REAL-TIME OPTIMIZATION FOR ROBUST STATE ESTIMATION AND CONTROL OF LEGGED ROBOTS

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Mechanical Engineering

Shuo Yang

B.Eng., Computer Engineering,

Hong Kong University of Science and Technology

M.Phil., Electrical & Computer Engineering,

Hong Kong University of Science and Technology

Carnegie Mellon University

Pittsburgh, PA

August 2024

Acknowledgements

During my years at CMU, aside from conducting research, I also navigated the upheaval of a global pandemic, celebrated the joyous birth of my son, and mourned the loss of several cherished family members. This challenging PhD journey tested my resilience in ways I never imagined, but it also transformed me. I emerged not only as a more skilled and knowledgeable roboticist but also as a more compassionate and resilient person. I would like to extend my heartfelt gratitude to everyone who supported and contributed to this thesis.

First and foremost, I express my deepest appreciation to my advisors, Prof. Zachary Manchester and Prof. Howie Choset, for their unwavering guidance, mentorship, and support throughout my PhD journey. Your insightful advice and encouragement have been invaluable. I am also immensely grateful to my thesis committee members, Prof. Aaron Johnson, Prof. Michael Posa, and Prof. Guanya Shi, for their critical feedback and constructive suggestions that significantly enhanced the quality of my research.

I extend my sincere thanks to the members of the CMU REx Lab, including Aaron Bishop, Ben Bokser, Chiyen Lee, John Zhang, Swaminathan Gurumurthy, Zixin Zhang, and Ibrahima Sory Sow, for their collaborative spirit and camaraderie. Your diverse perspectives and expertise greatly enriched my research experience.

I am also grateful to the CMU Biorobotics Lab members, especially Prof. Matt Travers, Dr. Lu Li, Ben Brown, Roshan Pradhan, Yizhu Gu, Chao Cao, and Yu Chen, for their valuable insights and assistance. Working alongside such talented individuals has been an inspiring experience.

A special mention goes to my collaborators, Prof. Mac Schwager, Dr. Simon Le Cleac'h, and Dr. Taylor Howell from Stanford University, Prof. Frank Dellaert, Gerry Chen, and Yetong Zhang from GeorgiaTech, Prof. Deva Ramanan and Gengshan Yang from CMU, and Zhengyu Fu from ETH Zurich. Your expertise and collaborative efforts have been instrumental in advancing my research.

I would like to acknowledge my industry mentors, Dr. Buck Babich from Nvidia and Anand Swaminathan from Tesla, for providing me with practical insights and guidance that bridged the gap between academic research and industry applications. Additionally, I would like to thank Xingxing Wang, CEO of Unitree Robotics, for providing robots to facilitate my research.

Finally, I would like to express my deepest gratitude to my wife, Lexie Ma, my son, Mason Yang, and my other family members for their love, unwavering support, patience, and encouragement throughout this journey. Without Lexie's support, I could not have made it through the devastating COVID lockdown period. Lexie took on countless household routines and sacrificed her time to allow me to focus on my research and studies. Her selflessness and dedication have been the bedrock of my success. Thank you, Lexie, for believing in me and for standing by my side through every challenge. My son Mason came into this world a few hours before an RAL editor got back to me requesting paper revisions. With the luck and courage Mason brought to me, I not only finished the paper revision in a few weeks and got it published, but also went on to publish many more papers. His birth made me a braver person, inspiring me to fight harder for my goals and persevere through challenges. Mason's arrival infused my life with new energy and determination, and I am grateful for the strength he has given me.

Abstract

This thesis aims to provide methods and algorithms that enhance legged robot locomotion capabilities in various aspects. In recent years, more legged robot solutions have emerged and begun to assume real-world applications like construction site inspection and law enforcement. As legged robots enter unstructured real-world scenarios, they need improved motion control stability and more precise state estimation. Through methodical investigations and experiments, this research contributes several hardware and software innovations that demonstrate significantly improved stability, mobility, and autonomy for legged robots.

On the control side, the thesis presents a legged robot hardware design incorporating two reaction wheels, which can be controlled alongside other joint motors in a model predictive controller. The additional reaction wheels greatly enhance the robot's stability. To harness the power of reaction wheels in real-time control, we explore the efficiency and flexibility of model predictive control. Due to the generic nature of its underlying numerical optimization framework, model predictive control can support different robot hardware designs within the same control framework.

On the state estimation side, this thesis develops several real-time odometry solutions combining multiple inertial measurement units, joint encoders, contact sensors, and cameras to achieve low-drift position estimation during long-term locomotion. We first study two ways to model error sources in leg odometry to improve estimation performance. Then, we develop two visual-inertial leg odometry solutions that achieve state-of-the-art estimation accuracy. Along the way, we also systematically study Kalman filtering and factor graph-based optimization, which are crucial tools for general robot state estimation.

Additionally, a chapter of this thesis is dedicated to the connection between optimization-based state estimation and optimization-based control through the factor graph. The factor graph, commonly used in large-scale state estimation and mapping, is a nuanced representation that explores sparsity in estimation problems. This similar sparsity, due to the inherent Markov property of robots, also appears in optimization-based trajectory generation and control. Thus, the graphical representation can illuminate some difficult control problems that are challenging to solve using conventional recursive methods.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Legged robots represent a critical frontier in robotics due to their potential to navigate complex and uneven terrains where wheeled robots falter. Their ability to mimic the mobility of biological organisms allows them to operate in environments such as disaster sites, rugged outdoor landscapes, and other scenarios that are challenging for traditional robots. This thesis presents a comprehensive exploration of novel algorithms and hardware designs aimed at enhancing the locomotion and state estimation capabilities of legged robots. Although robotic systems with any number of limbs can be called "legged" robots, in modern times, the term primarily refers to *quadrupeds* (four-legged robots) and *hexapods* (six-legged robots). Robots with more than six legs are rare, while two-legged robots have their own special terms: *biped* (if the robot only has a pelvis and a pair of legs) and *humanoid* (robots that resemble the human shape). In this thesis, we mainly use quadrupeds to discuss system and algorithm designs, but our work can be generalized to robots with any number of legs.

Our research pivots around the synergy between advanced control theories and state estimation techniques, underlined by the common thread of numerical optimization algorithms. By leveraging these advanced techniques, we have made several contributions that enable legged robots to achieve better balance, stability, and more precise state estimation, especially long-term position estimation.

In this chapter, we give an overview of the motivation, contributions, and organization of this thesis. The subsequent chapters will delve into the detailed development of these novel algorithms and hardware designs, demonstrating their effectiveness through rigorous analysis and real-world experiments.

1.1 Motivation

For decades, legged robots have been believed to surpass wheeled robots in terms of mobility and adaptability. In recent years, rapid developments in hardware components and embedded computing have drawn increasing attention to legged robots from both academia and industry. Numerous advanced legged robot platforms are now available, such as Boston Dynamics' Spot [39], ANYmal [59], and Unitree A1 [148]. These robots are capable of performing various tasks in challenging environments, such as climbing stairs, opening doors, and traversing rough terrains. The basic designs of legged robots are becoming mature, with robots converging on a few optimal configurations. As these robots transition from the lab to real-world applications, it is crucial to continue improving their performance. This includes not only enhancing existing capabilities but also enabling new functionalities whenever possible. There are two important aspects to this effort.

On the control side, although most legged robots have converged to a 12-motor configuration with point feet, there is still room for hardware innovation to improve stability. The literature already features some excellent hardware innovations that achieve this goal by adding spines [129], wheels [11], or tails [111] to legged systems. We aim to provide an alternative perspective in this direction, exploring novel hardware designs that can further enhance the stability and performance of legged robots.

At a high level, our contributions follow a specific design philosophy: we mimic human and animal shapes but do not constrain ourselves to them. Instead, we focus on identifying fundamental limitations and error sources and use the most compact solutions to address these issues, even if the solutions do not resemble animal shapes.

Another important topic is state estimation. As some legged robots have already entered real-world applications, various sensors have been employed for state estimation and mapping. Legged robots require precise estimation of physical states, such as body position and orientation (pose), as well as body velocity, to perform balancing control [14] and path planning [109] on challenging terrains. In many scenarios, robots must travel hundreds of meters autonomously [147], performing state estimation solely using onboard sensors, as external sensors like GPS and motion-capture systems are often unavailable. We are interested in identifying the most compact sensor solution that can achieve long-term position estimation. For control purposes, it is essential to obtain accurate orientation and velocity estimations. However, if we consider legged robots as measurement instruments, achieving precise position estimation becomes equally important.

An essential building block for state estimation in these cases is odometry. An odometer measures the robot's body velocity or incremental body position displacement, the integration of which results in global position estimation. Due to inevitable measurement errors, position estimation will drift unless an external sensor provides absolute global position measurements. Since global information may not always be accessible, reducing odometry noise is crucial for overall state estimation performance.

Although various odometry solutions have been developed for other mobile robots, legged robot odometry presents unique challenges and advantages that have yet to be systematically studied in the literature. Compared to wheels on conventional mobile robots, legs are less likely to slip even on uneven terrain, but using legs to infer robot body motion requires more careful modeling efforts. Existing platform-agnostic solutions for GPS-denied environments, such as lidar odometry [167] and Visual-Inertial Odometry (VIO)[73], are often too expensive and heavy for lightweight legged robots. VIO works well for drones and autonomous vehicles where motion is smooth and continuous[118]. But legged robots must deal with uneven terrain, high-frequency impacts, and contact dynamics, naively migrating VIO to legged robots does not work well [161, 165].

Some papers have studied the use of additional sensors on legged robots to improve odometry performance [56, 76, 92, 158], but there has been no systematic discussion of the factors affecting odometry accuracy. Part of the reason is that not a lot of existing applications require long-term odometry, where legged robots are mostly operated by humans. Even when autonomous, these robots are designed to move slowly and carefully, making bulky sensors and additional general Simultaneous Localization and Mapping (SLAM)[145] mechanisms sufficient[147]. However, these solutions are unsuitable for legged robots with limited payload capacity and onboard resources that need to move quickly in time-critical missions. A low-drift odometry solution leveraging only lightweight onboard sensors that perform well during fast and agile locomotion is needed. One of the core contributions of this thesis is a systematic study of odometry solutions with different hardware and software formulations and how control-related factors affect state estimation performance.

Additionally, because numerical optimization is the foundation for both control and state estimation, mathematical tools developed for one can be used in the other. Thus, exploring these connections in legged systems can illuminate some fundamental aspects of control and state estimation research.

1.2 Thesis Statement

This thesis explores the enhancement of legged robot locomotion and state estimation through innovative algorithms and hardware designs. Adhering to a flexible design philosophy, we draw inspiration from human and animal shapes without being constrained by them, focusing on using compact and effective solutions to address fundamental limitations and error sources in the system. This approach enhances robots' existing capabilities and enables new functionalities.

1.3 Thesis Contributions & Overview

The thesis statements are supported by not only new algorithms but also new hardware designs. Each chapter of this thesis contains contributions to either control or state estimation research. These contributions are not merely isolated instances of academic exploration; rather, they collectively forge a cohesive narrative that reinforces and validates the thesis's central statements.

Chapter 2 This chapter focuses on control. We propose a novel quadruped hardware system designed to enhance the balancing capabilities of legged robots. Using the Model Predictive Control (MPC) framework, we control this hardware system and demonstrate its significantly improved stability in real-world experiments. The chapter also provides a detailed explanation of MPC, with a particular emphasis on sparsity in MPC, which is crucial for efficient MPC implementation and closely related to factor graph-based state estimation. The contents of this chapter have been published in:

Lee, Chi-Yen*, **Shuo Yang***, Benjamin Bokser, and Zachary Manchester. "Enhanced Balance for Legged Robots Using Reaction Wheels." In 2023 IEEE International Conference on Robotics and Automation (ICRA), pp. 9980-9987. IEEE, 2023.

Chapter 3 This chapter focuses on state estimation. Building on conventional legged robot proprioceptive odometry (PO), we identify robot kinematic parameters as a major error source and propose a novel online calibration method to estimate these parameters. This method is validated through simulation and hardware experiments. The chapter also includes an observability analysis of the calibration problem, providing a general tool for analyzing the observability of nonlinear systems. The contents of this chapter have been published in:

Yang, Shuo, Howie Choset, and Zachary Manchester. "Online kinematic calibration for legged robots." IEEE Robotics and Automation Letters 7, no. 3 (2022): 8178-8185.

Chapter 4 This chapter focuses on state estimation. It presents Cerberus, a visual-inertial-leg odometry solution that achieves one of the best long-term position estimation performances on legged robots. Cerberus employs a sliding window factor graph formulation to perform real-time sensor fusion of visual, inertial, and leg odometry measurements. Additionally, online kinematics calibration is used to improve estimation accuracy. The contents of this chapter have been published in:

> **Yang, Shuo**, Zixin Zhang, Zhengyu Fu, and Zachary Manchester. "Cerberus: Low-drift visual-inertial-leg odometry for agile locomotion." In 2023 IEEE International Conference on Robotics and Automation (ICRA), pp. 4193-4199. IEEE, 2023.

Chapter 5 This chapter focuses on state estimation. We propose a novel hardware sensing solution for legged robots that leverages IMUs on the robot's feet to augment standard proprioceptive odometry (PO). The Multi-IMU Proprioceptive Odometry (MIPO) system is developed to harness sensor data from multiple IMUs. By using foot IMUs, MIPO eliminates a fundamental limitation of standard PO, achieving better velocity estimation accuracy. This improvement is validated through theoretical analysis, simulations, and hardware experiments. The contents of this chapter have been published in:

> **Yang, Shuo**, Zixin Zhang, Benjamin Bokser, and Zachary Manchester. "Multi-IMU Proprioceptive Odometry for Legged Robots." In 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 774-779. IEEE, 2023.

Chapter 6 This chapter focuses on state estimation but also explores connections to control. Building on the work presented in Chapters 4 and 5, we propose combining multiple IMUs, visual sensors, and leg kinematic sensors into the same odometry framework to achieve better long-term position estimation accuracy. This approach significantly upgrades the Cerberus algorithm, making it almost an order of magnitude more accurate than its previous version. Moreover, we systematically study how different control parameters, such as locomotion gait frequency, gait type, and commanded linear body movement speed, affect state estimation accuracy. This provides valuable insights into the co-design of control and estimation systems for legged robots.

Chapter 7 This chapter focuses on control but also has connections to state estimation. The core idea revolves around the factor graph, which has long been considered a state estimation tool. Common trajectory optimization methods, such as the Linear Quadratic Regulator (LQR), are essentially different forms of numerical optimization. Therefore, the factor graph can be used to solve control problems that are commonly known to be very difficult to address. The contents of this chapter have been published in:

Yang, Shuo, Gerry Chen, Yetong Zhang, Howie Choset, and Frank Dellaert. "Equality constrained linear optimal control with factor graphs." In 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 9717-9723. IEEE, 2021.

Chapter 2

Enhanced Model Predictive Control for Legged Robots With Reaction Wheels

This chapter presents a system development work that uses novel hardware innovations to improve legged robot control performance. We introduce a reaction wheel system that enhances the balancing capabilities and stability of quadrupedal robots during challenging locomotion tasks. Inspired by both the standard centroidal dynamics model common in legged robotics and models of spacecraft used in the aerospace community, we model the coupled quadruped-reaction-wheel system as a gyrostat and simplify the dynamics to formulate the problem as a linear discrete-time trajectory optimization problem. Modifications are made to a standard centroidal model-predictive control (MPC) algorithm to solve for both stance foot ground reaction forces and reaction wheel torques simultaneously. The MPC problem is posed as a quadratic program and solved online at 1000 Hz. We demonstrate improved attitude stabilization both in simulation and on hardware compared to a quadruped without reaction wheels, and perform a challenging traversal of a narrow balance beam that would be impossible for a standard quadruped. A video of our experiments is available online[1]. It is a perfect example that reflects our design philosophy: the reaction wheels do not mimic human or animal shapes directly, but they fundamentally improve the system's controllability. Moreover, Model Predictive Control (MPC) is an important application of numerical optimization in legged robots. It has close connections to all subsequent chapters.

2.1 Introduction

The core design of quadrupedal robots has largely converged over the past decade: high-performance contemporary designs share many similarities, including a rigid torso, four three-degree-of-freedom (3-DOF) legs, and a rounded "point foot" at the end of each leg. The most popular locomotion gait is the trotting gait, where a pair of diagonal feet switch contact states together (the gait pattern can be seen in Figure **??**). While simple and effective, during locomotion, only two feet are in contact with the ground at any given time, making the robot's orientation around the supporting line connecting the contact feet uncontrollable [60]. Therefore, large body orientation errors can only be eliminated by frequently switching stance feet [14, 60], making a quadrupedal robot especially vulnerable to impacts and disturbances during the two-foot standing phase.

In contrast, terrestrial animals use a multitude of strategies to perform inertial stabilization during dynamic movements: Humans heavily regulate their angular momentum through arm movements during locomotion [115], cheetahs use their tails

[1] `https://youtu.be/UroSaUg3Q6k`

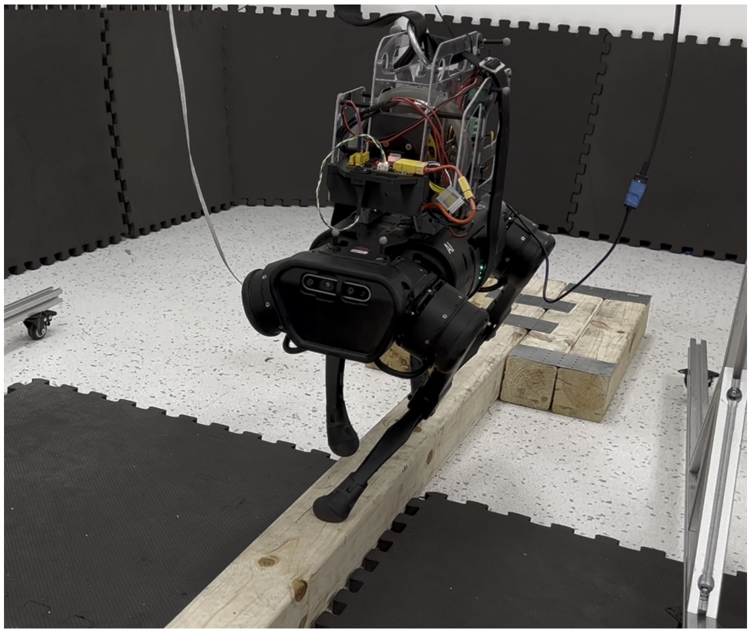


Figure 2-1: The Unitree A1 Quadruped walking on a six-centimeter-wide beam with the assistance of our reaction wheel actuator system and gyrostat MPC controller.

during high-speed chases and turning maneuvers [114], and falling cats adjust their attitude during falls using their highly flexible spines [100]. To improve the balancing and stabilization capabilities of quadrupedal robots, it is clear that we need to augment the current state-of-the-art quadruped design to enhance the robot's ability for inertial stabilization.

To bring similar capabilities to quadrupedal robots with a minimum of additional hardware and control complexity, with our design philosophy in mind, we take inspiration from the aerospace industry: Reaction wheel actuators (RWAs) are widely used on satellites to perform attitude control [45]. We develop a proof-of-concept RWA module that can be attached to the back of a standard Unitree A1 robot to provide additional angular momentum control. The 4.3-kg module, shown in Figure 2-1, is compact and has high control bandwidth.

RWAs offer a number of advantages over other mechanical appendages such as tails or multi-link limbs: Controllers do not need to consider the non-trivial collision-avoidance problem between appendages and the robot chassis. And, perhaps most importantly, RWAs lead to linearized dynamics that can be easily integrated into the standard centroidal model commonly used in MPC controllers for quadrupeds [37, 45]. One major drawback of RWAs is their reliance on wheel acceleration to provide body torque control, which can lead to saturation of rotor speed. In this chapter, we show that this limitation can be handled effectively by adding a set of linear inequality constraints in the MPC formulation, and a linear feedback term on reaction wheel momentum.

In this chapter, we modify the centroidal quadruped dynamics such that it captures the dynamics of the RWA system, and we use the new dynamics equation in the classic quadruped MPC framework to control all joint torques and wheel torques at the same time so that the robot orientation stays controllable during a two-leg stance phase. The method, which we call *Gyrostat MPC*, is validated in both simulation and hardware.

The chapter proceeds as follows: Section 2.2 discusses previous works related to inertial appendages and reaction wheels; Section 2.3 talks about the background ideas that we build upon in this chapter; Section 2.4 presents the mechanical design of the RWA module; Section 2.5 introduces the gyrostat MPC algorithm; and Section 2.6 presents our simulated and hardware results.

2.2 Related Work

In this section, we discuss some previous research results on quadruped balance control, related works on inertial appendages, and existing works on integrating

RWAs into legged robot locomotion.

2.2.1 Model-based Control of Legged robots

The classical optimal control framework finds control law of a given system by minimizing a optimality criterion. This framework can be realized using numerical optimization [10] as constrained nonlinear programs. The cost functions of such programs typically includes terms for energy efficiency, stability, and adherence to a desired trajectory or behavior. And the robot system dynamics are encoded as equality or inequality constraints [34, 74]. The MPC usually refers to the optimal control method of solving robot action from the current time to a future time after a certain "time horizon" as an optimization problem, and then send the first action to the robot as control. This is in contrast to iterative LQR (iLQR) [88] and differential dynamic programming (DDP) [142] where similar optimization problems are formulated and not only action but also a feedback control law is solved. Although MPC does not explicitly solves a feedback law, it is more flexible because different task constraints can be naturally formulated as constraints and the solution methods stay the same. In contrast, iLQR and DDP need to carefully consider constraints when solving the Ricatti equation [81, 143]. The expressiveness of MPC enables it to find state and action trajectories for very complicated systems even with discrete contact switches [96, 116].

