[GG76] M. Grewal and K. Glover. "Identifiability of linear and nonlinear dynamical systems." *IEEE Transactions on Automatic Control*, **21**(6):833–837, December 1976. Conference Name: IEEE Transactions on Automatic Control.

[GG16] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning." In *International Conference on Machine Learning*, pp. 1050–1059, June 2016.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite." In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[GPS17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. "On Calibration of Modern Neural Networks." In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1321–1330. JMLR.org, 2017. event-place: Sydney, NSW, Australia.

[GR18] Jochen Gast and Stefan Roth. "Lightweight Probabilistic Deep Networks." pp. 3369–3378, 2018.

[GT12] Dorian Gálvez-López and J. D. Tardós. "Bags of Binary Words for Fast Place Recognition in Image Sequences." *IEEE Transactions on Robotics*, **28**(5):1188–1197, October 2012.

[GY15] Guillermo Gallego and Anthony Yezzi. "A Compact Formula for the Derivative of a 3-D Rotation in Exponential Coordinates." *Journal of Mathematical Imaging and Vision*, **51**(3):378–384, March 2015.

[HAY20] J. Hooks, M. Ahn, J. Yu, X. Zhang, T. Zhu, H. Chae, and D. Hong. "ALPHRED: A Multi-Modal Operations Quadruped Robot for Package Delivery Applications." *IEEE Robotics and Automation Letters (RA-L)*, 2020.

[HD18] Dan Hendrycks and Thomas Dietterich. "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations." September 2018.

[HDF12] Jared Heinly, Enrique Dunn, and Jan-Michael Frahm. "Comparative Evaluation of Binary Features." In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, Lecture Notes in Computer Science, pp. 759–773, Berlin, Heidelberg, 2012. Springer.

[HHC15] A. Hereid, C. Hubicki, E. Cousineau, J. Hurst, and A. Ames. "Hybrid zero dynamics based multiple shooting optimization with applications to robotic walking." In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5734–5740, May 2015.

[HKB14] Joel A Hesch, Dimitrios G Kottas, Sean L Bowman, and Stergios I Roumeliotis. "Camera-IMU-based localization: Observability analysis and consistency improvement." *The International Journal of Robotics Research*, **33**(1):182–201, January 2014. Publisher: SAGE Publications Ltd STM.

[HMR09] Guoquan P. Huang, Anastasios I. Mourikis, and Stergios I. Roumeliotis. "A First-Estimates Jacobian EKF for Improving SLAM Consistency." In Oussama Khatib, Vijay Kumar, and George J. Pappas, editors, *Experimental Robotics*, Springer Tracts in Advanced Robotics, pp. 373–382, Berlin, Heidelberg, 2009. Springer.

[Hol13] Gerard J. Holzmann. "Landing a Spacecraft on Mars." *IEEE Software*, **30**(2):83–86, March 2013.

[Hor87] Berthold K. P. Horn. "Closed-form solution of absolute orientation using unit quaternions." *JOSA A*, **4**(4):629–642, April 1987.

[HOZ21] Yibo Hu, Yuzhe Ou, Xujiang Zhao, Jin-Hee Cho, and Feng Chen. "Multidimensional Uncertainty-Aware Evidential Neural Networks." *Proceedings of the AAAI Conference on Artificial Intelligence*, **35**(9):7815–7822, May 2021.

[HP18] Sejong Heo and Chan Gook Park. "Consistent EKF-Based Visual-Inertial Odometry on Matrix Lie Group." *IEEE Sensors Journal*, **18**(9):3780–3788, May 2018. Conference Name: IEEE Sensors Journal.

[HS12] Daniel Cabrini Hauagge and Noah Snavely. "Image matching using local symmetry features." In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 206–213, June 2012. ISSN: 1063-6919.

[HTS15] J. Hernandez, K. Tsotsos, and S. Soatto. "Observability, identifiability and sensitivity of vision-aided inertial navigation." In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2319–2325, May 2015.

[HZR16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." pp. 770–778, 2016.

[JCS20] Taejong Joo, Uijung Chung, and Min-Gwan Seo. "Being Bayesian about Categorical Probability." In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4950–4961. PMLR, 13–18 Jul 2020.

[JS11] E. Jones and S. Soatto. "Visual-inertial navigation, mapping and localization: A scalable real-time causal approach." *The International Journal of Robotics Research*, **30**(4):407–430, April 2011.

[KG17] Alex Kendall and Yarin Gal. "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pp. 5574–5584. Curran Associates, Inc., 2017.