A majority of recent MPC works address computation efficiency [37, 43]. The main idea is to leverage the inherent sparsity of MPC problem - each robot state along the solution trajectory is only affected by the two adjacent states. Therefore the matrix representation of the constraint of a MPC problem with long horizon is nearly block tridiagonal. So the problem sparsity can lead to efficient solution methods that only grow linearly with horizon length [37, 154].

2.2.2 Legged Robot Balance Strategies

The classical MPC for legged robots to perform body balancing using only ground reaction forces [13, 14, 30, 37, 105]. While these approaches have demonstrated incredible dynamic hardware behavior (including running, galloping, and jumping), quasi-static motion such as balancing in a highly under-actuated pose remains difficult. In [29], the authors balance a quadruped in a two-leg under-actuated pose. In [48], the authors demonstrate the same behavior with careful modeling of the robot in the balance pose and demonstrate narrow-beam walking in simulation. However, to the best of our knowledge, no hardware demonstration of continuous locomotion on a narrow beam — where the support polygon is nearly empty — has been achieved before. In addition to the control problem itself, hardware challenges such as deformable feet make this difficult to realize in practice as we going to show in Chapter 3.

2.2.3 Balancing Hardware

Inertial appendages have been widely explored for improving the stability of legged robots. A spine-like mechanism [129] can offer more control inputs to the system, but it is very complicated and difficult to manufacture. Tails, which can allow robots to stabilize their attitude in mid-air without ground contact, have received significant attention [89]. In [68, 114], the authors designed tails for hexapod and wheeled robots to perform aerial reorientation maneuvers. In [110], the authors leveraged the aerodynamics of tails. In [166], the authors introduced a sequential distributed MPC framework to stabilize legged robot locomotion with tails in simulation. In comparison to RWAs, tails can be more lightweight and energetically efficient [89]. However, RWAs offer simpler linearized dynamics and, unlike tail-based systems, do not encounter self collisions or joint limits.

There are numerous examples of adding RWAs to robots. Arguably one of the most well-known examples is the Cubli robot [46], which solely uses reaction wheels to achieve a variety of dynamic behaviors. RWAs have also been installed on bipeds to improve walking efficiency [21, 22] and to stabilize pitch during high-speed running [113]. On quadrupeds, RWAs have been studied for potential use in microgravity on the moon to perform attitude stabilization during jumping [79]. Most prior work on RWAs use simple proportional-derivative (PD) control laws to perform attitude stabilization, and do not reason about cross-coupling between the leg actuators and RWAs. In contrast, we present a new convex MPC algorithm that simultaneously optimizes both leg and RWA control inputs.

2.3 Background

We now review some concepts from legged robot contorl and state estimation with a set of notations that will be used throughout the entire thesis.

In general, we use lowercase letters for scalars and frame abbreviations, boldface lowercase letters for vectors, and upper case letters for matrices and vector sets. The operation $[a; b; c]$ vertically concatenates elements a, b and c with the same type (scalar, vector, or matrix). The operator $\lfloor \boldsymbol{v} \rfloor^{\times}$ converts a vector $\boldsymbol{v} = [v_1; v_2; v_3] \in \mathbb{R}^3$

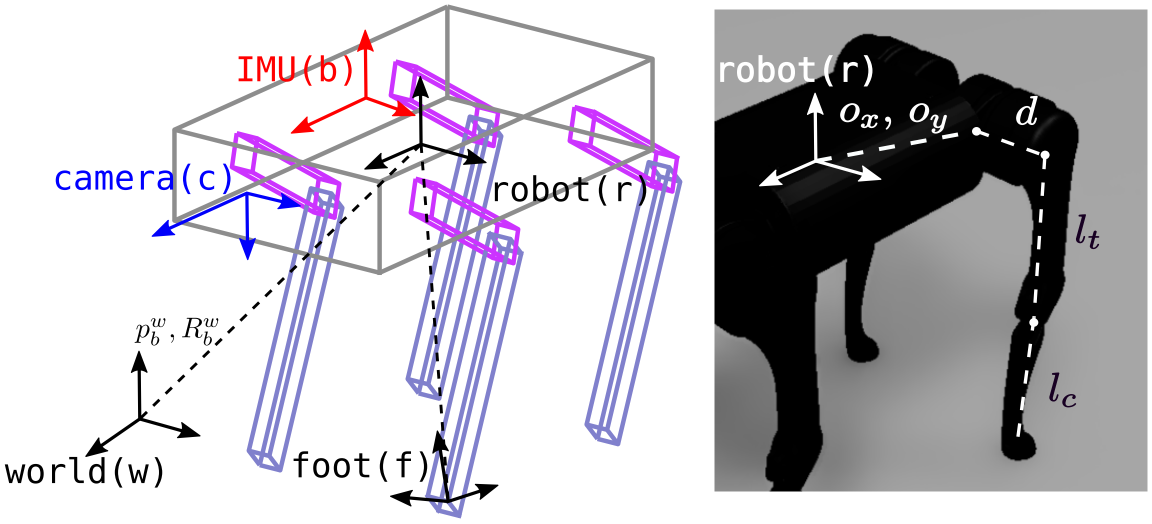


Figure 2-2: Frames & Kinematic parameters of A1 robot. For each leg, its hip XY offsets to robot body center are represented by o_x and o_y. d, l_t and l_c are kinematic offsets from hip to knee and knee to foot.

into the skew-symmetric "cross-product matrix,"

$$\lfloor \boldsymbol{v} \rfloor^{\times} = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix}, \tag{2.1}$$

such that $\boldsymbol{v} \times \boldsymbol{x} = \lfloor \boldsymbol{v} \rfloor^{\times} \boldsymbol{x}$. $\dot{\boldsymbol{a}}$ is the time derivative of $\boldsymbol{a}$.

2.3.1 Coordinate Frames & Rotation Representation

Important coordinate frames are shown in Fig. 2-2. To simplify discussion, we assume that the IMU frame and the robot's body frame coincide. We use $\boldsymbol{p}$ to denote the translation vector and R to denote the coordinate transformation matrix from the robot's body frame to the world frame. The matrix R can also be viewed as a representation of robot orientation, which is an element of special orthogonal group

$SO(3)$ [91, 103]. Where necessary, we use superscripts and subscripts to explicitly indicate the frames associated with rotation matrices and vectors, so $R_b^a \cdot \boldsymbol{p}^b$ means the matrix transforms a vector $\boldsymbol{p}^b$ represented in coordinate frame b into coordinate frame a [103]. For brevity, if frame b is time-varying, and the context of frame association is clear, we write R_k instead of $R_{b(k)}^w$ to indicate the rotation matrix is also time dependent. Similarly, $\boldsymbol{p}_k$ defines a time-varying vector, it can also be viewed as the origin vector of frame b_k in the world. The rotation matrix R also has properties such as $R^\top R = I$ and $\det(R) = 1$, therefore an inverse transformation from frame a into frame b satisfies $R_a^b = R_b^{aT} = R_b^{a^{-1}}$.

The rotation matrix is intuitively easy to understand and use in practice, but it is not an ideal choice for numerical optimization because it consists of 9 parameters. Two other compact representations, the Euler Angles [37] and quaternions [65] are preferred in control and estimation related optimizations. In this chapter we focus on the Euler Angles but defer a detailed introduction to quaternions to later chapters.

Euler Angles

Although there are multiple Euler Angles conventions, we parameterize the robot's orientation using Z-Y-X angles [67] (alternatively called Yaw-Pitch-Roll angles or Tait–Bryan angles). Specifically, $\boldsymbol{\theta} = [\theta_r; \theta_p; \theta_y]$ represents an orientation, where θ_y, θ_p, θ_r are commonly referred as yaw angle, pitch angle, and roll angle respectively. We denote $\cos(\theta_i) = c_i$ and $\sin(\theta_i) = s_i$ for $i \in \{y, p, r\}$. We can construct a rotation matrix from these angles:

$$R(\boldsymbol{\theta}) = R_z(\theta_y)R_y(\theta_p)R_x(\theta_r) = \begin{bmatrix} c_p c_y & c_y c_p c_r - c_r c_y & c_r c_y + c_r c_y c_p \\ c_p c_y & c_r c_y + c_p c_r c_y & c_r c_p c_y - c_y c_r \\ -c_p & c_p c_r & c_p c_r \end{bmatrix}, \tag{2.2}$$

which transforms a vector from the robot body (b) frame to the world inertial (w) frame [103]. More details can be seen at [67].

The derivative of the Z-Y-X angles $\dot{\boldsymbol{\theta}}$ is related to the robot body angular velocity $\boldsymbol{\omega}^b$ through the following equation:

$$\dot{\boldsymbol{\theta}} = \begin{bmatrix} \dot{\theta}_r \\ \dot{\theta}_p \\ \dot{\theta}_y \end{bmatrix} = \Omega(\boldsymbol{\theta})\boldsymbol{\omega}^b = \begin{bmatrix} 1 & (s_p s_r)/c_p & (c_r s_p)/c_p \\ 0 & c_r & -s_r \\ 0 & s_r/c_p & c_r/c_p \end{bmatrix} \boldsymbol{\omega}^b, \tag{2.3}$$

where $\Omega(*)$ is called “Euler’s kinematic” relation [136].

Leg Forward Kinematics

An important concept in multi-rigid body systems is forward kinematics, which describes the relationship between joint angles and the position of the end-effector. For the jth leg of a legged robot, we define $\boldsymbol{\phi}$ as a vector containing all joint angles, and $\dot{\boldsymbol{\phi}}$ as the joint angle velocities. The forward kinematics function is denoted as $\boldsymbol{p}_f = g(\boldsymbol{\phi}) \in \mathbb{R}^3$, whose output is the foot position in the robot body frame. The derivative of this equation leads to the kinematic Jacobian matrix $J(\boldsymbol{\phi})$ that maps $\dot{\boldsymbol{\phi}}$ into the foot’s linear velocity in the body frame:

$$\boldsymbol{v}_f = \dot{\boldsymbol{p}}_f = J(\boldsymbol{\phi})\dot{\boldsymbol{\phi}}. \tag{2.4}$$

As an concrete example, the forward kinematics function g of a leg of a Unitree

A1 robot with $\boldsymbol{\phi} = [\phi_1; \phi_2; \phi_3]$ and kinematic parameters shown in Figure 2-2 is

$$g(\boldsymbol{\phi}) = \begin{bmatrix} o_x - l_c s_{23} - l_t s_2 \\ o_y + d c_1 + l_t c_2 s_1 + l_c s_1 c_{23} \\ d s_1 - l_t c_1 c_2 - l_c c_1 c_{23} \end{bmatrix}, \tag{2.5}$$

where s_i denotes $\sin(\phi_i)$ and $c_i = \cos(\phi_i)$, where $i = 1, 2$. Also $s_{23} = \sin(\phi_2 + \phi_3)$ and $c_{23} = \cos(\phi_2 + \phi_3)$. The expression is derived using the product of exponentials (POE) method [103]. The kinematic Jacobian of g is

$$J(\boldsymbol{\phi}) = \begin{bmatrix} 0 & -l_c c_{23} - l_t c_2 & -l_c c_{23} \\ l_t c_1 c_2 - d s_1 + l_c c_1 c_{23} & -s_1(l_c s_{23} + l_t s_2) & -l_c s_{23} s_1 \\ l_t c_2 s_1 + d c_1 + l_c s_1 c_{23} & c_1(l_c s_{23} + l_t s_2) & l_c s_{23} c_1 \end{bmatrix}, \tag{2.6}$$

2.3.2 Centroidal MPC

The robot control problem aims to govern key robot states such as position, velocity, and orientation to achieve a desired behavior. A legged robot can only achieve balancing and locomotion by switching contact conditions of legs within a gait cycle. A foot goes through stance phase and swing phase periodically.

Our controller is built on the widely used convex centroidal MPC framework [14, 37]. Assuming the robot has heavy body and light weight limbs, the body dynamics can be approximated as a single rigid body with a point mass at the centroid (which coincides with the center of mass) and the legs are modeled as point mass that can be governed by conventional PD controller [91]. With these assumptions it is very convenient to decouple the controller into two sub-problems: swing leg tracking and body balancing control.

During swing phase, a swing foot's position $\boldsymbol{s}(t)$ moves from lifting up location to a desired foothold location $\boldsymbol{s}^d$ through a smooth curve. The desired foothold location is selected so that the robot get well supported at next contact switch. A standard selection strategy choose $\boldsymbol{s}^d$ using the Raibert heuristic [120] such that

$$\boldsymbol{s}^d = \bar{\boldsymbol{s}} + \boldsymbol{v} t_s / 2, \tag{2.7}$$

where $bar\boldsymbol{s}$ is a nominal foot position on the ground right beneath the corresponding leg hip, t_s is the swing phase duration, and $\boldsymbol{v}$ is the robot body velocity in the world frame. Foot position $\boldsymbol{s}(t)$ is then tracked using an end-effector PD controller [37].

The most critical part of the control problem is body balancing control, which uses an MPC controller with various optimization techniques to achieve fast solution times. The robot dynamics model, called the *centroidal model*, neglects the inertia of the robot's legs and treats the robot as a single rigid body subjected to ground reaction forces produced by the feet. Given a body mass m and body moment of inertia expressed in the body frame J^b, the continuous time dynamic equations of the centroidal model are

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{\theta}} \\ \ddot{\boldsymbol{p}} \\ \dot{\boldsymbol{\omega}}^b \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \Omega(\boldsymbol{\omega})\boldsymbol{\omega}^b \\ \frac{1}{m}\boldsymbol{f}_c - \boldsymbol{g}^w \\ (J^b)^{-1}(\boldsymbol{\tau}_c - \boldsymbol{\omega}^b \times J^b \boldsymbol{\omega}^b) \end{bmatrix}, \tag{2.8}$$

where the state of the system includes of the center of mass position $\boldsymbol{p} \in \mathbb{R}^3$, Euler angle representation of the body attitude $\boldsymbol{\theta} \in \mathbb{R}^3$, world-frame linear velocity $\boldsymbol{v} \in \mathbb{R}^3$, and body-frame angular velocity $\boldsymbol{\omega}^b \in \mathbb{R}^3$. The input of the system contains a world-frame force input $\boldsymbol{f}_c \in \mathbb{R}^3$ and a body-frame torque input $\boldsymbol{\tau}_c \in \mathbb{R}^3$ coming from ground reaction wrench at the body centroid. The input force and torque are mapped from the ground reaction forces $\boldsymbol{f}_i = [f_x, f_y, f_z]^T$ generated by each foot at position $\boldsymbol{s}_i$ for foot $i \in 1 \dots L$, where L is the number of total legs, through the following relationship:

$$\begin{bmatrix} \boldsymbol{f}_c \\ \boldsymbol{\tau}_c \end{bmatrix} = \begin{bmatrix} I_3 & \dots & I_3 \\ R(\boldsymbol{\theta})^T \lfloor \boldsymbol{s}_1 \rfloor^{\times} & \dots & R(\boldsymbol{\theta})^T \lfloor \boldsymbol{s}_n \rfloor^{\times} \end{bmatrix} \boldsymbol{u}. \tag{2.9}$$

where $\boldsymbol{u} = \begin{bmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_n \end{bmatrix}$ is the control input, and $I_n \in \mathbb{R}^{n \times n}$ denotes an n by n identity matrix. The ground reaction forces are further subjected to friction-cone constraints to prevent foot slip on a surface with friction coefficient μ. Often, the friction-cone constraint is approximated as a pyramid, which enables the constraints to be expressed as a set of linear inequalities:

$$\begin{aligned} -\mu f_z \leq & f_x \leq \mu f_z \\ -\mu f_z \leq & f_y \leq \mu f_z. \end{aligned} \tag{2.10}$$

With the model described in (2.8) and (2.9), the MPC controller enforces the dynamics as a set of constraints in a trajectory optimization problem that is solved online. The problem can be convexified and formulated as a quadratic program (QP), as explained in [37]. We briefly review the problem formulation and provide a new way of visualization of MPC problems, which will facilitate our discussion in subsequent chapters.

Mathematically, the most general MPC problem can be formulated as

$$\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{i=0}^{k-1} g(\boldsymbol{x}_i, \boldsymbol{u}_i) \\
\text{subject to} \quad & \boldsymbol{x}_{i+1} = f(\boldsymbol{x}_i, \boldsymbol{u}_i),\ i = 0 \ldots k-1 \\
& c(\boldsymbol{x}_i, \boldsymbol{u}_i) \leq 0,\ i = 0 \ldots k-1 \\
& \boldsymbol{x}_0 = \boldsymbol{x}_{\text{init}}
\end{aligned} \tag{2.11}$$

in which $g()$ is a cost function, $f()$ is the discrete dynamics function, $c()$ represents inequality constraint and equality constraints. Moreover, we assume the robot has a state estimator that estimates the robot state at time 0 as $\boldsymbol{x}_{\text{init}}$. Once the MPC problem solution is found, the solved $\boldsymbol{u}_0$ will be used to control the robot. The ground reaction forces in the control input $\boldsymbol{u}$ are then mapped to the leg joint torques through the transpose of the kinematic Jacobian.

Problem formulation 2.11 aligns with many conventional numerical optimization problems [108], thus can be solved by many existing numerical methods. However, the problem might take a long solver time if cost or dynamics functions are nonconvex. Luckily, the MPC problem for legged robots can be convexified by linearizing the dynamics and cost functions around a nominal trajectory $\boldsymbol{x}^{ref}$ [37] as follows

$$\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{i=0}^{k-1} \left\|\boldsymbol{x}_{i+1} - \boldsymbol{x}_{i+1}^{\text{ref}}\right\|_{Q_i} + \|\boldsymbol{u}_i\|_{R_i} \\
\text{subject to} \quad & \boldsymbol{x}_{i+1} = A_i \boldsymbol{x}_i + B_i \boldsymbol{u}_i,\ i = 0 \ldots k-1 \\
& \underline{\boldsymbol{c}}_i \leq C_i \boldsymbol{u}_i \leq \bar{\boldsymbol{c}}_i,\ i = 0 \ldots k-1 \\
& D_i \boldsymbol{u}_i = 0,\ i = 0 \ldots k-1 \\
& \boldsymbol{x}_0 = \boldsymbol{x}_{\text{init}}
\end{aligned} \tag{2.12}$$

where the cost function is quadratic, the discrete dynamics function is obtained by linearizing (2.8) and (2.9) around $\boldsymbol{x}^{ref}$. The inequality constraint is a rewritten version of (2.10). The equality constraint means that robot legs that are not in contact with the ground should have 0 forces. More details can be seen in [37]

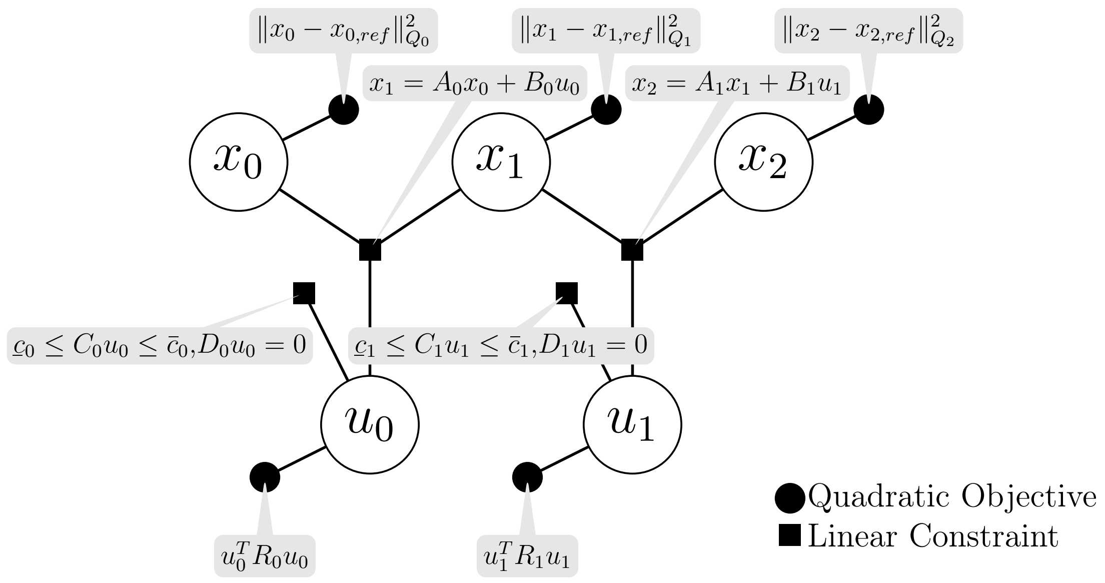
We want to emphasize the problem sparsity. First, it is possible to express Problem (2.12) graphically as Figure 2-3. This is not merely a visualization but also a way to unveil the underlying sparsity structure of the problem. It is known that a general quadratic programming problem

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \frac{1}{2}\boldsymbol{x}^T Q \boldsymbol{x} + \boldsymbol{q}^T \boldsymbol{x} \\ \text{subject to} \quad & H\boldsymbol{x} - \boldsymbol{h} = 0 \\ & G\boldsymbol{x} \leq 0 \end{aligned} \tag{2.13}$$

can be solved by the Karush–Kuhn–Tucker (KKT) system

$$\begin{bmatrix} Q & H^T & G^T \\ H & 0 & 0 \\ G & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\lambda} \\ \boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{q} \\ \boldsymbol{h} \\ 0 \end{bmatrix}, \tag{2.14}$$

where $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$ are the Lagrange multipliers. For Problem (2.12), the KKT system can be seen to be a sparse linear system, and the sparsity pattern is exactly the same as the graph structure shown in Figure 2-3. For example, if we arrange the decision variables of Problem (2.12) as $\boldsymbol{x} = [\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \ldots, \boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}]^T$, then the



$\|x_0 - x_{0,ref}\|^2_{Q_0}$
$\|x_1 - x_{1,ref}\|^2_{Q_1}$
$\|x_2 - x_{2,ref}\|^2_{Q_2}$
$x_1 = A_0x_0 + B_0u_0$
$x_2 = A_1x_1 + B_1u_1$
x_0
x_1
x_2
$\underline{c}_0 \leq C_0u_0 \leq \bar{c}_0, D_0u_0 = 0$
$\underline{c}_1 \leq C_1u_1 \leq \bar{c}_1, D_1u_1 = 0$
u_0
u_1
$u_0^T R_0 u_0$
$u_1^T R_1 u_1$
● Quadratic Objective
■ Linear Constraint

Figure 2-3: The graphical representation of a MPC problem

G matrix in the KKT system can be written as

$$G = \begin{bmatrix} A_1 & B_1 & -I & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & A_2 & B_2 & -I & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & A_{k-1} & B_{k-1} & I \end{bmatrix} \tag{2.15}$$

Moreover, the internal structure of A_i and B_i may also be sparse, since each leg of a legged robot individually contributes to the robot dynamics. Exploiting the sparsity can lead to very efficient solver.

2.3.3 Gyrostat Dynamics

The dynamics of a rigid body with RWAs can be modeled as a gyrostat [45]. A gyrostat is a system of coupled rigid bodies whose relative motions do not change the total inertia tensor of the system, however, the rotation of these RWAs will generate additional torques and angular momentums that need to be considered in control. A set of k reaction wheels with perfect static and dynamic balance have a constant inertia in the body frame of the robot that can be expressed as

$$\mathrm{J}_r = \sum_{i=1}^{k} \mathrm{L}_i \tag{2.16}$$

where $\mathrm{L}_i \in \mathbb{R}^{3\times 3}$ is the inertia of the i-th wheel expressed in the robot's body frame. The total angular momentum of the wheels in the body frame of the robot is

$$\boldsymbol{h}_r = \sum_{i=1}^{k} \mathrm{L}_i(\boldsymbol{\omega}^b + \boldsymbol{\psi}_i), \tag{2.17}$$

where $\boldsymbol{\omega}_i^b$ is the body angular velocity of the robot and $\boldsymbol{\psi}_i \in \mathbb{R}^3$ is the angular velocity of wheel i in the robot's body frame. The total angular momentum of the robot $\boldsymbol{h}$ is then

$$\boldsymbol{h} = J_b\boldsymbol{\omega}^b + \boldsymbol{h}_r \tag{2.18}$$

$$= (J_b + \mathrm{J}_r)\boldsymbol{\omega}^b + \sum_{i=1}^{k} \mathrm{L}_i\boldsymbol{\psi}_i. \tag{2.19}$$

Taking the derivative of the angular momentum yields the equation of motion for the gyrostat,

$$\boldsymbol{\tau}_c = \mathrm{J}\dot{\boldsymbol{\omega}}^b + \boldsymbol{\omega}^b \times (\mathrm{J}\boldsymbol{\omega}^b + \sum_{i=1}^{k} \mathrm{L}_i\boldsymbol{\psi}_i) + \boldsymbol{\tau}_r, \tag{2.20}$$

where $\mathrm{J} \in \mathbb{R}^{3\times 3}$ is the sum of J_b and J_r, $\boldsymbol{\tau}_c \in \mathbb{R}^3$ is the total body-frame external torque on the robot, and $\boldsymbol{\tau}_r \in \mathbb{R}^3$ is the net body-frame torque exerted by the RWAs. Depending on the RWAs configuration, the wheel-frame angular momentum $\boldsymbol{\rho} \in \mathbb{R}^k$

can be mapped to this equation with a constant jacobian matrix $\Lambda_r \in \mathbb{R}^{k\times 3}$ and the input torques along the wheel axis $\boldsymbol{u}_r \in \mathbb{R}^k$ such that

$$\sum_{i=1}^{k} \mathrm{L}_i \boldsymbol{\psi}_i = \Lambda_r^T \boldsymbol{\rho} \tag{2.21}$$

$$\boldsymbol{\tau}_c = \mathrm{J}\dot{\boldsymbol{\omega}}^b + \boldsymbol{\omega}^b \times (\mathrm{J}\boldsymbol{\omega}^b + \Lambda_r^T \boldsymbol{\rho}) + \Lambda_r^T \boldsymbol{u}_r. \tag{2.22}$$

This can be written in the standard manipulator form [91],

$$M\dot{\boldsymbol{v}} + C(\boldsymbol{q}, \boldsymbol{v}) + \boldsymbol{\tau}_{ext} = B(\boldsymbol{q})\boldsymbol{u} \tag{2.23}$$

where the mass matrix M, dynamic bias $C(\boldsymbol{q}, \boldsymbol{v})$, and control mapping $B(\boldsymbol{q})$ for generalized configuration $\boldsymbol{q}$, generalized velocity $\boldsymbol{v}$, external torque $\boldsymbol{\tau}_{ext}$, and control $\boldsymbol{u}$ are defined as

$$\underbrace{\begin{bmatrix} \mathrm{J} & 0 \\ 0 & I_k \end{bmatrix}}_{M} \begin{bmatrix} \dot{\boldsymbol{\omega}}^b \\ \dot{\boldsymbol{\rho}} \end{bmatrix} + \underbrace{\begin{bmatrix} \boldsymbol{\omega}^b \times (\mathrm{J}\boldsymbol{\omega}^b + \Lambda_r^T \boldsymbol{\rho}) \\ 0 \end{bmatrix}}_{C(\boldsymbol{q}, \boldsymbol{v})} + \underbrace{\begin{bmatrix} -\boldsymbol{\tau}_c \\ 0 \end{bmatrix}}_{\boldsymbol{\tau}_{ext}} = \underbrace{\begin{bmatrix} -\Lambda_r^T \\ I_k \end{bmatrix}}_{B(\boldsymbol{q})} \begin{bmatrix} \boldsymbol{u}_r \end{bmatrix}. \tag{2.24}$$

2.4 Hardware Design

This section presents the specifics of the mechanical design of the RWA module, which is pictured in Fig. 2-4. We use only two RWAs, one for the pitch axis and one for the roll axis. An RWA for the yaw axis is not included as yaw control is much less important for stability of the robot. The RWAs are driven by two permanent-magnet synchronous motors, each with a continuous maximum current draw of 60A, a maximum speed of 1900 RPM, and a maximum torque output of 4.7 Nm. The

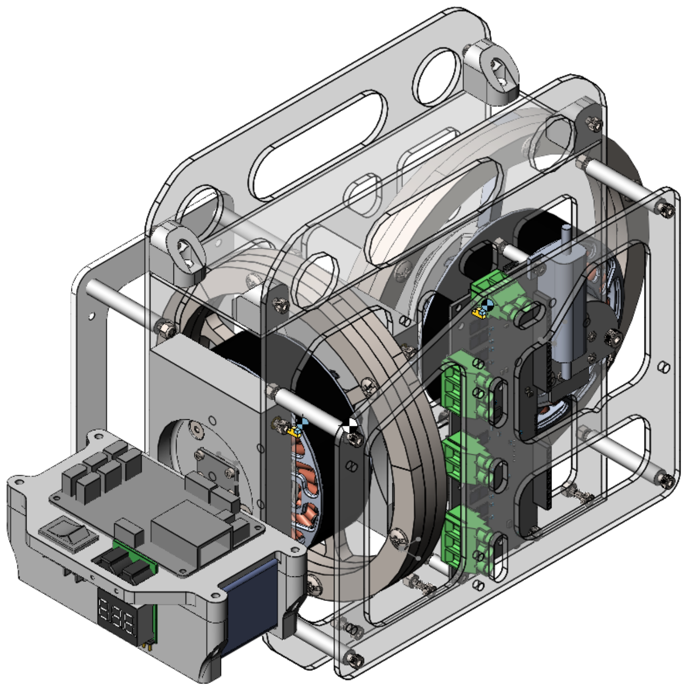


Figure 2-4: Proof-of-concept RWA system that mounts directly to the back of a Unitree A1 robot. The system includes two permanent-magnet synchronous motors that drive two high-inertia flywheels along the roll and pitch axes up to a maximum speed of 1900 RPM. The system contains its own battery and is capable of generating a maximum continuous torque of 5 Nm along each axis.

chassis is composed of polycarbonate plates and 3D-printed polylactic acid (PLA). The overall system has dimensions of 100 $\times$ 210 $\times$ 300 mm and a total mass of 4.3 kg, including a 2200 mAh lithium polymer battery pack. Variables such as efficiency, weight, and dimension were not optimized for this prototype. As demonstrated in Section 2.6, despite the increase in total mass and the change in inertial properties caused by the addition of the RWA module, the module still significantly improves the stabilizing capabilities of the robot.

2.5 Gyrostat MPC

This section describes an MPC framework that jointly solves for RWA torques and ground-reaction forces for a quadruped equipped with our RWA module. We incorporate the RWA dynamics into the centroidal quadruped controller using the gyrostat model, and we linearize the model to use it as a part of a convex discrete-time trajectory optimization problem.

2.5.1 Gyrostat Quadruped Dynamics

Our RWA module adds two RWAs that provide control over the body angular momentum in the pitch and roll axes. Combining the centroidal dynamics from (2.8) and the RWA rotational dynamics from (2.24), we obtain the following equations of motion:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{\theta}} \\ \ddot{\boldsymbol{p}} \\ \dot{\boldsymbol{\omega}}^b \\ \dot{\boldsymbol{\rho}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \Omega(\boldsymbol{\omega})\boldsymbol{\omega}^b \\ \frac{1}{m}\boldsymbol{f}_c - \boldsymbol{g}^w \\ \mathrm{J}^{-1}(\boldsymbol{\tau}_c + \Lambda_r^T u_r - \boldsymbol{\omega}^b \times (\mathrm{J}\boldsymbol{\omega}^b + \Lambda_r^T \boldsymbol{\rho})) \\ \boldsymbol{u}_r \end{bmatrix}, \tag{2.25}$$

where, in addition to the original centroidal model states, we introduce the RWA momentum state vector $\rho \in \mathbb{R}^{2\times 1}$ and torque input vector $u_r \in \mathbb{R}^{2\times 1}$. The control input vector $u \in \mathbb{R}^{14\times 1}$ for the whole system is now expressed as:

$$\begin{bmatrix} \boldsymbol{f}_c \\ \boldsymbol{\tau}_c \\ u_r \end{bmatrix} = \begin{bmatrix} I_3 & \dots & I_3 & 0_{32} \\ R(\boldsymbol{\theta})^T \lfloor \boldsymbol{s}_1 \rfloor^\times & \dots & R(\boldsymbol{\theta})^T \lfloor \boldsymbol{s}_n \rfloor^\times & 0_{32} \\ 0_{23} & \dots & 0_{23} & I_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_n \\ \boldsymbol{u}_r \end{bmatrix}. \tag{2.26}$$

We linearize equations (2.25) about a nominal yaw angle θ_y as in [14]. We simplify the rotation matrix $R(\boldsymbol{\theta})$ as $R_z(\theta_y)$ from (2.2) using small angle approximation, and we ignore the Coriolis term in the rotational dynamics assuming that body angular velocity is small. We also assume that the RWAs' velocity are small during stable trotting, and we eliminate the same Coriolis term in the Gyrostat dynamics, reducing the rotation dynamics from (2.26) to where 0_{nm} represents a $n \times m$ matrix with all zero elements, and 0_n represents a $n \times n$ square matrix with all zero elements. We now linearize (2.25) for a convex MPC formulation assuming that attitude displacement, body angular velocity, and RWA velocities are small during normal operating conditions. This allows us to parameterize rotation using $R_z(\theta_y)$ and simplify the rotational kinematics and dynamics from (2.25) to

$$\begin{aligned} \dot{\boldsymbol{\theta}} &\approx R_z^T(\theta_y)\boldsymbol{\omega}^b \\ \dot{\boldsymbol{\omega}}^b &\approx \mathrm{J}^{-1}(\boldsymbol{\tau}_c + \boldsymbol{u}_r). \end{aligned} \tag{2.27}$$

Finally, we convert the angular velocities into the inertial frame, and the resulting linearized dynamics for the gyrostat quadruped model becomes

$$
\begin{aligned}
\frac{d}{dt}\begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{\theta} \\ \dot{\boldsymbol{p}} \\ \boldsymbol{\omega}^b \\ \boldsymbol{\rho} \end{bmatrix} &= \begin{bmatrix} 0_3 & 0_3 & I_3 & 0_3 & 0_{32} \\ 0_3 & 0_3 & 0_3 & R_z^T(\theta_y) & 0_{32} \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_{32} \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_{32} \\ 0_{23} & 0_{23} & 0_{23} & 0_{23} & 0_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{\theta} \\ \dot{\boldsymbol{p}} \\ \boldsymbol{\omega}^b \\ \boldsymbol{\rho} \end{bmatrix} + \begin{bmatrix} 0_{31} \\ 0_{31} \\ -\boldsymbol{g}^w \\ 0_{31} \\ 0_{21} \end{bmatrix} \\
&+ \begin{bmatrix} 0_3 & \dots & & 0_{32} \\ 0_3 & \dots & & 0_{32} \\ \frac{I_3}{m} & \dots & \frac{I_3}{m} & 0_{32} \\ \mathrm{J}^{-1}R(\boldsymbol{\theta})^T\lfloor \boldsymbol{s}_1 \rfloor^\times & \dots & \mathrm{J}^{-1}R(\boldsymbol{\theta})^T\lfloor \boldsymbol{s}_n \rfloor^\times & \mathrm{J}^{-1}R(\boldsymbol{\theta})^T\Lambda_r^T \\ 0_{23} & \dots & & I_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_n \\ \boldsymbol{u}_r \end{bmatrix},
\end{aligned} \tag{2.28}
$$

where $\boldsymbol{g}^w \in \mathbb{R}^3$ represents the gravity vector. The dynamics is now in a linearized form parameterized by the yaw angle θ_y and foot positions $\boldsymbol{s}_i$ such that

$$\dot{\boldsymbol{x}}(t) = A(\theta_y)\boldsymbol{x}(t) + B(\boldsymbol{s}_1, \dots, \boldsymbol{s}_n, \theta_y)\boldsymbol{u}(t). \tag{2.29}$$

2.5.2 MPC Problem

The problem is now linearized with the continuous-time transition and control matrices A and B. We convert those matrices into discrete-time $\mathbb{A}$ and $\mathbb{B}$ matrices. This control problem is then posed as a discrete-time convex quadratic program as Probme (2.12): where $\boldsymbol{x}_i, \boldsymbol{u}_i \in \mathbb{R}^{14\times 1}$, $Q_i, R_i \in \mathbb{R}^{14\times 14}$ are the state of the robot, control inputs to the robot, and cost matrices for state and control inputs at time step i, respectively. The matrices C_i in (2.12) are used to enforce linearized friction cone constraints from (2.10). We also regularize the speed of the RWAs inside the dynamics penalty term and constrain the RWA torques and speeds with the affine constraints

$$\underline{\boldsymbol{u}}_r \leq \boldsymbol{u}_r \leq \bar{\boldsymbol{u}}_r \tag{2.30}$$

$$\underline{\boldsymbol{\rho}} \leq \boldsymbol{\rho}_i \leq \bar{\boldsymbol{\rho}}, i = \dots k-1, \tag{2.31}$$

where $\bar{\boldsymbol{u}}_r$, $\underline{\boldsymbol{u}}_r$, $\bar{\boldsymbol{\rho}}_r$, $\underline{\boldsymbol{\rho}}_r$ represents the upper bound and lower bound of the RWAs control torque and momentum, respectively. The equality constraints in (2.12) are used to constrain foot forces to be zero when a foot is in the swing phase. The solution to the QP problem (2.12) returns the ground reaction forces for each of the feet in contact with the ground and the torques to be applied to the RWAs. Finally we convert the ground reaction forces into joint torques using the kinematic jacobian (2.4)

$$\boldsymbol{\tau}_i = J(\boldsymbol{\phi}_i)_i R(\boldsymbol{\theta})^T \boldsymbol{f}_i, \tag{2.32}$$

where $\boldsymbol{\tau}_i \in \mathbb{R}^3$, $\boldsymbol{\phi}_i \in \mathbb{R}^3$ and $\boldsymbol{f}_i \in \mathbb{R}^3$ are the joint torques, joint angles, and ground reaction forces for leg i, respectively.

2.5.3 Angular Momentum Error Feedback

During online MPC execution, we introduce an error feedback term on the angular momentum of the roll RWA such that

$$\theta_r^d = \bar{\theta}_r^d + k_r \rho_r, \tag{2.33}$$

where $\bar{\theta}_r^d$ is the nominal desired roll angle that is usually set to 0, k_r is a feedback gain for the roll RWA momentum, ρ_r is the roll RWA momentum, and θ_r^d is the final desired roll angle used for MPC. We found that this feedback term is essential to avoid RWA rotor speed saturation during the hardware experiments. There are a number of factors that contribute to RWA saturation during the hardware experiments, including center-of-mass mismatch between simulated and physical model, and biases in the state estimation. This feedback term, along with the inequality constraints in (2.12), keeps the RWA from saturating due to model mismatches and disturbances.

2.6 Experiments and Results

We now present the simulation and hardware experiment results for the gyrostat MPC on a Unitree A1 robot equipped with our RWA module. In simulation, we tested the system's disturbance rejection and aerial reorientation abilities. On hardware, we tested the system's balancing capability through a beam walking experiment. To the best of our knowledge, the experiment presented in this chapter is the first successful hardware demonstration of a beam walk done by a quadruped robot, where the robot has to continuously balance itself with a near-empty support polygon.

2.6.1 Hardware/Simulation Setup

We built the gyrostat MPC controller on top of an open-source convex MPC implementation for the Unitree A1 robot[2]. The MPC problem from (2.12) is solved using

[2]`https://github.com/ShuoYangRobotics/A1-QP-MPC-Controller`

OSQP [138] online at 1000Hz on a computer equipped with an AMD Threadripper 3 CPU. The MPC employs a 20-step horizon and a 0.05 s time step, and we design a controller stack that uses the same MPC code to control either a hardware robot or a simulated robot. On hardware, the control torque command is sent to the robot joint motors and the RWA drivers via Ethernet. In the simulation, the controller interfaces with the Gazebo simulator [77]. To reduce the sim-to-real gap, the Gazebo simulation uses the precise model mass and inertial parameters from the robot manufacturer. It also uses a motor dynamics simulation, a fine-tuned soft foot contact model, and accurate sensor noise injections according to sensor data sheets.

2.6.2 Locomotion Disturbance Rejection

In the simulated disturbance rejection tests, we applied a 600 N, 650 N, and 700 N force to the y-axis of the robot body while trotting. The impulse used in our experiment is defined as a continuous force disturbance that lasts 0.05 s, and the disturbance is applied at precisely the same gait phase for each test. The same impulse is applied to a robot equipped with the RWA module but running the base centroidal MPC controller [37] and the same robot running the gyrostat MPC controller. Figure 2-5 shows the roll error response of both robots during the 650 N impulse test in Gazebo. The maximum roll error is reduced by 26% with faster steady-state convergence. The experiments demonstrate an enhanced ability to recover from sudden impacts due to better inertial stabilization when the robot is airborne. During the 700 N impulse experiments, the gyrostat MPC controller consistently recovers from the impact while the base centroidal MPC controller fails. Figure 2-6 demonstrates this disturbance rejection behavior on hardware, though without precisely quantified impulses.

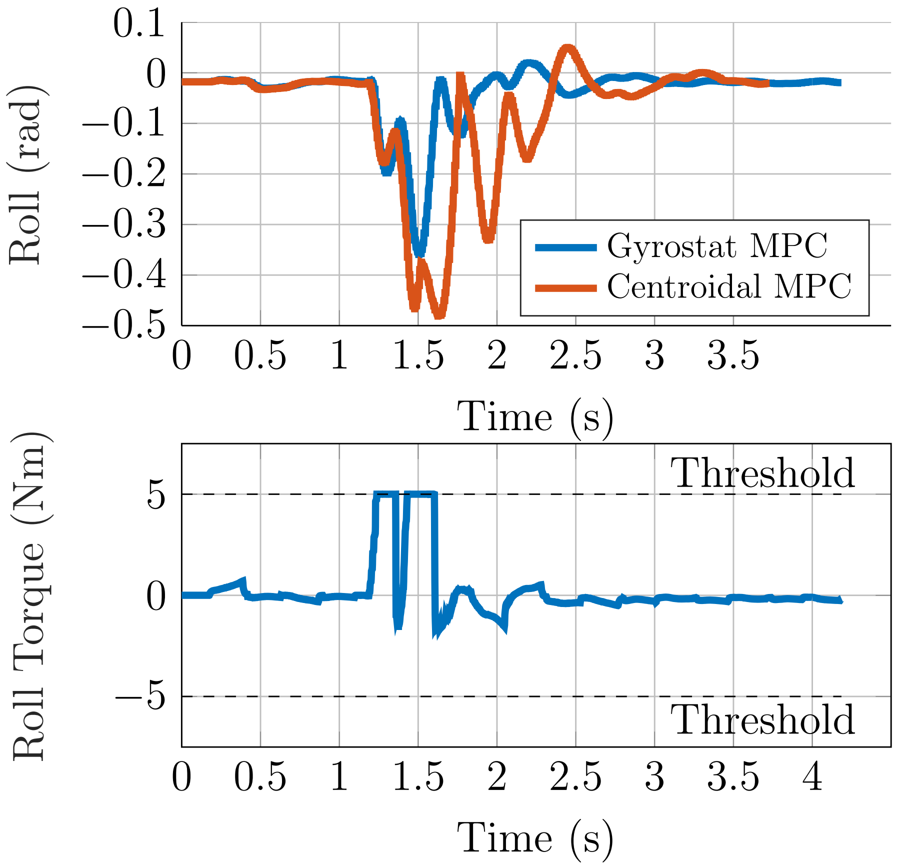


Figure 2-5: Roll error (top) and RWA torque responses (bottom) to a 650 N impulse applied to the robot's body y-axis at $t = 1.2$ seconds.