[KGC15] Alex Kendall, Matthew Grimes, and Roberto Cipolla. "PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization." pp. 2938–2946, 2015.

[KH] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator.".

[KHB13] Dimitrios G. Kottas, Joel A. Hesch, Sean L. Bowman, and Stergios I. Roumeliotis. "On the Consistency of Vision-Aided Inertial Navigation." In Jaydev P. Desai, Gregory Dudek, Oussama Khatib, and Vijay Kumar, editors, *Experimental Robotics: The 13th International Symposium on Experimental Robotics*, Springer Tracts in Advanced Robotics, pp. 303–317. Springer International Publishing, Heidelberg, 2013.

[KHI19] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. "The Marabou Framework for Verification and Analysis of Deep Neural Networks." In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pp. 443–452, Cham, 2019. Springer International Publishing.

[KIS19] V. Kalogeiton, K. Ioannidis, G. Ch. Sirakoulis, and E. Kosmatopoulos. "Real-Time Active SLAM and Obstacle Avoidance for an Autonomous Robot Based on Stereo Vision." *Cybernetics and Systems*, **50**(3):239–260, 2019.

[KK01] Y. Kanazawa and K. Kanatani. "Do we really have to consider covariance matrices for image features?" In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pp. 301–306 vol.2, July 2001.

[KLC95] P. Kaelbling, M. Littman, and A. Cassandra. "Planning and acting in partially observable stochastic domains." *Elsevier*, 1995.

[Lau10] Odile Laurent. "Using Formal Methods and Testability Concepts in the Avionics Systems Validation and Verification (V&V) Process." In *Verification and Validation 2010 Third International Conference on Software Testing*, pp. 1–10, April 2010. ISSN: 2159-4848.

[LC19a] Seong Hun Lee and Javier Civera. "Closed-Form Optimal Two-View Triangulation Based on Angular Errors." In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[LC19b] Seong Hun Lee and Javier Civera. "Closed-Form Optimal Two-View Triangulation Based on Angular Errors." pp. 2681–2689, 2019.

[LHD06] C. Leung, S. Huang, and G. Dissanayake. "Active SLAM using Model Predictive Control and Attractor based Exploration." pp. 5026 – 5031, 11 2006.

[LK81] Bruce D. Lucas and Takeo Kanade. "An iterative image registration technique with an application to stereo vision." In *Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2*, IJCAI'81, pp. 674–679, San Francisco, CA, USA, August 1981. Morgan Kaufmann Publishers Inc.

[LLC19] Nicolas Lanzetti, Ying Zhao Lian, Andrea Cortinovis, Luis Dominguez, Mehmet Mercangöz, and Colin Jones. "Recurrent neural network based MPC for process industries." In *2019 18th European Control Conference (ECC)*, pp. 1005–1010. IEEE, 2019.

[Lon81] H. C. Longuet-Higgins. "A computer algorithm for reconstructing a scene from two projections." *Nature*, **293**(5828):133–135, September 1981. Number: 5828 Publisher: Nature Publishing Group.

[LOV18] Katherine Liu, Kyel Ok, William Vega-Brown, and Nicholas Roy. "Deep Inference for Covariance Estimation: Learning Gaussian Noise Models for State Estimation." In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1436–1443, May 2018.

[Low99] D.G. Lowe. "Object recognition from local scale-invariant features." In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pp. 1150–1157 vol.2, September 1999.

[LPB17] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles." In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pp. 6402–6413. Curran Associates, Inc., 2017.

[LSS20] Antonio Loquercio, Mattia Segu, and Davide Scaramuzza. "A General Framework for Uncertainty Estimation in Deep Learning." *IEEE Robotics and Automation Letters*, **5**(2):3153–3160, April 2020.

[LW12] Eugene Lavretsky and Kevin Wise. *Robust and Adaptive Control: With Aerospace Applications*. Springer Science & Business Media, November 2012. Google-Books-ID: a2128lhlWfQC.

[MG18] Andrey Malinin and Mark Gales. "Predictive Uncertainty Estimation via Prior Networks." *Advances in Neural Information Processing Systems*, **31**:7047–7058, 2018.

[MHB10] Elmar Mair, Gregory D. Hager, Darius Burschka, Michael Suppa, and Gerhard Hirzinger. "Adaptive and generic corner detection based on the accelerated segment test." In *Proceedings of the 11th European conference on Computer vision: Part II*, ECCV'10, pp. 183–196, Berlin, Heidelberg, September 2010. Springer-Verlag.