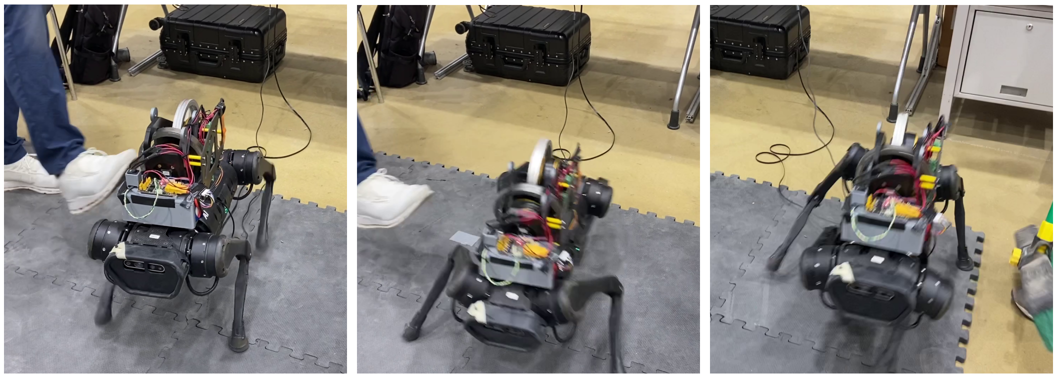

(a) (b) (c)

Figure 2-6: Hardware impulse test where we provide an impulse force on the robot during locomotion with a kick. 2-6a shows the robot performing stable trotting when the impulse is applied. 2-6b shows the robot losing balance on the footholds while maintaining a stable attitude as it eventually recovers from the impulse in 2-6c.

.

2.6.3 Aerial Re-orientation

We also tested the aerial reorientation capability of the gyrostat MPC controller. By locking the joints of the robot and solely relying on the torques from the RWAs, we dropped the robot from a height of 0.5 m with a 0.6 radian initial roll error, as shown in Figure 2-7. The RWAs were able to correct the robot's orientation in before touchdown. This experiment verifies that the RWAs are able to quickly correct large orientation errors in mid-air.

2.6.4 Balance-Beam Walking

To demonstrate the full capability of the gyrostat MPC controller on hardware, we perform a traversal of a narrow wooden beam, as shown in Figure 2-1 and the supplementary video. Since the support polygon is always close to a line, it is almost impossible for a standard quadruped to perform balancing and locomotion

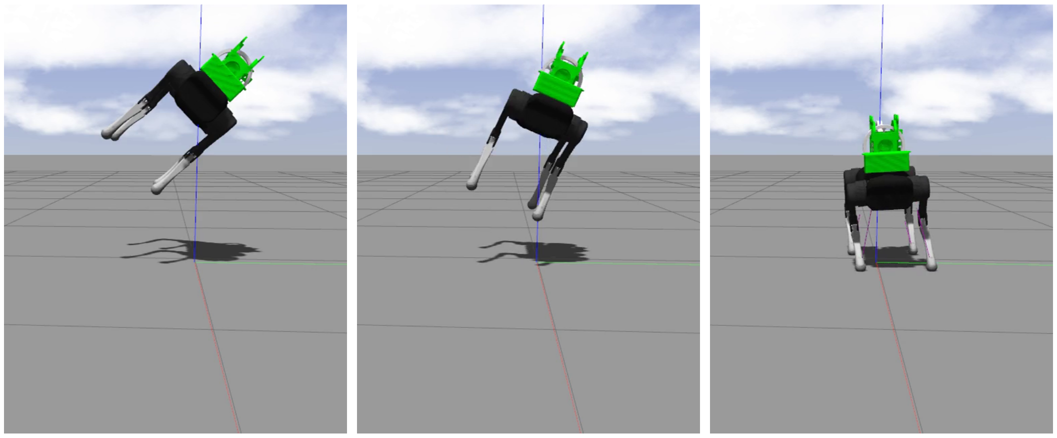


Figure 2-7: A drop test sequence where the robot reorients itself with the torques from the reaction wheels. The robot is initially positioned 50 cm off the ground with a 0.6 radian roll error. The RWA controller is set to turn on immediately after release to correct the attitude error.

simultaneously on the beam. In contrast, our robot with the RWA module and the gyrostat MPC controller is able to maintain a stable roll angle to keep itself from falling.

Hardware demonstration with the Unitree A1 poses a major challenge — the robot's deformable rubber feet cause unwanted vibration at higher walking frequency. To achieve stable beam walking, we lengthen the gait cycle from 1s to 3s. This significantly reduced the robot's walking speed and largely eliminated the vibration. We also fuse the robot's proprioceptive sensor data with an external motion-capture system in a Kalman Filter to achieve sub-centimeter position estimation so the robot can place foothold locations with sufficient accuracy to stay on the beam, which is less than 6 cm wide. Figure 2-8 shows the roll angle, roll RWA torque, and roll RWA velocity collected during the beam walking experiment. The RWA speed and torque are effectively kept under the threshold limits (200 rad/s and 5 Nm respectively) by the MPC constraints.

2.7 Discussion

We have presented a reaction-wheel actuator system that enhances quadrupedal robots' balancing and stabilization abilities during challenging locomotion tasks. We have shown that RWAs can be integrated into a state-of-the-art MPC framework with a few relatively minor modifications, and we have demonstrated our proposed gyrostat MPC controller in a series of hardware and simulation experiments. Our simulated experiments demonstrate that the RWA module helps the quadruped handle larger disturbances and gives it self-righting capabilities in mid-air. On hardware, we have successfully demonstrated the first narrow-beam walking performed by a quadruped. We believe that RWA modules like ours can be better optimized for lower power consumption and weight, and be integrated into many legged robot designs for improved robustness. The benefit of MPC

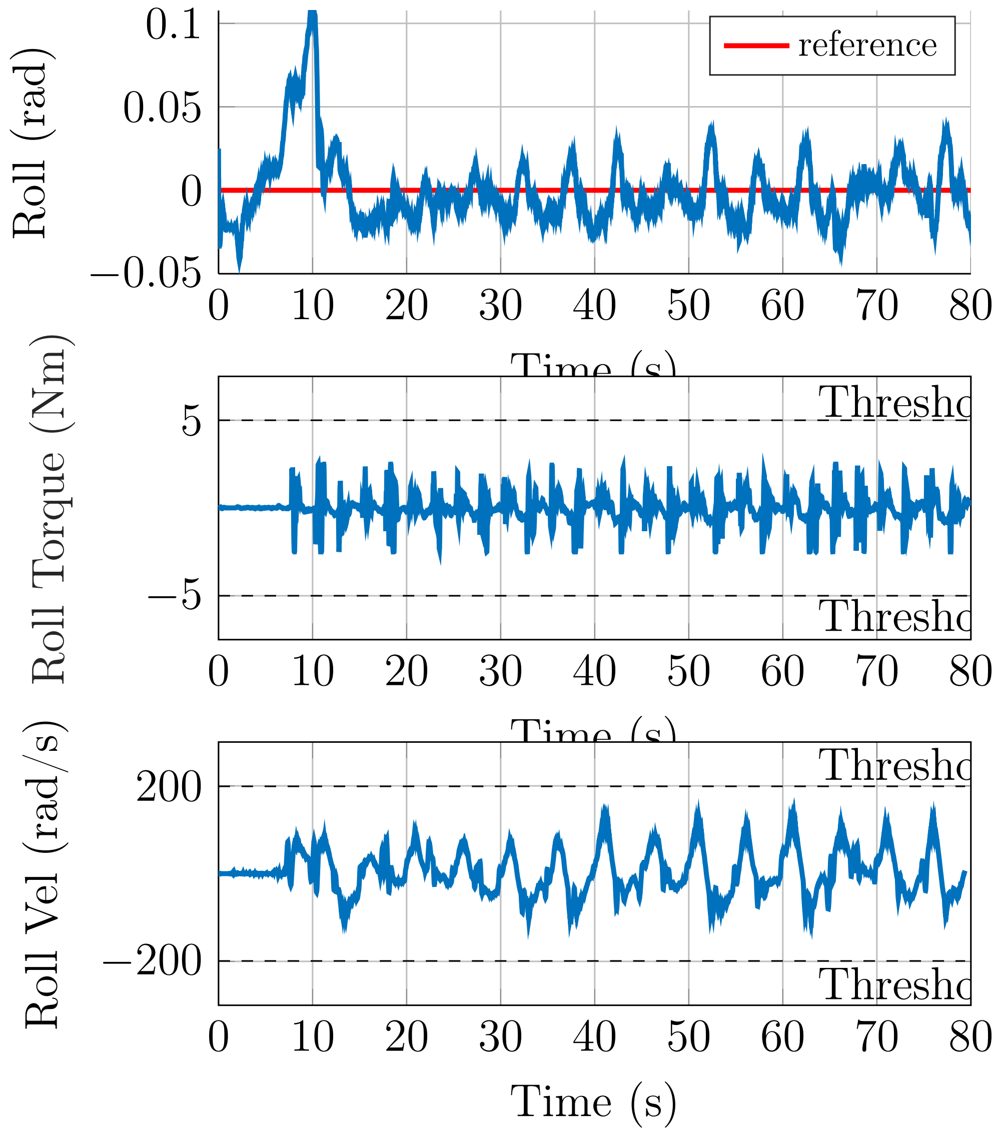


Figure 2-8: Roll angle, roll RWA torque, and roll RWA velocity during the hardware beam walking experiment.

Chapter 3

Proprioceptive Odometry and Online Kinematic Calibration

This chapter introduces a classical state estimation technology for legged robots: Kalman filter-based proprioceptive odometry. We then describe an online method to calibrate certain kinematic parameters that can be difficult to measure offline due to dynamic deformation effects and rolling contacts. A kinematic model of the robot's legs that depends on these parameters is used, along with measurements from joint encoders, foot contact sensors, and an inertial measurement unit (IMU), to predict the robot's body velocity. This predicted velocity is then compared to another velocity measurement from, for example, a camera or motion capture system. The difference between them is used to compute an update on the kinematic parameters. The method can be incorporated into any Kalman filter observation model involving leg odometry. We provide a theoretical observability analysis of our method, as well as validation both in simulation and on hardware. Hardware experiments demonstrate that online kinematic calibration can significantly reduce position drift when relying on odometry.

3.1 Introduction

Control and state estimation algorithms for legged robots depend critically on leg kinematic parameters. During locomotion, a planner calculates foot positions to generate collision-free foot-swing trajectories [121], and an estimator computes foot velocities to estimate the robot body's velocity [14, 19]. Foot positions and velocities are calculated using the forward kinematics [14, 103], which depend on kinematic parameters such as leg link lengths and motor offset distances.

On legged robots, the Leg Odometry (LO) [24] is commonly used to estimate robot body velocity. Assuming non-slipping contact, the joint angle velocity measurements of a leg can be mapped into a body velocity estimation by the Jacobian of the forward kinematics. Inaccurate leg-length knowledge can lead to velocity estimation errors, as shown in Fig. 3-1b, which prevent the robot state estimator from getting correct odometry information.

Existing legged robot controllers usually use fixed leg-length values obtained from a 3D model of the robot or manual measurement [14, 59, 158]. However, actual kinematic parameters are often not precisely known due to manufacturing variations, wear over time, rolling contacts, and dynamic deformation during normal operation (see Fig. 3-1a). Taking the Unitree A1 robot's kinematic structure as an example [148], its deformable foot makes the calf link length vary between 0.19-0.22 m. When the robot moves at 2.0m/s, joint angle velocity can reach 20rad/s, then a 0.01m error in link length leads to 0.2m/s velocity estimation error. If we integrate this poor velocity estimate to get a position estimate, position drift can grow by 0.2m every second. Moreover, it is possible for the leg to experience sudden length changes due to external impacts or damage. Although the controller may be robust enough to maintain balance [32], knowledge of the new length can greatly improve stability and reduce foot slip. Therefore, online calibration of kinematic parameters can be hugely beneficial during robot operation.

We propose a method to enable legged robots to calibrate unknown kinematic parameters online. We design a velocity measurement model that compares the LO velocity with accurate body velocity information from an external motion-capture system or visual odometry [86], and use the difference between these measurements to update the kinematic parameters. This can be integrated into many standard state-estimation algorithms so that kinematic parameters can be estimated in real time together with the robot's state.

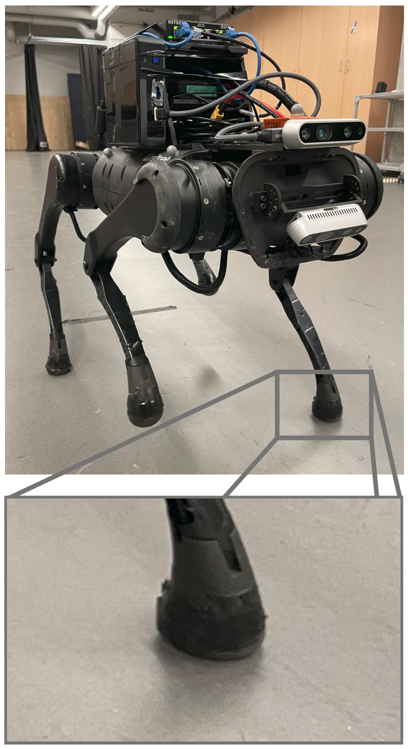
We present experiments with a Kalman filter that demonstrate position drift can be reduced by up to an order of magnitude through online calibration. We also show doing online calibration in an optimization-based sliding-window state estimator [118] is possible.

The chapter proceeds as follows: In Section 3.2 we survey related work in kinematics calibration and legged robot state estimation. Section 3.3 reviews Kalman filtering and forward kinematics. In Section 3.4 we define the calibration problem, derive our solution, and provide an observability analysis. The results of simulation and hardware experiments are presented in Section 3.5. Finally, Section 3.6 concludes the chapter.

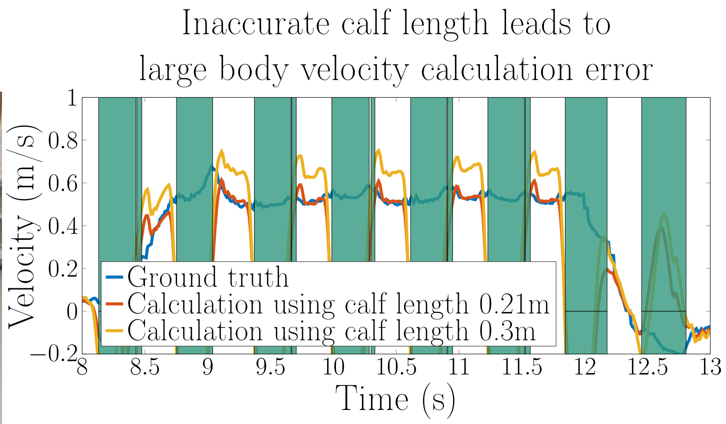
Our contributions include:

- A measurement model to calibrate unknown kinematic parameters of legged robots online and reduce odometry drift.

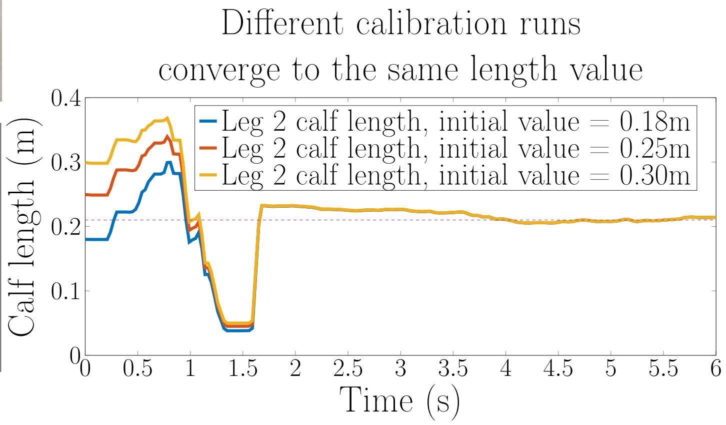
- A theoretical observability analysis to examine which kinematic parameters are



(a)



(b)



(c)

Figure 3-1: **(a)** Foot compression during locomotion changes calf length. **(b)** Body velocity inferred from kinematics of the Leg 2 (front-left) with wrong calf length (yellow) has larger error comparing to that using calibrated length (red). we enlarge possible leg length error to make velocity error more perceptible. The shaded green regions are periods when the leg not contacts the ground, during which the LO velocity is meaningless. **(c)** Our method calibrates calf length of the Leg 2 to around 0.21m (dash line) whatever the initial length value is.

observable.

- Integration of the measurement model into two different state estimators.
- Experimental validation on a Unitree A1 robot.

3.2 Related Work

We first review legged robot state estimation, with a particular focus on using proprioceptive odometry to obtain robot body position, velocity, orientation, and rotation rate for control purposes. Following this, we review works on kinematic calibration.

3.2.1 Legged Robot State Estimation

Research on onboard real-time state (including mainly pose and velocity) estimation for legged robots became popular in the recent decade [14, 19, 56, 75, 125, 158, 165]. With the increasing commercial availability of low-cost quadrupedal [39, 148] and humanoid robots, there is a strong need for cost-effective and reliable sensing solutions for such robots with limited computation resources.

Leg Odometry (LO) is the earliest state estimation method for humanoid and hexapod robots [70, 90]. The robot pose can be calculated using the leg kinematics from feet that are in stable contact with the ground. Repeated dead reckoning using stance foot allows position trajectory to be estimated [124]. However, LO itself is prone to errors such as foot slippage and joint encoder noise. [123, 134] combines IMU and LO using Extended Kalman Filters (EKF) to improve accuracy and robustness. An observability analysis presented in [19] shows that on a 12DOF quadruped robot, the body pose, velocity, and IMU biases can be recovered using one body IMU, joint encoders, and foot contact sensors stably. This EKF formulation is also applicable to humanoid robots [125]. A similar linear KF formulation is proposed in [14]. The invariant EKF is proposed in [53] to improve orientation estimation convergence. Since all sensors are proprioceptive, in other words, contained within the body of the robot, these methods all belong to Proprioceptive Odometry (PO). Moreover, although many variations on the EKF have been proposed, they all use the same basic types and number of sensors. The inertia information aids contact sensing [17, 23] to ensure LO is only used while feet are in contact. Some algorithms estimate contacts using kinematic information [61], eliminating the dependency on foot contact sensors. In PO methods, velocity estimations are typically good enough for stable closed-loop control, but position drifts are often as high as 10%-15% [19, 53, 75, 159]. The reason is, in addition to foot slippages, a legged robot also often experiences link deformations, rolling ground contacts [163], and excessive impacts with the ground, all of which lead to either large sensor noise or incorrect or biased velocity estimation if sensor measurement models do not capture the actual contact behavior. It has been shown that addressing these issues properly in PO can improve position estimation accuracy [163]. Nevertheless, PO is widely used because not all legged robot control applications require high-accuracy position estimation.

3.2.2 Kinematics calibration

Kinematics calibration for robot manipulators has been studied for decades [126]. Standard methods use the error between the end-effector position output from the forward kinematics model and position measurement from an accurate sensor to update the model parameters. Early calibration approaches relied on laser trackers [106]. Today, "hand-eye" calibration with computer vision is accurate enough to calibrate manipulators and even humanoid robot arms [107].

Online parameter calibration during state estimation has also been investigated in multi-sensor fusion. When using different sensors together to estimate robot pose (position and orientation), rigorous observability proofs have demonstrated that the extended Kalman Filter (EKF) [73, 86] can estimate spatial transformations among sensors (extrinsics). The visual-inertial system (VINS) [119] calibrates not only IMU-camera extrinsics parameters, but also sensor time delay in a sliding-window (also called fix-lag smoother [101] or receding-horizon [104]) optimization based estimation scheme. This work shows that, in visual-inertial odometry, jointly estimating robot state, sensor bias, and sensor transformations can reduce long-term position estimation drift. The observability study of these parameters to guarantee that they can be estimated in both EKF and optimization-based state estimators is done in [168].

Only a few existing works consider online kinematics-parameter calibration in legged robot state estimation, and the effects of parameter variations over time on estimator performance has not been well studied in the literature. Fusing IMU data and LO velocity in Kalman filters has been used on several different robots [14, 19, 24]. The invariant EKF was introduced in [53] to improve filter convergence. Optimization-based methods were proposed in [54, 158, 160]. The position drift due to leg odometry was studied in [159], where drift was compensated by a body velocity bias.

With the aid of special markers, “foot-eye” kinematics calibration on legged robots was demonstrated in [15]. The authors of [122] calibrated camera extrinsics in an optimization-based legged robot state estimator. A measurement model to allow the robot’s controller to adapt to dynamic model changes online was proposed in [140]. Dynamic deformation during bipedal locomotion was considered in [153], where the deformation is modeled as rotations among links. Estimating the deformation improves both position estimation and control. Finally, the kinematics calibration method proposed in [18] is similar to ours. However, it runs an expensive offline batch optimization and no observability analysis is provided.