[MHZ17] Josef Maier, Martin Humenberger, Oliver Zendel, and Markus Vincze. "Ground Truth Accuracy and Performance of the Matching Pipeline." In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 969–979, July 2017. ISSN: 2160-7516.

[ML94] Richard M. Murray and Zexiang Li. *A Mathematical Introduction to Robotic Manipulation*. Routledge, Boca Raton, 1 edition edition, March 1994.

[MMT15a] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System." *IEEE Transactions on Robotics*, **31**(5):1147–1163, Oct 2015.

[MMT15b] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. "ORB-SLAM: a Versatile and Accurate Monocular SLAM System." *IEEE Transactions on Robotics*, **31**(5):1147–1163, October 2015.

[MS05] K. Mikolajczyk and C. Schmid. "A performance evaluation of local descriptors." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **27**(10):1615–1630, October 2005.

[MSK12] Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf, and Oskar von Stryk. "Comprehensive Simulation of Quadrotor UAVs using ROS and Gazebo." In *3rd Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, p. to appear, 2012.

[MTS05] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. "A Comparison of Affine Region Detectors." *International Journal of Computer Vision*, **65**(1):43–72, November 2005.

[NDZ19] Jeremy Nixon, Michael W. Dusenberry, Linchuan Zhang, Ghassen Jerfel, and Dustin Tran. "Measuring Calibration in Deep Learning." pp. 38–41, 2019.

[Nea12] Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer Science & Business Media, December 2012.

[NH02] Kevin Nickels and Seth Hutchinson. "Estimating uncertainty in SSD-based feature tracking." *Image and Vision Computing*, **20**(1):47–58, January 2002.

[OFR19] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. "Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift." *Advances in Neural Information Processing Systems*, **32**:13991–14002, 2019.

[PN18] Mikael Persson and Klas Nordberg. "Lambda Twist: An Accurate Fast Robust Perspective Three Point (P3P) Solver." pp. 318–332, 2018.

[PSA17] Alberto Padoan, Giordano Scarciotti, and Alessandro Astolfi. "A Geometric Characterization of the Persistence of Excitation Condition for the Solutions of Autonomous Systems." *IEEE Transactions on Automatic Control*, **62**(11):5666–5677, November 2017. Conference Name: IEEE Transactions on Automatic Control.

[PXS11] Barnabás Póczos, Liang Xiong, and Jeff Schneider. "Nonparametric divergence estimation with applications to machine learning on distributions." In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI'11, pp. 599–608, Barcelona, Spain, July 2011. AUAI Press.

[Rai86] M. Raibert. *Legged robots that balance*. MIT press, 1986.

[RDS15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge." *International Journal of Computer Vision (IJCV)*, **115**(3):211–252, 2015.

[RF18] J. Redmon and A. Farhadi. "YOLOv3: An Incremental Improvement." *arXiv*, 2018.

[RRK11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. "ORB: An efficient alternative to SIFT or SURF." In *2011 International Conference on Computer Vision*, pp. 2564–2571, Nov 2011.

[RTS18] Carlos Riquelme, George Tucker, and Jasper Snoek. "Deep Bayesian Bandits Showdown: An Empirical Comparison of Bayesian Deep Networks for Thompson Sampling." February 2018.

[SDR19] Vaishaal Shankar, Achal Dave, Rebecca Roelofs, Deva Ramanan, Benjamin Recht, and Ludwig Schmidt. "Do Image Classifiers Generalize Across Time?" *arXiv:1906.02168 [cs, stat]*, December 2019. arXiv: 1906.02168.

[Comment: 23 pages, 11 tables, 11 figures. Paper Website: https://modestyachts.github.io/natural-perturbations-website/.]

[SEE12] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. "A benchmark for the evaluation of RGB-D SLAM systems." In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 573–580, October 2012. ISSN: 2153-0866, 2153-0858, 2153-0858.

[SGD18] David Schubert, Thore Goll, Nikolaus Demmel, Vladyslav Usenko, Jörg Stückler, and Daniel Cremers. "The TUM VI Benchmark for Evaluating Visual-Inertial Odometry." In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1680–1687, October 2018. ISSN: 2153-0866.

[SHS17a] Johannes L. Schönberger, Hans Hardmeier, Torsten Sattler, and Marc Pollefeys. "Comparative Evaluation of Hand-Crafted and Learned Local Features." In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6959–6968, 2017.

[SHS17b] Johannes L. Schönberger, Hans Hardmeier, Torsten Sattler, and Marc Pollefeys. "Comparative Evaluation of Hand-Crafted and Learned Local Features." In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6959–6968, July 2017. ISSN: 1063-6919.

[SKK18] Murat Sensoy, Lance Kaplan, and Melih Kandemir. "Evidential Deep Learning to Quantify Classification Uncertainty." In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[SKY15] Sameer Sheorey, Shalini Keshavamurthy, Huili Yu, Hieu Nguyen, and Clark N. Taylor. "Uncertainty Estimation for KLT Tracking." In C.V. Jawahar and Shiguang Shan, editors, *Computer Vision - ACCV 2014 Workshops*, Lecture Notes in Computer Science, pp. 475–487, Cham, 2015. Springer International Publishing.

[SLG76] T. Soderstrom, L. Ljung, and I. Gustavsson. "Identifiability conditions for linear multivariable systems operating under feedback." *IEEE Transactions on Automatic Control*, **21**(6):837–840, December 1976. Conference Name: IEEE Transactions on Automatic Control.

[SMT18] Torsten Sattler, Will Maddern, Carl Toft, Akihiko Torii, Lars Hammarstrand, Erik Stenborg, Daniel Safari, Masatoshi Okutomi, Marc Pollefeys, Josef Sivic, Fredrik Kahl, and Tomas Pajdla. "Benchmarking 6DOF Outdoor Visual Localization in Changing Conditions." In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8601–8610, June 2018. ISSN: 2575-7075.

[SNS18] Yasser Shoukry, Pierluigi Nuzzo, Alberto L. Sangiovanni-Vincentelli, Sanjit A. Seshia, George J. Pappas, and Paulo Tabuada. "SMC: Satisfiability Modulo Convex Programming." *Proceedings of the IEEE*, **106**(9):1655–1679, September 2018. Conference Name: Proceedings of the IEEE.

[STS21] Alexander Schperberg, Stephanie Tsuei, Stefano Soatto, and Dennis Hong. "SABER: Data-Driven Motion Planner for Autonomously Navigating Heterogeneous Robots." *IEEE Robotics and Automation Letters*, **6**(4):8086–8093, October 2021. Conference Name: IEEE Robotics and Automation Letters.

[Sut74] I.E. Sutherland. "Three-dimensional data input by tablet." *Proceedings of the IEEE*, **62**(4):453–461, April 1974. Conference Name: Proceedings of the IEEE.

[SZY21] Huihui Sun, Weijie Zhang, Runxiang Yu, and Yujie Zhang. "Motion Planning for Mobile Robots—Focusing on Deep Reinforcement Learning: A Systematic Review." *IEEE Access*, **9**:69061–69081, 2021. Conference Name: IEEE Access.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, August 2005. Google-Books-ID: wjM3AgAAQBAJ.

[TGE21] Christian Tomani, Sebastian Gruber, Muhammed Ebrar Erdem, Daniel Cremers, and Florian Buettner. "Post-Hoc Uncertainty Calibration for Domain Drift Scenarios." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10124–10132, June 2021.

[TM22] Patrizio Tomei and Riccardo Marino. "An Enhanced Feedback Adaptive Observer for Nonlinear Systems with Lack of Persistency of Excitation." *IEEE Transactions on Automatic Control*, pp. 1–6, 2022. Conference Name: IEEE Transactions on Automatic Control.

[TPM14] David Tedaldi, Alberto Pretto, and Emanuele Menegatti. "A robust and easy to implement method for IMU calibration without external equipments." In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3042–3049, May 2014. ISSN: 1050-4729.

[TYM20] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. "NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems." In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pp. 3–17, Cham, 2020. Springer International Publishing.

[USS17] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. "Sparsity Invariant CNNs." In *2017 International Conference on 3D Vision (3DV)*, pp. 11–20, October 2017. ISSN: 2475-7888.

[Van14] W. Van der Hof. "Robot search in unknown environments using POMDPs." *TU Delft University Thesis*, 2014.

[VBB13] William Vega-Brown, Abraham Bachrach, Adam Bry, Jonathan Kelly, and Nicholas Roy. "CELLO: A fast algorithm for Covariance Estimation." In *2013 IEEE International Conference on Robotics and Automation*, pp. 3160–3167, May 2013.

[Vec] VectorNav. "Calibration and Characterization of IMUs.".

[VT20] Cristiano Maria Verrelli and Patrizio Tomei. "Nonanticipating Lyapunov Functions for Persistently Excited Nonlinear Systems." *IEEE Transactions on Automatic Control*, **65**(6):2634–2639, June 2020. Conference Name: IEEE Transactions on Automatic Control.