3.3 Background

In this section we provide a detailed explanation to quaternion rotation representation and extended Kalman filtering. These two topics are fundamental to the rest of the thesis and extensively used in the algorithm implementation of this chapter, Chapter 4, Chapter 5, and Chapter 6.

3.3.1 Quaternion

A quaternion represents a spatial rotation in the following form

$$q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k} \tag{3.1}$$

where q_x, q_y, q_z, and q_w are real numbers. $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are basis elements. The Hamilton convention for basis element operations is

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \tag{3.2}$$

We often quaternion as

$$\boldsymbol{q} = \begin{bmatrix} q_x \\ q_y \\ q_z \\ q_w \end{bmatrix} \tag{3.3}$$

where q_w is the scalar part and $\mathbf{q}_v = [q_x; q_y; q_z]$ is the vector part. So it can also be written as

$$q = \begin{bmatrix} \mathbf{q}_v \\ q_w \end{bmatrix} \tag{3.4}$$

A unit quaternion is a quaternion with unit norm $\|\boldsymbol{q}\| = 1$. It can always be written as

$$\boldsymbol{q} = \begin{bmatrix} \mathbf{u} \sin \frac{\phi}{\mathbf{2}} \\ \cos \frac{\phi}{2} \end{bmatrix} \tag{3.5}$$

The vector $\mathbf{u}$ is a rotation axis and ϕ is a rotational angle.

3.3.2 Quaternion Multiplicative Map

The quaternion multiplication is shown to be [135]

$$\boldsymbol{p} \otimes \boldsymbol{q} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ p_w \end{bmatrix} \otimes \begin{bmatrix} q_x \\ q_y \\ q_z \\ q_w \end{bmatrix} = \begin{bmatrix} p_w q_x + p_x q_w + p_y q_z - p_z q_y \\ p_w q_y - p_x q_z + p_y q_w + p_z q_x \\ p_w q_z + p_x q_y - p_y q_x + p_z q_w \\ p_w q_w - p_x q_x - p_y q_y - p_z q_z \end{bmatrix} \tag{3.6}$$

This allows us to write the quaternion multiplication into matrix multiplication form

$$\boldsymbol{p} \otimes \boldsymbol{q} = \begin{bmatrix} p_w & -p_z & q_y & p_x \\ p_z & p_w & -p_x & p_y \\ -p_y & p_x & p_w & p_z \\ -p_x & -p_y & -p_z & p_w \end{bmatrix} \begin{bmatrix} q_x \\ q_y \\ q_z \\ q_w \end{bmatrix} = \begin{bmatrix} q_w & q_z & -q_y & q_x \\ -q_z & q_w & q_x & q_y \\ q_y & -q_x & q_w & q_z \\ -q_x & -q_y & -q_z & q_w \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ p_w \end{bmatrix} \tag{3.7}$$

By inspection we can write these matrices as

$$\mathcal{L}(\boldsymbol{p}) = \begin{bmatrix} p_w & -p_z & q_y & p_x \\ p_z & p_w & -p_x & p_y \\ -p_y & p_x & p_w & p_z \\ -p_x & -p_y & -p_z & p_w \end{bmatrix} = \begin{bmatrix} p_w\mathbf{I} + \lfloor \mathbf{p}_v \rfloor^{\times} & \mathbf{p}_v \\ -\mathbf{p}_v^T & p_w \end{bmatrix} \tag{3.8}$$

and

$$\mathcal{R}(\boldsymbol{q}) = \begin{bmatrix} q_w & q_z & -q_y & q_x \\ -q_z & q_w & q_x & q_y \\ q_y & -q_x & q_w & q_z \\ -q_x & -q_y & -q_z & q_w \end{bmatrix} = \begin{bmatrix} q_w\mathbf{I} - \lfloor \mathbf{p}_v \rfloor^{\times} & \mathbf{q}_v \\ -\mathbf{q}_v^T & q_w \end{bmatrix} \tag{3.9}$$

where $\lfloor * \rfloor^{\times}$ is defined as (2.1). Therefore

$$\boldsymbol{p} \otimes \boldsymbol{q} = \mathcal{L}(\boldsymbol{p})\boldsymbol{q} = \mathcal{R}(\boldsymbol{q})\boldsymbol{p} \tag{3.10}$$

3.3.3 Quaternion & Rotation Matrix

We introduce a matrix $B = \begin{bmatrix} I_{3\times 3} \\ 0 \end{bmatrix}$ that converts a vector in $\mathbb{R}^3$ to a quaternion with zero scalar part. We denote rotation matrix $R(\boldsymbol{q}) \in SO(3)$ as a function of $\boldsymbol{q}$, which is in the form of

$$R(\boldsymbol{q}) = B^T \mathcal{L}(\boldsymbol{q}) \mathcal{R}(\boldsymbol{q})^T B. \tag{3.11}$$

3.3.4 Quaternion Exponential & Logarithm Maps

In both robot control and estimation, two critical tools are the exponential map (exp map) and the logarithmic map (log map) of rotations. These tools are necessary to properly define derivatives and perturbations of rotations. To understand it well we recommend reading [135].

Any $SO(3)$ element can be viewed as a rotation angle ϕ around an unit axis $\boldsymbol{u}$. (Actually, according to the Chasles' theorem, any spatial displacement of a rigid body can be defined by a rotation about an axis and a translation along the same axis.) This angle-axis parameterization is commonly used for explaining and visualizing a rotation. It can be written as a quaternion

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{u}\sin(\frac{\phi}{2}) \\ \cos(\frac{\phi}{2}) \end{bmatrix}$$

This half angle comes from the fact that quaternion has a "double cover" property [135]. Let's call vector $\boldsymbol{\theta} = \phi\boldsymbol{u}$ a 3-parameter representation of the rotation, and define

$$\mathrm{Exp}(\boldsymbol{\theta}) = \begin{bmatrix} \boldsymbol{u}\sin(\frac{\phi}{2}) \\ \cos(\frac{\phi}{2}) \end{bmatrix} \tag{3.12}$$

as the quaternion exponential map, which converts a 3-parameter representation to a quaternion.

The logarithm map is essentially the inverse of the exponential map. It converts a quaternion to a 3-parameter representation as angle-axis parameterization. From (3.12), it can be seen that we can get angle and axis easily and define the logarithm map as

$$\mathrm{Log}(\boldsymbol{q}) = \boldsymbol{\theta} = \phi\boldsymbol{u} = 2\arctan(\|q_v\|, q_w) \cdot q_v/\|q_v\| \tag{3.13}$$

where $\phi = 2\arctan(\|q_v\|, q_w)$ is the rotation angle, and $\boldsymbol{u} = q_v/\|q_v\|$ is the rotation axis.

3.3.5 Small Angle Exp & Lop Maps

In practice we need some approximation to the exp and the log map. For one thing, we don't want to use (3.13) when the rotation is small - if $\|q_v\|$ approaches 0, it will cause numerical problems. However, small rotation is ubiquitous in rotation related state estimation and optimal control. For another, we often need the following Jacobian matrix

$$\frac{\partial \mathrm{Log}(\boldsymbol{q} \otimes \mathrm{Exp}(\boldsymbol{\theta}))}{\partial \boldsymbol{\theta}}. \tag{3.14}$$

The term

$$\boldsymbol{q} \otimes \mathrm{Exp}(\boldsymbol{\theta}) \tag{3.15}$$

is called small angle update. We will explain the significance of this matrix in the next section. With arctan and $\|\boldsymbol{q}_v\|$ involved, it is really hard to write down the jacobian analytically.

In this work, we use approximations of the exponential and logarithm maps common in spacecraft attitude estimation [84] and visual-inertial odometry [118]:

$$\mathrm{Exp}(\boldsymbol{\theta}) \approx \begin{bmatrix} 1 \\ \frac{\phi}{2}\mathbf{u} \end{bmatrix}, \tag{3.16}$$

$$\mathrm{Log}(\boldsymbol{q}) = \phi\mathbf{u} \approx 2\boldsymbol{q}_v. \tag{3.17}$$

We also make use of a similar first-order approximation of the matrix exponential [51]:

$$\mathrm{Exp}(\boldsymbol{\theta}) \approx (I + \lfloor \boldsymbol{\theta} \rfloor^{\times}). \tag{3.18}$$

The result quaternion in (3.16) is not a unit quaternion so a manual normalization can be added after performing small angle update $q \otimes \text{Exp}(\theta)$. With this simplication, the jacobian matrix 3.14 is also got simplified as

$$\frac{\partial \text{Log}(\boldsymbol{q} \otimes \text{Exp}(\boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} = \begin{bmatrix} q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \end{bmatrix} = q_w \mathbf{I} + \lfloor \boldsymbol{q}_v \rfloor^{\times} \tag{3.19}$$

where $\boldsymbol{q}_v = [q_x; q_y; q_z]$ and q_w are elements of $\boldsymbol{q}$. This is the top-left 3×3 block of $\mathcal{L}(\boldsymbol{q})$ as defined in 3.8. We can write $\mathcal{L}(\boldsymbol{q})_{3\times3}$ to denote it.

In literature, there are other approximations to the exp and log map. For example, the Rodrigues parameters and Cayley map [7] is used in LQR based trajectory optimization problems [65], which is defined as

$$\delta\boldsymbol{q} = \text{Exp}_{cayley}(\delta\boldsymbol{\theta}) = \frac{1}{\sqrt{1 + \|\delta\boldsymbol{\theta}\|^2}} \begin{bmatrix} 1 \\ \delta\boldsymbol{\theta} \end{bmatrix}. \tag{3.20}$$

And the inverse Cayley map [65]

$$\text{Log}_{cayley}(\boldsymbol{q}) = \boldsymbol{q}_v / q_s$$

converts small angle into the Rodrigues parameters $\delta\boldsymbol{\theta}$.

3.3.6 Quaternion Kinematics

The Exp Map allows for defining derivatives of quaternions which is critical for IMU kinematics.

If a rigid body has an orientation $\boldsymbol{q}$ and it rotates at an body frame angular velocity $\boldsymbol{\omega}$. The changing rate of the quaternion coefficients is

$$\dot{\boldsymbol{q}} = \lim_{\Delta t \to 0} \frac{\boldsymbol{q} \otimes \operatorname{Exp}(\boldsymbol{\omega}\Delta t) - \boldsymbol{q}}{\Delta t} = \lim_{\Delta t \to 0} \frac{\boldsymbol{q} \otimes \begin{bmatrix} \frac{1}{2}\boldsymbol{\omega}\Delta t \\ 1 \end{bmatrix} - \boldsymbol{q} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix}}{\Delta t} = \frac{1}{2}\boldsymbol{q} \otimes \begin{bmatrix} \boldsymbol{\omega} \\ 0 \end{bmatrix} \tag{3.21}$$

3.3.7 Rotation Matrix Exp & Log Map

Similiar in the quaternion case, a rotation angle ϕ and an rotation axis $\boldsymbol{u}$ can be writen as a rotation matrix as well. This is the rotation matrix exponential map

$$\operatorname{Exp}(\phi\boldsymbol{u}) = e^{\phi\lfloor \boldsymbol{u} \rfloor^{\times}} = I + \phi\lfloor \boldsymbol{u} \rfloor^{\times} + \frac{1}{2}\phi^2(\lfloor \boldsymbol{u} \rfloor^{\times})^2 + \frac{1}{3}\phi^3(\lfloor \boldsymbol{u} \rfloor^{\times})^3 + \cdots, \tag{3.22}$$

The Rodrigues rotation formula proves that

$$\operatorname{Exp}(\phi\boldsymbol{u}) = I + \sin\phi\lfloor \boldsymbol{u} \rfloor^{\times} + (1 - \cos\phi)(\lfloor \boldsymbol{u} \rfloor^{\times})^2 \tag{3.23}$$

Also in practice this exponential map can be simplified by approximiation. We truncate the Taylor series in 3.22, then let

$$\operatorname{Exp}(\phi\boldsymbol{u}) \approx I + \phi\lfloor \boldsymbol{u} \rfloor^{\times} \tag{3.24}$$

or we just write

$$\operatorname{Exp}(\boldsymbol{\omega}) \approx I + \lfloor \boldsymbol{\omega} \rfloor^{\times} \tag{3.25}$$

for any $\boldsymbol{\omega} \in \mathbb{R}^3$ with small norm.

The rotation matrix exponential map (especially the simplified version) is sometimes prefered during rotation related derivation because it is a 3-by-3 matrix that performs vector rotation by direct multiplcation. Also it is a bit cleaner than quaternion exponential map because the half angle in quaternion can be confusing sometimes.

On the other hand, the rotation logarithm map is not easy to work with. Given a rotation matrix R, its logarithm map $\mathrm{Log}(R) = \phi\boldsymbol{u}$ recovers the rotation angle and axis as

$$\phi = \arccos(\frac{trace(R) - 1}{2}) \tag{3.26}$$

$$\boldsymbol{u} = \frac{[R - R^T]^\vee}{2\sin\phi} \tag{3.27}$$

where $[]^\vee$ is the inverse of Eqn. (2.1) This map is undefined when R is identity so it potentially has numerical problems for small angles.

3.3.8 Rotation Kinematics

If a rigid body has an orientation R and it rotates at an body frame angular velocity $\boldsymbol{\omega}$. The changing rate of the rotation matrix coefficients is

$$\dot{R} = \lim_{\Delta t \to 0} \frac{R \otimes \mathrm{Exp}(\boldsymbol{\omega}\Delta t) - R}{\Delta t} = \lim_{\Delta t \to 0} \frac{R(I + \lfloor \boldsymbol{\omega}\Delta t \rfloor^\times) - R}{\Delta t} = R\lfloor \boldsymbol{\omega} \rfloor^\times \tag{3.28}$$

3.3.9 IMU-driven Error-state Kalman Filter

One of the standard approaches to estimate a robot's pose and velocity is the extended Kalman Filter (EKF) [135]. When working with EKF, we often first formulate the continuous time process model

$$\dot{x} = f(x) + w, \tag{3.29}$$

where f is a state evolution function and w is the noise. In reality, the true state x_t cannot be directly measured, nor does the noise. Our best knowledge is

$$\dot{\hat{x}} = f(\hat{x}). \tag{3.30}$$

Compute the Taylor expansion around $x = \hat{x} + \delta x$, then we can easily see

$$\delta \dot{x} = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{t-1}} \delta x + w. \tag{3.31}$$

This model is the error dynamic process model which describes how the error of our estimation evolves. In practice we need its discrete time version because sensor data arrives periodically. Although precise discretization is complicated, a practical simplication for two consecutive time instances $t - 1$ and t is

$$\delta x_t = F|_{\hat{x}_{t-1}} \delta x_{t-1} + w = (I + \Delta t \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{t-1}}) \delta x_{t-1} + w \tag{3.32}$$

At each time instance t we also get some sensor data z_t, we know it is an observation of the underlying true state

$$z_t = h(x_t) + v, \tag{3.33}$$

where h is a measurement model, and v is the measurement noise. Also, using the same measurement model, we can calcuate an expected output given our current state estimation

$$y_t = h(\hat{x}_t). \tag{3.34}$$

The two equations give us

$$z_t - y_t = z_t - h(\hat{x}_t) = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_t} \delta x_t + v \tag{3.35}$$

Eqn (3.32) and eqn (5.3) allows us to formulate the error state Kalman Filter (ESKF). Denote "innovation" $s_t(x) = z_t - h(x)$, $F = I + \Delta t \frac{\partial f}{\partial x}$, $H = \frac{\partial h}{\partial x}$, covariance of w as Q, and covariance of v as R. With ignoring some technical caveats, we define the ESKF as follows:

Algorithm 1: Error State Kalman Filter

```
procedure
    $\hat{x}_{t-1} = x_0$                                      ▷ Initial State
    $\hat{P}_{t-1} = P_0$                                      ▷ Initial Covariance
    $\delta x_{t-1} = 0$                                       ▷ Assume $\hat{x}_{t-1} = x_{t-1}$
    $\delta x_{t|t-1} = F|_{\hat{x}_{t-1}} \delta x_{t-1}$     ▷ Can be neglected since always 0
    $P_{t|t-1} = F \hat{P}_{t-1} F^T + Q$                      ▷ Process Update
    $s_t = z_t - h(\hat{x}_{t-1})$                             ▷ Innovation evaluated at $\hat{x}_{t-1} + \delta x_{t|t-1}$
    $S_t = H P_{t|t-1} H^T + R$
    $K_t = P_{t|t-1} H^T S_k^{-1}$                             ▷ Kalman gain calculation
    $\hat{x}_t = \hat{x}_{t-1} + K_t s_t$                      ▷ Kalman update
    $\hat{P}_t = (I - K_t H) P_{t|t-1}$
end procedure
```

For legged robot state estimation, IMU is a central building block for estimating robot pose. Let $\boldsymbol{x}_k = [\boldsymbol{p}; \boldsymbol{q}; \boldsymbol{v}] \in \mathbb{R}^{10}$ be the true robot state at time step k, where $\boldsymbol{p} \in \mathbb{R}^3$ is the robot position in the world frame, $\boldsymbol{q}$ is the robot's orientation quaternion, and $\boldsymbol{v} \in \mathbb{R}^3$ is the linear velocity of the robot's body represented in the world frame. We also denote the estimate of the robot's state as $\hat{\boldsymbol{x}}_k = [\hat{\boldsymbol{p}}; \hat{\boldsymbol{q}}; \hat{\boldsymbol{v}}]$. State errors are parameterized as $\delta \boldsymbol{x}_k = [\delta \boldsymbol{p}; \delta \boldsymbol{\theta}; \delta \boldsymbol{v}] \in \mathbb{R}^9 = [\boldsymbol{p} - \hat{\boldsymbol{p}}; \mathrm{Log}(\hat{\boldsymbol{q}}^{-1} \otimes \boldsymbol{q}); \boldsymbol{v} - \hat{\boldsymbol{v}}]$.

Assuming the robot's body has an IMU that outputs bias-free linear acceleration $\boldsymbol{a} \in \mathbb{R}^3$ and angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$ at time k, The discrete error-state process dynamics $\delta\boldsymbol{x}_{k+1} = f(\delta\boldsymbol{x}_k, \boldsymbol{a}, \boldsymbol{\omega}, \boldsymbol{n})$ are [135]:

$$\delta\boldsymbol{p}_{k+1} = \delta\boldsymbol{p}_k + \delta\boldsymbol{v}_k\Delta t + \boldsymbol{n}_v \tag{3.36}$$

$$\delta\boldsymbol{\theta}_{k+1} = (I - \lfloor\boldsymbol{\omega}\Delta t\rfloor^{\times})\delta\boldsymbol{\theta}_k + \boldsymbol{n}_\omega \tag{3.37}$$

$$\delta\boldsymbol{v}_{k+1} = \delta\boldsymbol{v}_k - R(\boldsymbol{q})\lfloor\boldsymbol{a}\Delta t\rfloor^{\times}\delta\boldsymbol{\theta}_k + R(\boldsymbol{q})\boldsymbol{n}_a, \tag{3.38}$$

where Δt is the time between two IMU readings and $\boldsymbol{n} = [\boldsymbol{n}_v; \boldsymbol{n}_\omega; \boldsymbol{n}_a]$ contains random noise sampled from a Gaussian distribution. We refer readers to [135] for a detailed derivation of the process dynamics.

In addition to an IMU, other sensors may provide observations of the robot's state. For example, a motion-capture system or cameras can measure the robot's pose [86]. We use $\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{n}_r$ to denote such noisy sensor measurements, where $\boldsymbol{n}_r$ is assumed to be Gaussian noise.

As for the Kalman update, we use the following equations to convert estimated error state to the actual state. Special care is taken for the quaternion part, which leverages the exp map we explained in Section 3.3.4.

$$\hat{\boldsymbol{p}}_{k+1} \leftarrow \hat{\boldsymbol{p}}_k + \delta\hat{\boldsymbol{p}}, \tag{3.39}$$

$$\hat{\boldsymbol{q}}_{k+1} \leftarrow L(\hat{\boldsymbol{q}}_k)\mathrm{Exp}(\delta\boldsymbol{\theta}), \tag{3.40}$$

$$\hat{\boldsymbol{v}}_{k+1} \leftarrow \hat{\boldsymbol{v}}_k + \delta\hat{\boldsymbol{p}}. \tag{3.41}$$

3.3.10 Leg Odometry Velocity

Assuming the j'th foot is in contact with the ground and does not slip, g and J can be used to calculate the body velocity of the robot. Let $\boldsymbol{p}_f^w$ denote the foot position in the world frame (see Fig. 2-2); It is a function of the robot's body position $\boldsymbol{p}$ and joint angles $\boldsymbol{\phi}$:

$$\boldsymbol{p}_f^w = \boldsymbol{p} + R(\boldsymbol{q})\boldsymbol{p_f} = \boldsymbol{p} + R(\boldsymbol{q})g(\boldsymbol{\phi}). \tag{3.42}$$

Let the time derivative of $\boldsymbol{p}_f^w$ be $\boldsymbol{v}_f^w$. The no-slip assumption means $\boldsymbol{v}_f^w = 0$. Therefore, by differentiating (3.42), we have

$$0 = \boldsymbol{v}_f^w = \dot{\boldsymbol{p}}_f^w = \dot{\boldsymbol{p}} + R(\boldsymbol{q})\frac{d}{dt}g(\boldsymbol{\phi}) + \frac{d}{dt}R(\boldsymbol{q})g(\boldsymbol{\phi}). \tag{3.43}$$

It is shown in [103] that $\frac{d}{dt}R(\boldsymbol{q}) = R(\boldsymbol{q})\lfloor\boldsymbol{\omega}\rfloor^{\times}$, and we defined $\boldsymbol{v} = \dot{\boldsymbol{p}}$. Therefore from (3.43) we derive an expression for the body velocity in the world frame:

$$\boldsymbol{v} = -R(\boldsymbol{q})[J(\boldsymbol{\phi})\dot{\boldsymbol{\phi}} + \lfloor\boldsymbol{\omega}\rfloor^{\times}g(\boldsymbol{\phi})]. \tag{3.44}$$

This velocity is called the LO velocity because its integration is the body displacement[90]. However, the LO velocity is never used alone as an odometer because it is very noisy [19] due to foot contacts. In this work we treat the LO velocity as a measurement to estimate kinematic parameters.

3.3.11 Standard Single-IMU Proprioceptive Odometry

Let the robot's state be $\boldsymbol{x} = [\boldsymbol{p}; \boldsymbol{v}; \boldsymbol{\theta}; \boldsymbol{s}_1, \ldots, \boldsymbol{s}_j, \ldots, \boldsymbol{s}_L]$, where $\boldsymbol{p} \in \mathbb{R}^3$ is the robot position in the world frame, $\boldsymbol{\theta}$ is the robot's orientation Tait-Bryan angles, and $\boldsymbol{v} \in \mathbb{R}^3$ is the linear velocity of the robot's body represented in the world frame. For $j \in \{1, \ldots, L\}$ where L is the total number of legs of the robot, $\boldsymbol{s}_j$ is the foot position of leg j represented in the world frame. For clarity, we will only discuss the case when $L = 1$ in this section and drop the symbol j from subsequent equations. The robot's sensors generate a number of measurements including IMU linear acceleration $\boldsymbol{a}_b$, IMU angular velocity $\boldsymbol{\omega}_b$, joint angle $\boldsymbol{\phi}$, joint-angle velocity $\dot{\boldsymbol{\phi}}$, and c which is a binary contact flag with $c = 1$ indicating foot contact.

Standard PO uses the EKF to estimate the state from a single IMU and the LO velocity [14, 19]. The IMU is biased [127] and the LO velocity may also have a bias due to leg kinematic parameter changes [163]. We do not address these biases in this work, but they can be easily added to the EKF using well-known techniques to improve the overall estimation accuracy [97].

The process model of standard PO is based on IMU kinematics. A discrete-time dynamics update using Euler integration is presented in [14],

$$\hat{\boldsymbol{x}}_{k+1} = \begin{bmatrix} \hat{\boldsymbol{p}}_{k+1} \\ \hat{\boldsymbol{v}}_{k+1} \\ \hat{\boldsymbol{\theta}}_{k+1} \\ \hat{\boldsymbol{s}}_{k+1} \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{p}}_k + \Delta t \hat{\boldsymbol{v}}_k \\ \hat{\boldsymbol{v}}_k + \Delta t (R(\hat{\boldsymbol{\theta}}_k)\boldsymbol{a}_b - \boldsymbol{g}_w) \\ \hat{\boldsymbol{\theta}}_k + \Delta t (\Omega(\hat{\boldsymbol{\theta}}_k)\boldsymbol{\omega}_b) \\ \hat{\boldsymbol{s}}_k \end{bmatrix}, \tag{3.45}$$

where Δt is the time interval between k and $k+1$.

A common technique used in the literature is contact-based covariance scaling [14, 19]. Since we cannot update the foot position in the process model [19] during foot swing, for the term corresponding to $\hat{\boldsymbol{s}}$ in $\boldsymbol{n}_k$, we set its covariance σ_s to a large value if $c = 0$, and a small value otherwise:

$$\sigma_s = c\sigma_c + (1 - c)\sigma_n. \tag{3.46}$$

σ_c and $\sigma_n >> \sigma_c$ are all tunable hyperparameters. This technique works well in practice, but it will increase the condition number of the covariance matrix, thus reducing the stability of the filter [97].

We formulate the EKF measurement model following [14]. From sensor measurements, a vector $\bar{\boldsymbol{y}}_k$ is obtained as

$$\bar{\boldsymbol{y}}_k = \begin{bmatrix} g(\boldsymbol{\phi}) \\ -J(\boldsymbol{\phi})\dot{\boldsymbol{\phi}} + \lfloor \boldsymbol{\omega}_b \rfloor^{\times} g(\boldsymbol{\phi}) \end{bmatrix} \tag{3.47}$$

The measurement function $h(\hat{\boldsymbol{x}}_k)$ is defined as

$$h(\hat{\boldsymbol{x}}_k) = \begin{bmatrix} R(\hat{\boldsymbol{\theta}}_k)^T(\hat{\boldsymbol{s}}_k - \hat{\boldsymbol{p}}_k) \\ R(\hat{\boldsymbol{\theta}}_k)^T \hat{\boldsymbol{v}}_k \end{bmatrix} \tag{3.48}$$

The first term of the residual $\bar{\boldsymbol{y}}_k - h(\hat{\boldsymbol{x}}_k)$ indicates that the estimated body position and foot position must differ by a distance equal to the leg forward kinematics position. The second term ensures that the estimated robot body velocity matches the LO velocity (3.44), which we refer to as a “zero-velocity” observation model. Subsequently, we can utilize residuals from all legs in the EKF as shown in Algorithm 1. However, measurement residuals are only applicable for non-slipping standing feet. Therefore, the noise covariance Σ_w is adjusted based on the contact flag c as in equation (3.46) [19].

3.4 Technical Approach

The key idea underlying our technical approach to realize online calibration is to treat the LO velocity (3.44) as a measurement of the robot body's velocity that is dependent on a set of kinematic parameters $\boldsymbol{\rho}$ for each leg. We then append $\boldsymbol{\rho}$ to the filter state. The dimension of $\boldsymbol{\rho}$ depends on the kinematic structure of the leg. For example, if we are estimating calf leg lengths of a quadruped robot,

$\boldsymbol{\rho} = [l_{c1}; l_{c2}; l_{c3}; l_{c4}] \in \mathbb{R}^4$, where l_{ci} is the calf length of leg i. If both the calf and the thigh leg lengths need to be considered, then $\boldsymbol{\rho} = [\dots, l_{ci}; l_{ti}, \dots]$, for $i = 1, 2, 3, 4$, has dimension 8. We also write ρ_i to indicate the kinematic parameters related to leg i.

3.4.1 Body Velocity Measurement Model

We modify (3.44) to explicitly include kinematics parameters and sensor noise. For leg i, assuming the joint encoders on the leg measure joint angles $\boldsymbol{\phi}_i$ and angular velocities $\dot{\boldsymbol{\phi}}_i$ with additive Gaussian noise Gaussian $\boldsymbol{n}_\phi$ and $\boldsymbol{n}_{\dot{\phi}}$, repsectively, then the true LO velocity in the world frame is

$$\boldsymbol{v}_{m,i} = -R(\boldsymbol{q})[J(\boldsymbol{\phi}_i - \boldsymbol{n}_\phi, \boldsymbol{\rho}_i)(\dot{\boldsymbol{\phi}}_i - \boldsymbol{n}_{\dot{\phi}}) - \lfloor \boldsymbol{\omega} \rfloor^\times g(\boldsymbol{\phi}_i - \boldsymbol{n}_\phi, \boldsymbol{\rho}_i)]. \tag{3.49}$$

We also define an estimated LO velocity as

$$\hat{\boldsymbol{v}}_{m,i} = -A(\hat{\boldsymbol{q}})[J(\boldsymbol{\phi}_i, \hat{\boldsymbol{\rho}}_i)\dot{\boldsymbol{\phi}}_i - \lfloor \boldsymbol{\omega} \rfloor^\times g(\boldsymbol{\phi}_i, \hat{\boldsymbol{\rho}}_i)]. \tag{3.50}$$

When the leg i has non-slipping contact with the ground, it contributes to a measurement model

$$\boldsymbol{z}_{leg} = h_{leg}(\hat{\boldsymbol{x}}, \boldsymbol{\omega}, \boldsymbol{\phi}, \dot{\boldsymbol{\phi}}) + \boldsymbol{n}_c = \hat{\boldsymbol{v}} - \sum_i c_i \hat{\boldsymbol{v}}_{m,i} + \boldsymbol{n}_l(\boldsymbol{n}_\phi, \boldsymbol{n}_{\dot{\phi}}) + \boldsymbol{n}_c, \tag{3.51}$$

where $\boldsymbol{n}_l$ is a noise function related to joint encoder noise, which can be derived from linearizing (3.49). We assume each foot of the legged robot has a contact detector [19, 23] that generates a binary contact flag c_i. Therefore, $c_i = 1$ means the foot has nonzero velocity relative to the ground. Otherwise, the non-slipping assumption of

(3.50) is invalid. $\boldsymbol{n}_c$ is a Gaussian measurement noise whose variance is a tunable hyper-parameter.

3.4.2 Kalman Filter Kinematics Calibration

To achieve kinematics calibration using the error-state KF, we use a new process dynamics model and add the body velocity measurement (3.51) to the measurement model.

In the process dynamics, in addition to (3.36), (3.37), and (3.38), we model the evolution of $\delta\boldsymbol{\rho}$ as a random-walk process,

$$\delta\boldsymbol{\rho}_{k+1} = \delta\boldsymbol{\rho}_k + \boldsymbol{n}_\rho, \tag{3.52}$$

where $\boldsymbol{n}_\rho$ is a Gaussian white noise.

For the measurement model, we calculate a measurement vector as $\boldsymbol{z} = [\boldsymbol{z}_{mocap}; \boldsymbol{z}_{cam}; \boldsymbol{z}_{leg}]$, where $\boldsymbol{z}_{mocap}$ and $\boldsymbol{z}_{cam}$ are measurements from a motion-capture system or camera. $\boldsymbol{z}_{leg}$ is described in (3.51). The algorithm then proceeds as in Section 3.3.9.

3.4.3 Observability Analysis

Prior research has shown that robot pose and velocity are observable when the measurement model contains information from a motion-capture system or camera [73, 86]. Therefore, we only focus on the observability of the kinematics parameters $\boldsymbol{\rho}$ in this section. We also note that our observability analysis applies to both EKF and sliding-window estimators.

Neglecting sensor noise, the dynamics and observation model for a system with only leg measurements can be written as,

$$\begin{aligned} \dot{\boldsymbol{x}} &= f_c(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{\omega}) \\ \boldsymbol{z} &= h(\boldsymbol{x}, \boldsymbol{\phi}, \dot{\boldsymbol{\phi}}), \end{aligned} \tag{3.53}$$

where f_c is the continuous state process dynamics (closely related to the error-state dynamics (3.36), (3.37), and (3.38)); h is the measurement model (3.51) considering just a single leg; and $\boldsymbol{a}$, $\boldsymbol{\omega}$, $\boldsymbol{\phi}$, and $\dot{\boldsymbol{\phi}}$ are IMU acceleration, IMU angular velocity, joint angles, and joint angle velocities respectively. We then compute the observability Gramian [26, 80]

$$\mathcal{W}(\boldsymbol{x}_0) = \int_0^T \Phi^\top(t) H^\top(\boldsymbol{x}_t) H(\boldsymbol{x}_t) \Phi(t) \mathrm{d}t, \tag{3.54}$$

where $\Phi(t)$ is the state transition matrix associated with the linearized dynamics:

$$\dot{\Phi}(t) = F_x(\boldsymbol{x}_t)\Phi(t), \quad \Phi(0) = I. \tag{3.55}$$

If $\mathcal{W}(\boldsymbol{x}_0)$ is positive definite along the trajectory from $\boldsymbol{x}_0$ to $\boldsymbol{x}_T$, the system is locally observable [80].

For the Unitree A1 quadruped, its important kinematic parameters are indicated in Fig. 2-2. o_x, o_y are offsets distances between the robot COM and leg base. d is an offset between motor 2 and 3. l_t is the upper leg (thigh) length and l_c is the lower leg (calf) length. The analytical form of the forward kinematics function is provided in the Appendix. Among these parameters, we may choose to calibrate $\boldsymbol{\rho} = [l_c]$ (just the calf length) or $\boldsymbol{\rho} = [l_t; l_c]$ (both the calf and the thigh).

We analyze how parameter $\boldsymbol{\rho}$ is related to the observability gramian by expanding blocks in $\mathcal{W}(\boldsymbol{x}_0)$ analytically. From (3.36), (3.37), (3.38), and (3.52), we get

$$F_x = \begin{bmatrix} I & 0 & I\Delta t & 0 \\ 0 & I - \lfloor \boldsymbol{\omega}\Delta t \rfloor^\times & 0 & 0 \\ 0 & -R(\boldsymbol{q})\lfloor \boldsymbol{a}\Delta t \rfloor^\times & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix}. \tag{3.56}$$

From (3.56) and (3.55), $\Phi(t)$ is always in the form of

$$\Phi(t) = \begin{bmatrix} I & * & * & 0 \\ 0 & * & 0 & 0 \\ 0 & * & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix}, \tag{3.57}$$

where "$*$"s are nonzero block terms that are not relevant to our discussion. We linearize (3.51) to compute,

$$H_{leg} = \begin{bmatrix} \mathbf{0} & A(\hat{\boldsymbol{q}})\lfloor J\dot{\boldsymbol{\phi}} - \lfloor \boldsymbol{\omega} \rfloor^\times g \rfloor^\times & I & -A(\hat{\boldsymbol{q}})D \end{bmatrix}, \tag{3.58}$$

where,

$$D = (\dot{\boldsymbol{\phi}}^\top \otimes I_3)\frac{\partial vec(J(\boldsymbol{\phi}, \hat{\boldsymbol{\rho}}))}{\partial \hat{\boldsymbol{\rho}}} + \lfloor \boldsymbol{\omega} \rfloor^\times \frac{\partial g(\boldsymbol{\phi}, \hat{\boldsymbol{\rho}})}{\partial \hat{\boldsymbol{\rho}}}, \tag{3.59}$$

and the $vec(\cdot)$ operator returns a column vector by stacking the columns of the input matrix [94], and the $\otimes$ is the Kronecker product [94].

Plugging (3.57) and (3.58) into (3.54), we can see the last block of the integrand of $\mathcal{W}(\boldsymbol{x}_0)$ is $D^\top D$. Therefore, a sufficient condition for fully observable $\boldsymbol{\rho}$ is the null space of D is empty or, equivalently, that D has full column rank. An immediate conclusion we can draw is $\boldsymbol{\omega}$ and $\dot{\boldsymbol{\phi}}$ cannot both be zero, otherwise D will become a zero matrix. Therefore the robot cannot stand still or trot in place, since the joint velocities of stance legs will be close to zero. The rank condition of D also depends on the forward kinematics function, which is problem specific since $g(\boldsymbol{\phi}, \boldsymbol{\rho})$ may have different forms depending the robot leg structure. In the Appendix we show the forward kinematics of the Unitree A1 robot and how to calculate D. When $\boldsymbol{\rho} = [l_c]$ or $\boldsymbol{\rho} = [l_t; l_c]$, D has full rank so $\boldsymbol{\rho}$ is observable.

The observability analysis also suggests we trust the body velocity measurement model less when D is near singular. We change the covariance σ_c of the noise term $\boldsymbol{n}_c$ in (3.51) to be related to the mean of singular values of D.

$$\sigma_c = \sigma_0 + \frac{\alpha_1}{1 + \exp(\alpha_2(mean(SVD(D)) - \alpha_3))} \tag{3.60}$$

where σ_0 is a constant term, the second term is a logistic regressor [31] that assigns large value to D close to singular, which is equivalent to the local unobservablility index (LUI) proposed in [80]. Thus we call it LUI noise. This noise function can prevent parameter estimation fluctuation. We will study its effect in Section 3.5.

3.5 Experiments

All of our hardware and simulation experiments are based on a Unitree A1 [148] robot. We first verify that the algorithm is able to converge to unbiased parameter estimates in simulation. The simulated robot has the same leg structure as that on the A1 robot, but kinematic parameter values are varied for testing. We then perform hardware experiments on a real A1 to demonstrate the practical performance of the algorithm. MATLAB implementations of the error-state KF and the sensor data we

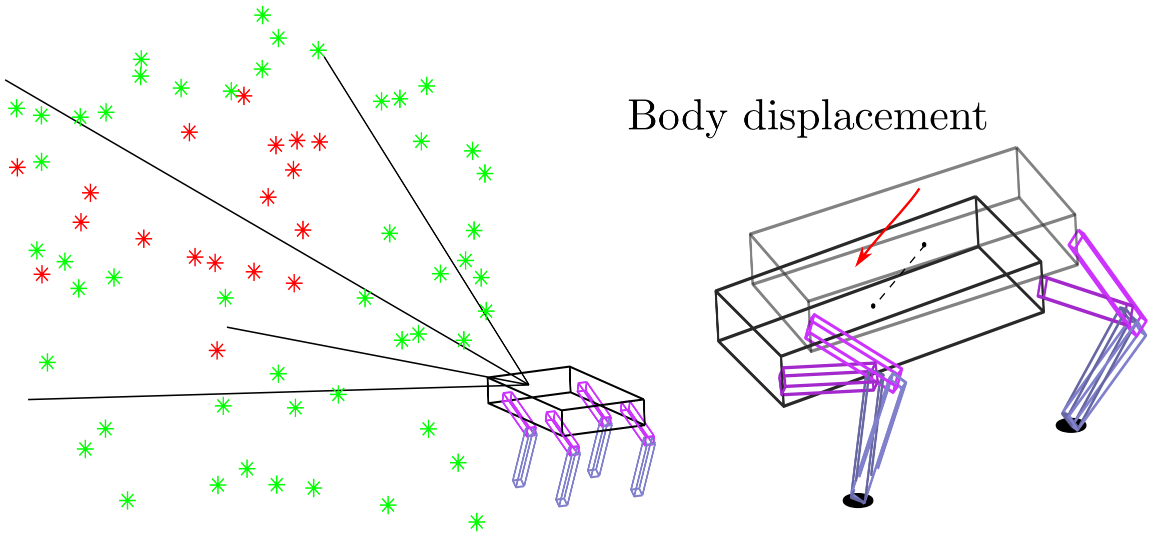


Figure 3-2: **Left:** The simulated robot and environment landmark locations. **Right:** In the simulation we focus on analyzing how state is estimated within one gait cycle, during which the body shifts a small distance, feet stand on the ground without moving, and joints change configuration accordingly.

collected are available on GitHub[1].

3.5.1 Simulation

We implement a simulator in MATLAB to generate simulated sensor data. We assume a periodic gait and known initial and final poses of the robot at the beginning and end of each gait cycle. We also assume perfect contact knowledge so that, during this gait cycle, contact feet have known fixed world positions. We use cubic Hermite splines, which are twice differentiable, to interpolate positions, and quaternion SLERP [131] to interpolate orientations. Therefore, we can query the robot's body position, orientation, velocity, and acceleration at any time in the gait cycle. From these quantities, we can calculate body-frame acceleration and angular velocities to

[1]`https://github.com/ShuoYangRobotics/legged-kinematics-calibration`

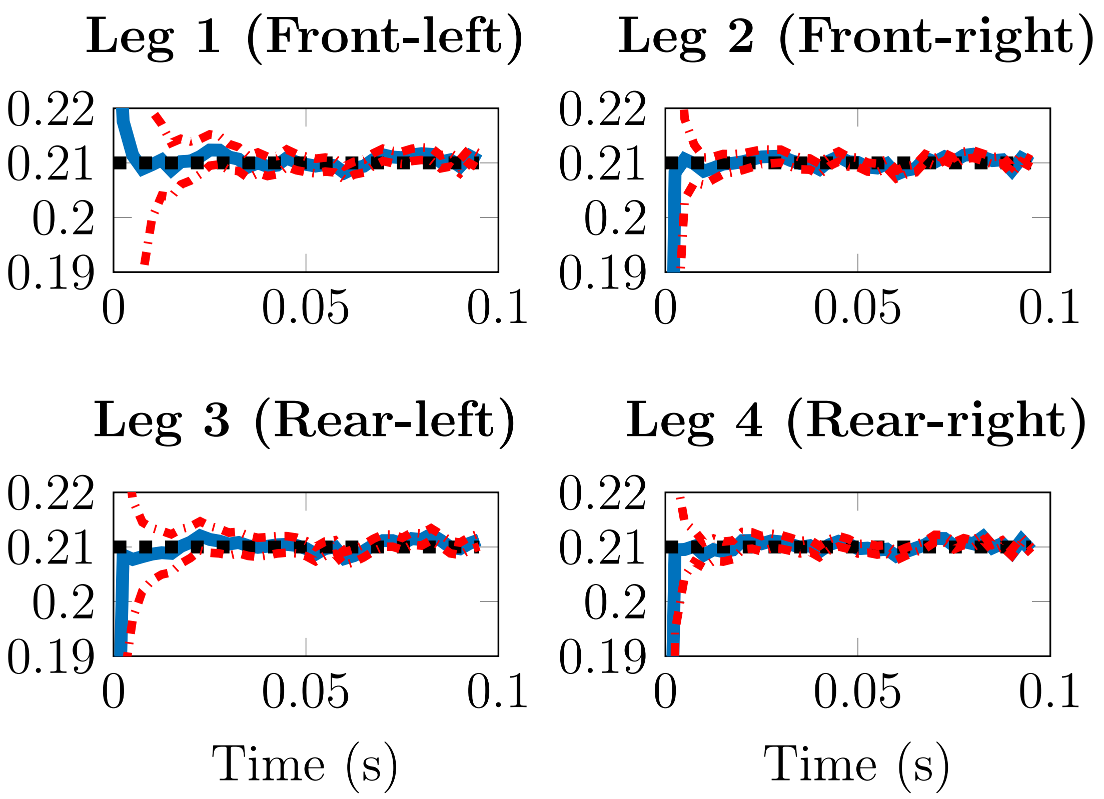


Figure 3-3: The calf length estimation result using simulation data. $\boldsymbol{\rho} = [l_c]$. In all plots, blue lines are estimated length. Red dash lines show the 3σ uncertainty envelope. Black dash lines indicate the ground truth length $0.21m$ for reference. All estimations converge to ground truth quickly with final errors $< 0.01m$.

generate simulated IMU data. Additionally, by using inverse kinematics, we can calculate the joint angles of a leg given a body position and a foot position. We also include a camera based on the pin-hole model [52] to observe landmarks with known locations in the environment. We generate camera observations by projecting landmarks onto the camera image frame [86]. Finally we add random noise to all simulated sensor data.