[WB06] Andreas Wächter and Lorenz T. Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming." *Mathematical Programming*, **106**(1):25–57, March 2006.

[WLA19] Guotai Wang, Wenqi Li, Michael Aertsen, Jan Deprest, Sébastien Ourselin, and Tom Vercauteren. "Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks." *Neurocomputing*, **338**:34–45, April 2019.

[WM17] Xue Iuan Wong and Manoranjan Majji. "Uncertainty Quantification of Lucas Kanade Feature Track and Application to Visual Odometry." In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 950–958, July 2017. ISSN: 2160-7516.

[WOB17] Song Wu, Ard Oerlemans, Erwin M. Bakker, and Michael S. Lew. "A comprehensive evaluation of local detectors and descriptors." *Signal Processing: Image Communication*, **59**:150–167, 2017.

[WRM05] Jan C. Willems, Paolo Rapisarda, Ivan Markovsky, and Bart L.M. De Moor. "A note on persistency of excitation." *Systems & Control Letters*, **54**(4):325–329, April 2005.

[WVB18] Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. "Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches." February 2018.

[WZW20] Wenshan Wang, Delong Zhu, Xiangwei Wang, Yaoyu Hu, Yuheng Qiu, Chen Wang, Yafei Hu, Ashish Kapoor, and Sebastian Scherer. "TartanAir: A Dataset to Push the Limits of Visual SLAM." 2020.

[XOT19] S. Xin, R. Orsolino, and N. Tsagarakis. "Online Relative Footstep Optimization for Legged Robots Dynamic Walking Using Discrete-Time Model Predictive Control." 03 2019.

[YGE19] Yulin Yang, Patrick Geneva, Kevin Eckenhoff, and Guoquan Huang. "Degenerate Motion Analysis for Aided INS With Online Spatial and Temporal Sensor Calibration." *IEEE Robotics and Automation Letters*, **4**(2):2070–2077, April 2019. Conference Name: IEEE Robotics and Automation Letters.

[YWL19] J. Yuan, H. Wang, C. Lin, D. Liu, and D. Yu. "A Novel GRU-RNN Network Model for Dynamic Path Planning of Mobile Robot." *IEEE Access*, **7**:15140–15151, 2019.

[ZCY20] Chen Zhao, Zhiguo Cao, Jiaqi Yang, Ke Xian, and Xin Li. "Image Feature Correspondence Selection: A Comparative Study and a New Contribution." *IEEE Transactions on Image Processing*, **29**:3506–3519, 2020. Conference Name: IEEE Transactions on Image Processing.

[ZGS09] Bernhard Zeisl, Pierre Fite Georgel, Florian Schweiger, Eckehard Steinbach, and Nassir Navab. "Estimation of Location Uncertainty for Scale Invariant Feature Points." In *Proceedings of the British Machine Vision Conference*, pp. 57.1–57.12. BMVA Press, 2009. doi:10.5244/C.23.57.

[ZHF22] Lintong Zhang, Michael Helmberger, Lanke Frank Tarimo Fu, David Wisth, Marco Camurri, Davide Scaramuzza, and Maurice Fallon. "Hilti-Oxford Dataset: A Millimetre-Accurate Benchmark for Simultaneous Localization and Mapping.", 2022.

[ZHL21] Albert Zhao, Tong He, Yitao Liang, Haibin Huang, Guy Van den Broeck, and Stefano Soatto. "SAM: Squeeze-and-Mimic Networks for Conditional Visual Driving Policy Learning." In Jens Kober, Fabio Ramos, and Claire Tomlin, editors, *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pp. 156–175. PMLR, 16–18 Nov 2021.

[ZL21] Aurick Zhou and Sergey Levine. "Amortized Conditional Normalized Maximum Likelihood: Reliable Out of Distribution Uncertainty Estimation." In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12803–12812. PMLR, 18–24 Jul 2021.

[ZWS17] Teng Zhang, Kanzhi Wu, Jingwei Song, Shoudong Huang, and Gamini Dissanayake. "Convergence and Consistency Analysis for a 3-D Invariant-EKF SLAM." *IEEE Robotics and Automation Letters*, **2**(2):733–740, April 2017. Conference Name: IEEE Robotics and Automation Letters.

[Ås65] K. J Åström. "Optimal control of Markov processes with incomplete state information." *Journal of Mathematical Analysis and Applications*, **10**(1):174–205, February 1965.