We run an error-state KF with the camera and the leg measurement models on the simulated data. Fig. 3-3 shows the calf length estimation result for a 0.1s body trajectory with a linear displacement of $(0.1m, 0.1m, 0.05m)$ and orientation displacement of 5 degrees about the pitch, roll, and yaw axes. The KF state includes robot body pose, velocity, and calf lengths l_c of all legs. Even if the initial l_c values have large errors, the filter converges to ground truth values quickly with final errors less than 0.01m. The detailed setup of the error-state KF can be seen in the open source codebase.

3.5.2 Error-state KF Hardware Experiment

We test our calibration method using sensor data from an actual A1 robot. Its sensors include one IMU, 12 joint encoders and 4 foot contact sensors. We implement a MPC controller in C++ as explained in 2.3.2. The robot moves in an arena equipped with an OptiTrack motion capture system, which provides high-quality body pose data. Robot sensor data and motion-capture data with timestamps are recorded as datasets. Although our filter can easily be run in real time, we perform all experiments offline so that we can replay the sensor datasets and run the filter with different settings. We refer interested readers to our open-source implementation for implementation details.

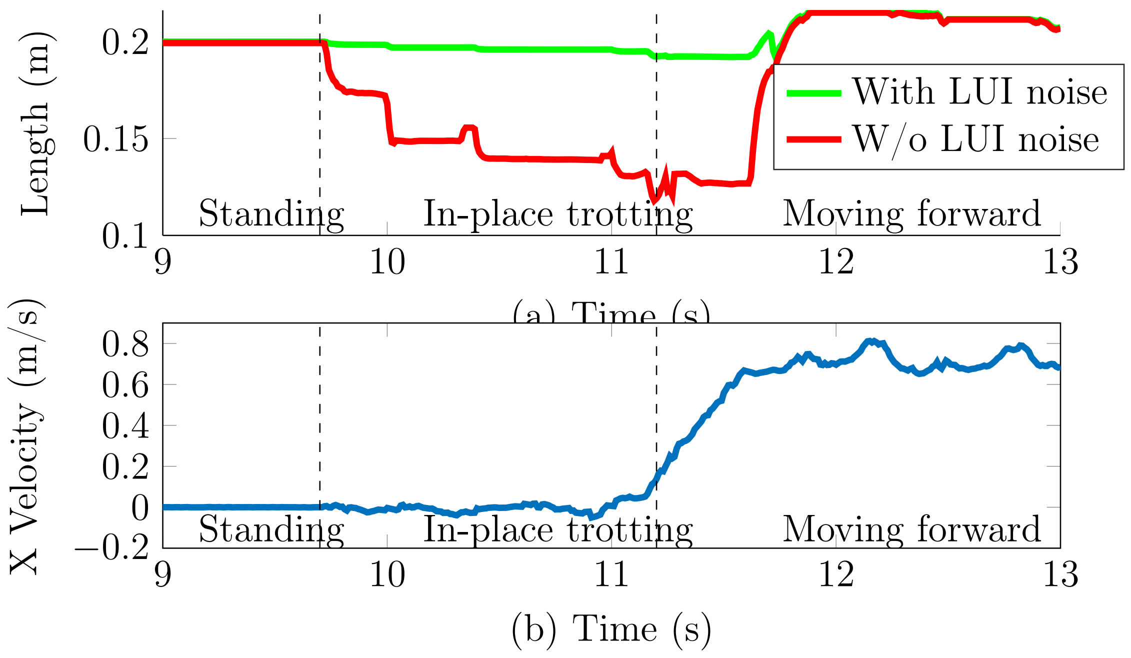
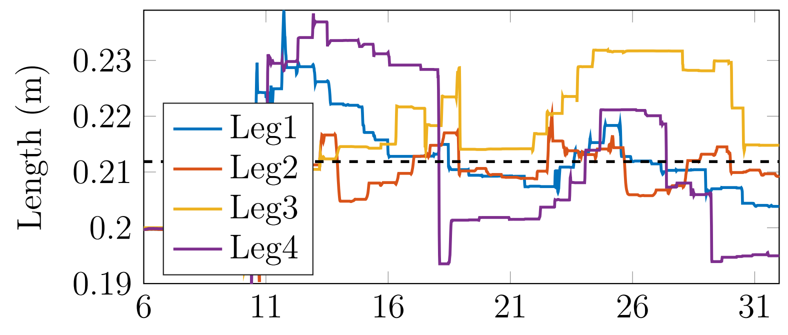
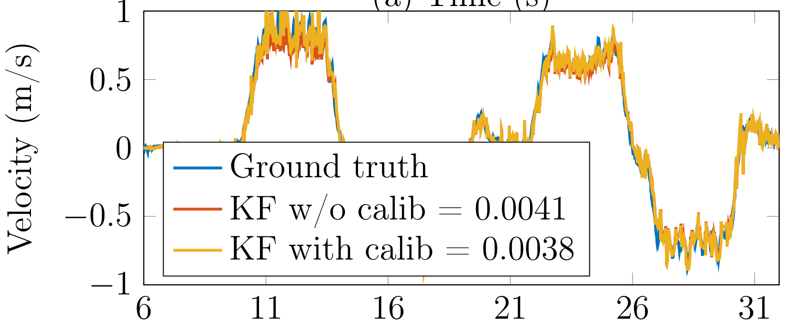


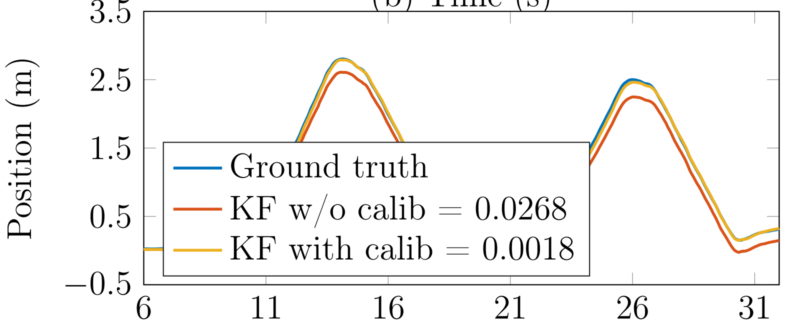
Figure 3-4: **(a)** The calf length estimation for leg 1 during two calibration runs. The green line comes from a filter that has the LUI noise term (3.60), while the red line is generated by a filter does not have the term. Both filters have the same other configurations. Black dash lines indicate time instances when the robot changes behavior modes. During in-place trotting the red line drifts significantly. **(b)** Velocity profile in each mode. The robot has small body velocity hence small joint angle velocities when trotting. According to (3.59), the observability matrix is very close to singular so the measurement update is inaccurate.



(a) Time (s)



(b) Time (s)



(c) Time (s)

Figure 3-5: Results of a hardware dataset run. **(a)** Calibrated calf lengths of each leg. The black dash line indicates the mean value of all lengths (0.2113m). **(b)** The velocity estimations using either fixed or calibrated length do not differ much. The mean square error (MSE) of them from the ground truth velocity are 0.0041 and 0.0038 respectively. **(c)** The KF with calibrated calf length has much smaller position drift than the KF using fixed calf length. The MSEs from the ground truth are 0.0018 and 0.0268 respectively.

Calibration During Standing Up

In the first experiment, we record data while the robot stands up from a crouched pose. All feet are always in contact with the ground. The process is repeated for four trials and four standing-up datasets are collected. We then run the error-state KF with kinematics calibration on each dataset three times with different initial values $l_c = 0.1m$, $0.2m$, and $0.3m$. In Fig. 3-1c, we compare estimated values of l_c against time for each run on one dataset. In all three runs, the final value reaches around 0.21m after about 3s. This calibrated value is roughly consistent with the CAD model value of 0.20m and the foot sensor head radius of 0.02m, and implies that the soft, deformable foot is compressed to half of its original size under the robot's weight.

Calibration During Walking

We move the robot on flat ground to examine how kinematics calibration performs during walking. We collect ten datasets with the robot moving at different speeds and different total travel distances ranging from 5m to 15m. The calibration results using one of the datasets is shown in Fig. 3-5a. Initially all leg calf lengths are set to 0.2m. After the robot starts to walk, the leg length estimation fluctuates between 0.19m-0.23m, the range is larger than the longest possible leg length in CAD model (0.22m). Comparing Fig. 3-5a, Fig. 3-5b, and the experiment videos, the maximum leg length happens when the robot moves forward with relatively high speed, and the robot feet have rolling contact with the ground. The rolling contact is equivalent to a slightly longer leg with fixed point contact.

LUI Noise Ablation Study

In Section 3.4.3 we present an LUI noise term (3.60). In Figure 3-4 we show that, without this term (red line), the calf-length estimation drifts quickly when the robot is doing in-place trotting, since the observability matrix is poorly conditioned. When the LUI noise term is included, the estimation does not change much, as expected.

Position Drift Reduction

We show that adding kinematics calibration to a baseline IMU and leg odometry filter can dramatically reduce position estimation drift. Our baseline filter follows Section 3.3.9, with the IMU driving the process model and leg odometry captured by the measurement model (3.51). We also treat body orientation, as observed by the motion-capture system, as a known quantity following [14]. The baseline filter always uses a fixed leg length of $0.20m$ (referred to as "KF w/o calib" in the figure legend). Fig. 3-5b and Fig. 3-5c compare the estimated X-direction velocity and position using either fixed length or calibrated length shown in Fig. 3-5a. It can be seen that the KF with calibrated leg length achieves an order of magnitude better precision than that using fixed length. Table 3.1 summarizes the mean-squared error in the position and velocity estimates, final position drifts, and maximum position drifts across the ten datasets. The kinematics calibration significantly improves position estimation accuracy in all cases by providing the estimator with time-varying kinematic parameters.

	w/o calib	with calib	Improvement
Vel MSE	0.0023	0.0022	4.3%
Pos MSE	0.0070	0.0016	77.1%
Max Pos Drift	15.0cm	6.2cm	58.6%
Final Pos Drift	7.1cm	4.1cm	42.3%

Table 3.1: The table shows the average peformance metrics of ten datasets and the improvement of using calibrated length. Max pos drift is the maximum deviation of estimated position from the ground truth. Final pos drift is the position deviation at the end of the traveling trajectory.

3.6 Conclusion & Future Work

We have presented a method to calibrate kinematic parameters of legged robots online. A detailed observability analysis, along with simulation and hardware experiments, validate our method. Kinematics calibration of deformable leg lengths results in more accurate body velocity estimation and, hence, significantly lower odometry drift. The calibration method can be easily integrated into standard state estimator formulations.

In future work, we will investigate kinematic parameter formulations that can better capture rolling contacts. We will also research whether jointly estimating robot states and kinematics parameters can achieve sub-centimeter calibration accuracy and reduce long term position estimation drift using the optimization based state estimator.

3.7 Appendix

The forward kinematics function g of a leg of a Unitree A1 robot with $\boldsymbol{\phi} = [\phi_1; \phi_2; \phi_3]$ and $\boldsymbol{\rho} = [l_c]$ is

$$g(\boldsymbol{\phi}, \boldsymbol{\rho}) = \begin{bmatrix} o_x - l_c s_{23} - l_t s_2 \\ o_y + d c_1 + l_t c_2 s_1 + l_c s_1 c_{23} \\ d s_1 - l_t c_1 c_2 - l_c c_1 c_{23} \end{bmatrix}, \tag{3.61}$$

where s_i denotes $\sin(\phi_i)$ and $c_i = \cos(\phi_i)$, where $i = 1, 2$. Also $s_{23} = \sin(\phi_2 + \phi_3)$ and $c_{23} = \cos(\phi_2 + \phi_3)$. The expression is derived using the product of exponentials (POE) method [103]. The Jacobian of g is

$$J(\boldsymbol{\phi}, \boldsymbol{\rho}) = \begin{bmatrix} 0 & -l_c c_{23} - l_t c_2 & -l_c c_{23} \\ l_t c_1 c_2 - d s_1 + l_c c_1 c_{23} & -s_1(l_c s_{23} + l_t s_2) & -l_c s_{23} s_1 \\ l_t c_2 s_1 + d c_1 + l_c s_1 c_{23} & c_1(l_c s_{23} + l_t s_2) & l_c s_{23} c_1 \end{bmatrix}, \tag{3.62}$$

It is easy to calculate their derivatives with respect to $\boldsymbol{\rho}$ through symbolic computation tools. And if we let $\dot{\boldsymbol{\phi}} = [\dot{\phi}_1; \dot{\phi}_2; \dot{\phi}_3]$ and $\boldsymbol{\omega} = [\omega_1; \omega_2; \omega_3]$, then

$$D = \begin{bmatrix} -\dot{\phi}_2 c_{23} - \dot{\phi}_3 c_{23} - \omega_2 c_{23} c_1 - \omega_3 c_{23} s_1 \\ \dot{\phi}_1 c_{23} c_1 - \omega_3 s_{23} + \omega_1 c_{23} c_1 - \dot{\phi}_2 s_{23} s_1 - \dot{\phi}_3 s_{23} s_1 \\ \omega_2 s_{23} + \dot{\phi}_1 c_{23} s_1 + \dot{\phi}_2 s_{23} c_1 + \dot{\phi}_3 s_{23} c_1 + \omega_1 c_{23} s_1 \end{bmatrix} \tag{3.63}$$

We can confirm $\boldsymbol{\rho}$ is observable because when $\dot{\boldsymbol{\phi}}$ and $\boldsymbol{\omega}$ are non-zero vectors, the rank of D is 1 regardless of the value of $\boldsymbol{\phi}$. Then $D^\top D$ is non-singular and the observability gramian will always be positive definite, thus $\boldsymbol{\rho}$ is observable. We can do the same calculation for $\boldsymbol{\rho} = [l_t; l_c]$ and show that it is observable as well.

Chapter 4

Cerberus: Low-Drift Visual-Inertial-Leg Odometry For Agile Locomotion

In this chapter, we present an open-source Visual-Inertial-Leg Odometry (VILO) state estimation solution for legged robots, called Cerberus, which precisely estimates position on various terrains in real-time using a set of standard sensors, including stereo cameras, IMU, joint encoders, and contact sensors. In addition to estimating robot states, we perform online kinematic parameter calibration and outlier rejection to substantially reduce position drift.

The VILO algorithm is based on the factor graph formulation of the state estimation problem. A factor graph is a graphical representation of a state estimation problem, where the nodes represent the state variables and the edges represent the measurement models. The factor graph is a generalization of the Kalman filter, as we will show in this chapter. Moreover, the factor graph leverages the sparsity of the measurement models to achieve efficient computation, which closely resembles the sparsity of the MPC problem. The next chapter will explore their connections in more detail, while the basic knowledge of the factor graph is introduced in this chapter.

4.1 Introduction

A sensor solution including only one pair of stereo cameras and critical proprioceptive sensors (IMU, joint encoders, and foot contact sensors) serves as an ideal choice for resource-constrained robots because this set of sensors is low cost, compact, and has low power consumption [16]. We call a state estimator using this sensing solution a Visual-Inertial-Leg Odometry (VILO) estimator. VILO fuses data from different sensors by constructing observation models that predict measurements given robot states. Observation models combined with a dynamics model of the robot form a factor graph [36] describing a nonlinear optimization problem whose solution is the maximum-likelihood state estimate. Prior work [54, 76, 158] has shown that VILO outperforms methods that only utilize a subset of the aforementioned sensors, such as Visual-Inertial-Odometry (VIO) [86] or Leg Odometry (LO) [24] alone.

A key feature of VIO estimators is online calibration of IMU biases using visual measurements[44]. Other key error sources in VIO have recently been systematically addressed [119]. However, in the VILO setting, systematic error analysis has yet to be established for leg sensors (joint encoders and contact sensors). Prior work [17, 159] and Chapter 3 have identified that when generating body velocity estimates using LO, error sources such as foot slippages, impacts, rolling contacts, and kinematic parameter errors could degrade velocity estimation accuracy. However, no prior work has studied how to handle these error sources in a VILO estimator.

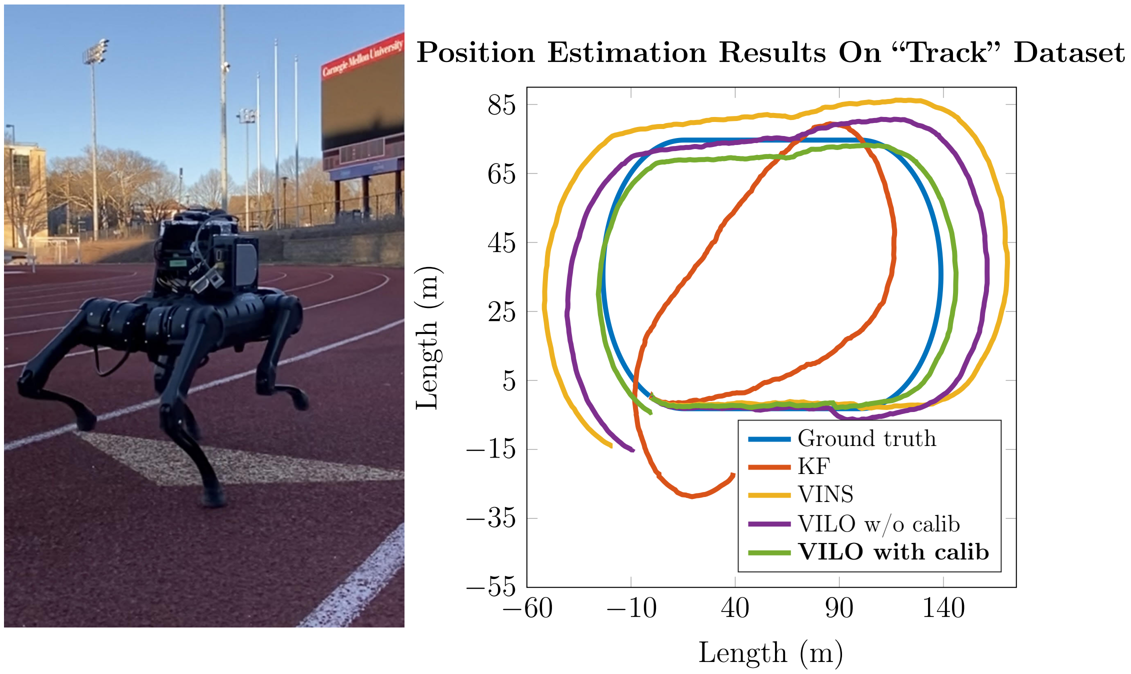


Figure 4-1: On the A1 robot, the Cerberus algorithm has lower than 1% position estimation drift after traveling 450m on standard stadium track, better than any baseline methods and better than any drift performance reported in literature using the same set of sensors. The ground truth is obtained using dimensions of standard running track.

Since different legged robots have different leg configurations, locomotion strategies, and sensor qualities, it is hard to fairly compare the performance of different VILO implementations. An open-source baseline VILO implementation and public datasets are needed for the benefit of the entire legged robot community.

As a first step toward establishing a standard VILO benchmark, we present a state-of-the-art real-time VILO algorithm called Cerberus that incorporates kinematic calibration for improved accuracy, as well as several datasets from two different quadruped robots. The algorithm implementation uses standard ROS interfaces to process sensor data and publish estimation results, and the datasets are in the format of ROS bags [137]. Docker [99] provides easy installation of a unified testing environment. Our contributions are:

- Cerberus, a VILO algorithm that estimates kinematic parameters online to achieve drift rates lower than any other results reported in the literature.
- Datasets collected on multiple robots in various indoor and outdoor environments to benchmark the Cerberus implementations.
- Open-source algorithm implementations using standard ROS interfaces that can be readily adapted to different robots and sensor configurations.

This chapter is organized as follows. In Section 4.2 we review related work. Section 4.3 introduces notation and provides background, we especially highlights the connection between Kalman filter and the factor graph. Section 4.3.3 presents a basic VILO algorithm. Section 4.4 derives an online kinematic calibration method in the Cerberus. Section 4.5 describes details of the algorithm implementation and presents hardware experiment results. Section 4.6 summarizes our conclusions.

4.2 Related Work

Many robots need to operate in GPS-denied environments use Visual odometry (VO) [128], which estimates robot pose using a monocular or a stereo camera, can provide a solution in these settings. By matching features across image sequences, feature locations constrain the possible motion of the camera so displacement can be solved from multiple-view geometry [52]. To improve the robustness and accuracy of the estimation, VIO [86] uses both the camera and the IMU as motion constraints.

As we introduced in the previous chapter, Kalman filtering (KF) is widely used in mobile robot and legged robot state estimation. Although KF is very efficient and easy to implement for fusing single sensors such as IMU and leg odometry, it is difficult to incorporate visual information. The drawback of KF is that it is based on the Markov assumption, which means the current state only depends on the previous state. However, a map point may be observed by the robot from multiple poses, which breaks the Markov assumption. Some prior works tried to address this problem by augmenting the KF state with a history of robot poses and map points [102]. However, the number of map points is usually very large, which makes the KF computationally expensive. Therefore, VIO solutions gradually shift from KF to optimization-based methods.

The optimization-based VIO methods formulate the state estimation problem as a nonlinear least-squares problem and solve it using Gauss-Newton or Levenberg-Marquardt algorithms [108]. The factor graph formulation [36] is a popular way to formulate the state estimation problem. In the factor graph, the nodes represent the state variables and the edges represent the measurement models. Preintegration [44] and inherent problem sparsity [36] can also help VIO to exploit problem structure, hence reducing computation cost. Depending on how sensor measurements are used, VIO has loosely-coupled or tightly-coupled approaches, which are two concepts used in GPS/INS community [156] . In the loosely-coupled approach, different sensors separately estimate the state. And then the estimations averaged to get the final result. While in the tightly-coupled approach, one integrated model for all sensors is designed.

After the development of several VIO algorithms [86, 118, 139], researchers continue to study how to reject different error sources in VIO including IMU biases, sensor time delay, and extrinsic parameter errors [119]. The position drift percentage, measuring how many meters the estimation deviates from the ground truth after traveling 100 meters, is often used as an important performance metric. Once the error sources are properly addressed, position drift of a VIO estimator can be as low as 0.29% on drones [118].

The factor graph formulation used in VIO can be easily extended to include the LO motion constraint, which leads to the VILO estimator [54, 75, 158]. [158] uses the velocity estimation result of a KF as the motion constraint. Contact preintegration is developed in [54], but bias correction is not performed. [159] describes the LO velocity bias and models it as a linear term that can be corrected in the preintegration. However, this bias model does not explain the source of the bias and its physical meaning. With the velocity bias model, [161] further shows that VILO can reach around 1% position drift with the aid of lidar, though their VILO implementation and datasets are not publicly available.

4.3 Background

We have extensively discussed the Kalman filter in the previous chapter. It's connection to the factor graph will be the focus of this section, which prepares for an introduction to the sliding window estimation and visual inertial-leg odometry in the coming sections.

4.3.1 An optimization view of state estimation

The KF can be viewed as an unconstrained optimization [58]. To show this, first, we introduce a way to write a standard Kalman filter as an optimization problem, then we show how this formulation naturally extends to more states and observations. This optimization form can additionally estimate parameters other than the robot's physical state. We gave sensor delay estimation as an example.

Linear Kalman Filter

Recall the Error-state Kalman filter we explained in Section 3.3.9, its corresponding linear form, if the system dynamics and observation are linear, is given as follows: For system

$$x_k = Fx_{k-1} + v_k, \quad v_k \sim \mathcal{N}(0, Q) \tag{4.1}$$

$$y_k = Hx_k + w_k, \quad w_k \sim \mathcal{N}(0, R) \tag{4.2}$$

As the standard Kalman filter procedure goes, given $\hat{x}_{k-1}$ and $\hat{P}_{k-1}$ at time step $k-1$. At time step k we receive a sensor measurement y_m, then we calculate

$$\hat{x}_{k|k-1} = F\hat{x}_{k-1} \tag{4.3a}$$

$$\hat{P}_{k|k-1} = F\hat{P}_{k-1}F^T + Q \tag{4.3b}$$

$$S_k = H\hat{P}_{k|k-1}H^T + R \tag{4.3c}$$

$$K_k = \hat{P}_{k|k-1}H^T S^{-1} \tag{4.3d}$$

$$\hat{x}_k = \hat{x}_{k|k-1} + K_k(y_m - H\hat{x}_k) \tag{4.3e}$$

$$\hat{P}_k = (I - K_k H)\hat{P}_{k|k-1} \tag{4.3f}$$

where $\hat{x}_k$ and $\hat{P}_k$ are estimated posterior mean and covariance of the state [145].

It can be shown that Kalman filter steps show in (4.3) are equivalent to the following procedure [58]:

Given $\hat{x}_{k-1}$ and $\hat{P}_{k-1}$ at time step $k-1$. At time step k we receive a sensor measurement y_m, then we formulate

$$\begin{aligned}\min_{x_{k-1},x_k} J = \|x_{k-1} - \hat{x}_{k-1}\|^2_{\hat{P}^{-1}_{k-1}} \\ + \|x_k - Fx_{k-1}\|^2_{Q^{-1}} + \|y_m - Hx_k\|^2_{R^{-1}}\end{aligned} \tag{4.4}$$

Solve for x^*_{k-1},x^*_k, where $*$ stands for the minimizer of the above problem. We also calculate the Hessian matrix [108] at the solution as

$$\nabla^2 J = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix}$$

Let

$$\hat{x}_k = x^*_k \tag{4.5}$$

$$\hat{P}_k = (P_{22} - P_{12}P^{-1}_{11}P_{21})^{-1} \tag{4.6}$$

The final $\hat{x}_k$ and $\hat{P}_k$ gives the same posterior mean and covariance as the KF in Eqn. (4.3f).

To prove this optimization is equivalent to the KF, it is straightforward to compute and solve $\nabla J = 0$ to see its equivalence to KF steps. The proof of Eqn (4.6) is more involved, we write out the Hessian of Eqn (4.4) as

$$\nabla^2 J = \begin{bmatrix} \hat{P}^{-1}_{k-1} + F^T Q^{-1} F & F^T Q^{-1} \\ Q^{-1} F & Q^{-1} + H^T R^{-1} H \end{bmatrix}$$

This can be verified by expanding the J and take derivative twice w.r.t $[x_{k-1}; x_k]$.

Perform a Schur compliment according to Eqn (4.6), then

$$\begin{aligned}P_{22} - P_{12}P_{11}^{-1}P_{21} =& Q^{-1} + H^T R^{-1} H - Q^{-1} F (\hat{P}_{k-1}^{-1} + F^T Q^{-1} F)^{-1} F^T Q^{-1} \\ =& H^T R^{-1} H + (Q + F\hat{P}_{k-1}F^T)^{-1}\end{aligned}$$

The simplification step is done using the matrix inversion formula:

$$(A - BD^{-1}C)^{-1} = A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1}$$

On the other hand, inserting Eqn (4.3b) and Eqn (4.3d) into Eqn (4.3f), with applying the matrix inversion formula again, one can see that

$$\hat{P}_k = [(Q + F\hat{P}_{k-1}F^T)^{-1} + H^T R^{-1} H]^{-1} = [P_{22} - P_{12}P_{11}^{-1}P_{21}]^{-1} \tag{4.7}$$

A deeper meaning of this result is, the hessian of the cost function at the solution describes the joint distribution of x_{k-1} and x_k in the information matrix form as $p(x_{k-1}, x_k) \sim \mathcal{N}(\begin{bmatrix} x_{k-1}^* \\ x_k^* \end{bmatrix}, \nabla^2 J^{-1})$. And Eqn (4.6) corresponds to marginalize x_{k-1} out to get the posterior distribution of x_k alone as $p(x_k|\hat{x}_{k-1}; y_m) \sim \mathcal{N}(x_k^*, P_k^*)$. In the next iteration of the filter, this serves as a part of prior distribution of the next state to estimate ($p(x_k, x_{k+1})$).

It is worth visualizing terms in Problem (4.4) using the graphical model we introduced in Section 2.3.2. As shown in Figure 4-2, where the correspondance is self-explanatory. The most important thing to emphasis is a "factor" can be viewed in two different ways, either a term in the cost function of an optimization problem, or a Gaussian distribution. For example, $\|x - \hat{x}\|^2_{\hat{P}^{-1}}$ can be interpreted as that

- at the minimizer, x^* must be as close to $\hat{x}$ as possible, or

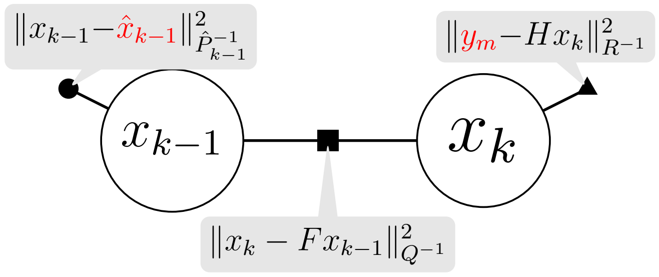

$\|x_{k-1} - \hat{x}_{k-1}\|^2_{\hat{P}^{-1}_{k-1}}$
$\|y_m - Hx_k\|^2_{R^{-1}}$
x_{k-1}
x_k
$\|x_k - Fx_{k-1}\|^2_{Q^{-1}}$

Figure 4-2: The graphical representation of a Kalman filter. The circles are estimation states. The line segment with square represents the process dynamic model. The unitary dot line is the prior information of the state distribution. And the line segment with square is the measurement model. This graph follows the convention of standard factor graphs [36], where all the circles are "nodes" and line segments are "factors".

- x has a prior Gaussian distribution $\mathcal{N}(\hat{x}, P)$.

Also Figure 4-3 describes the marginalization process to get the posterior distribution. Using the factor graph jargon, the marginalization process is called "variable elimination" [36]. We will differ a detailed explanation of the variable elimination algorithm to the next chapter, here we want to emphasize that from Figure 4-2 to Figure 4-3(a), we remove a node from the graph and add a new factor (red unitary factor) to the graph, the new factor is the posterior distribution of x_k.

Sliding Window Estimation As A Factor Graph

The KF's optimization counterpart (Problem (4.4)) makes it easy to think about adding more states, which is called the sliding window estimation (SWE) [132] or moving horizon estimation [2]: we keep track of $p(x_{k-M}, x_{k-M+1}, \ldots, x_{k-1}, x_k)$, solve a similar optimization problem at each time step, marginalize one robot state and add one more state.

More specifically, at time step $k-1$, we have $\hat{x}_{k-M:k-1}$ and $\hat{P}_{k-M:k-1}$, the prior

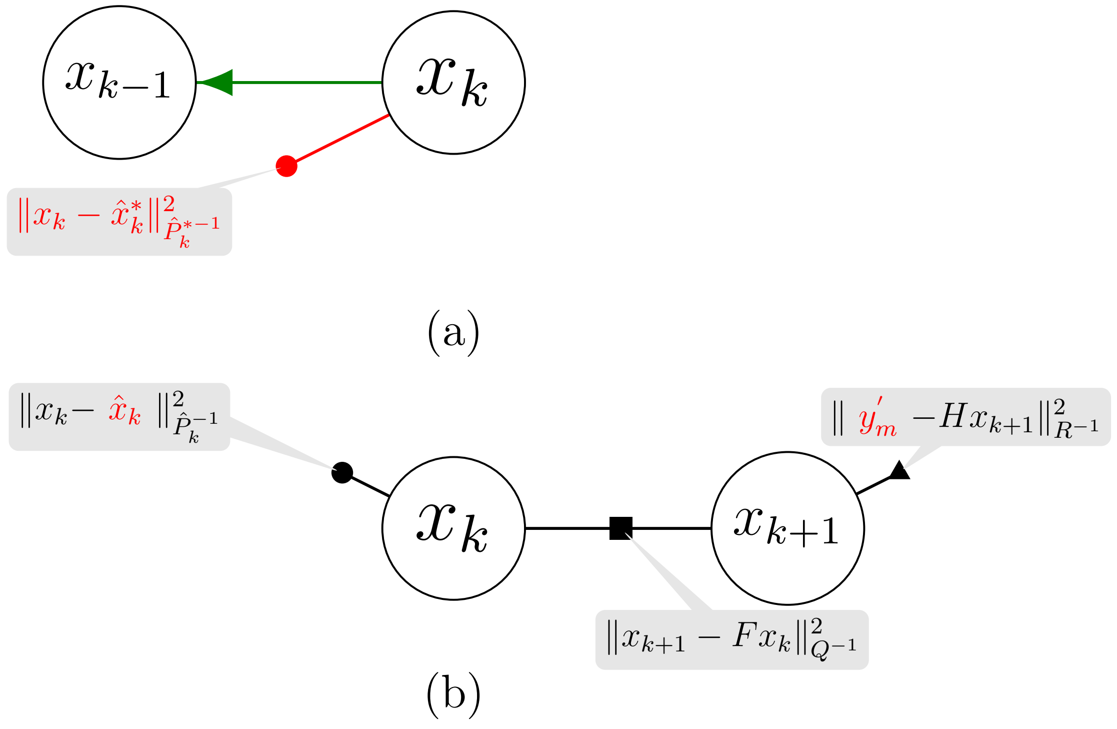

x_{k-1}
x_k
$\|x_k - \hat{x}_k^*\|^2_{\hat{P}_k^{*-1}}$
(a)
$\|x_k - \hat{x}_k\|^2_{\hat{P}_k^{-1}}$
x_k
x_{k+1}
$\| y_m' - Hx_{k+1}\|^2_{R^{-1}}$
$\|x_{k+1} - Fx_k\|^2_{Q^{-1}}$
(b)

Figure 4-3: **Top (a):** Marginalize x_{k-1} out of the joint distribution $p(x_{k-1}, x_k)$ to get the posterior distribution of x_k alone. The green arrow represents the Bayes net after eliminating variable x_{k-1} from the factor graph. The new red unitary factor is a new factor results from the elimination process and represents the posterior distribution. **Bottom (b):** Add the next state and construct a new Kalman filter problem.

distribution of a window of past M robot states. We also have y_m at time step k. The sliding window estimation is as follows

$$\begin{aligned}\min_{x_{k-M:k-1},x_k} J = \|x_{k-M:k-1} - \hat{x}_{k-M:k-1}\|^2_{\hat{P}^{-1}_{k-M:k-1}} \\ + \sum_{i=k-M}^{k-1} \|x_{i+1} - Fx_i\|^2_{Q^{-1}} + \|y_m - Hx_k\|^2_{R^{-1}}\end{aligned} \tag{4.8}$$

Solve the problem as a conventional unconstrained optimization problem to get the optimal solution $x^*_{k-M:k-1}$,x^*_k. Like in Problem (4.4), again we can calculate the Hessian matrix at the solution

$$\nabla^2 J = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix}, \text{ where } \dim(P_{11}) = \dim(x_{k-M})$$

Then we conduct a marginalization or variable elimination process to get the Gaussian posterior distribution of states $x_{k-M+1:k}$ as

$$\begin{aligned}\hat{x}_{k-M+1:k} &= x^*_{k-M+1:k} \\ \hat{P}^{-1}_{k-M+1:k} &= P_{22} - P_{21}P_{11}^{-1}P_{12}\end{aligned}$$

The procedure also gives a recursive algorithm that allows real-time computation. However, when M and the dimension of the state are large, the Schur complement will become very computationally intensive. We must exploit the problem sparsity to speed up the computation. The factor graph formulation of this problem has been explored in [63], where the factor graph and the variable elimination algorithm make it easy to conduct sparse matrix operations for marginalization. The whole prodecure is illustrated in Figure 4-4.

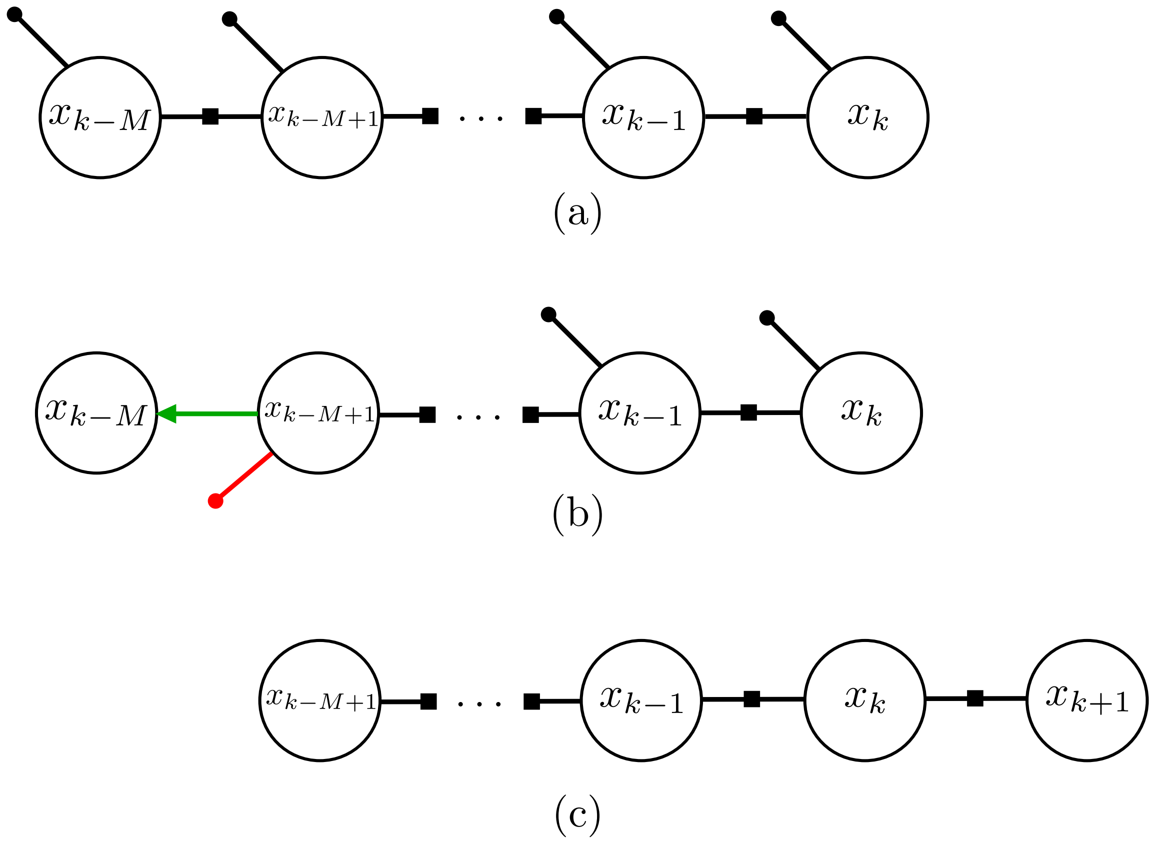


Figure 4-4: **Top (a):** The factor graph formulation of a SWE. **Middle (b):** Marginalized the last state x_{k-M}. **Bottom (c):** Add a next state x_{k+1}.

We also want to note three things. First, the SWE is equivalent to the KF if $M = 1$. Second, if we do not do marginalization at all but keep adding new states and measurements into the window, then we get the Kalman smoother [25]. Lastly, for nonlinear problems where cost terms involve nonlinear functions, we can iteratively linearize the problem and solve the linearized version using the same formulation until convergence.

4.3.2 Estimating Additional Parameters

The optimization view of the SWE make it easy to not only add more states but also estimate different parameters. In Chapter 3 we have discussed how to estimate legged robot kinematic parameters in a KF. It is not hard to see these additional parameters fit into the optimization framework naturally as well. Similarly, parameters such as transformations among sensors [73] and IMU biases [87] can also be estimated together with other quantities as a part of the robot state.

However, there is another type of parameter that is more difficult, namely sensor temporal offset. Figure 4-5 shows the definition of sensor temporal offset. In the literature Temporal offset has been addressed with a few assumptions. If delay time t_d is known, then simply calculate t_s and recalculate the estimator from t_s to the current time [82]. If the delay time is not known and must be estimated but the sensor has special properties, for example, Sensor 1 is a position sensor and we have velocity estimation in the state, then the measurement from Sensor 1 can be shifted by the current estimation of t_d [87]. The most general setting is presented in [47] where instead of shifting sensor measurement, the states are interpolated to the estimated t_s to form an observation model in moving horizon estimation (an alternative name to sliding window estimation). This method suits most types of sensors. However,

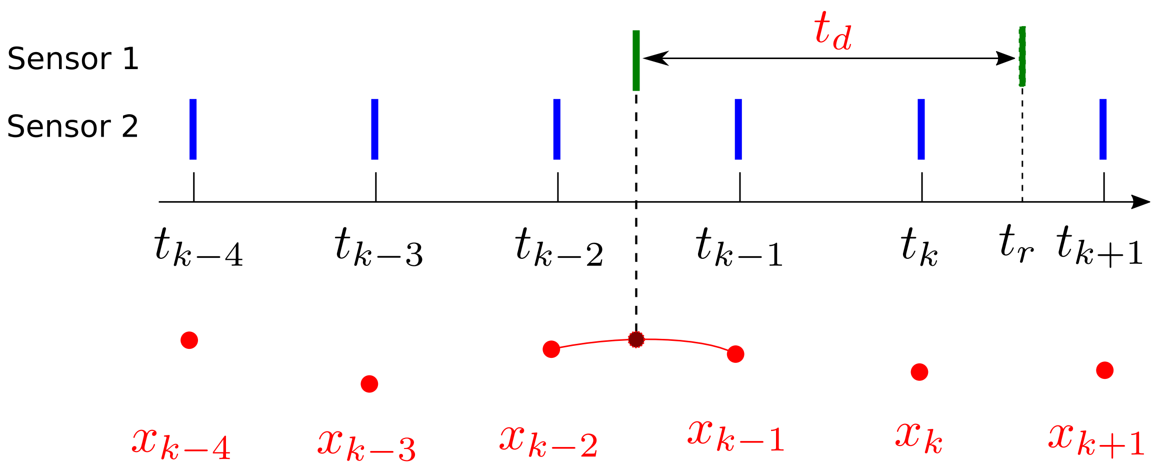


Figure 4-5: After receiving sensor data at time t_r, we search in the sliding window backward to find states close to $t_s = t_r - t_d$ (x_{k-2} and x_{k-1}), then we do a cubic spline interpolation between $\bar{x}_{k-2}$ and $\bar{x}_{k-1}$ to get an interpolated state $x_{\text{interp}}(t_d, \bar{x}_{k-2}, \bar{x}_{k-1})$.

the interpolation range needs to be constrained. In many applications, the temporal offset may be unknown and changing. We only know a rough bound of the offset.

In SWE, assume the original estimation problem has state $\bar{x}$, we augment the state to include an estimation of the offset time $x = [\bar{x}; t_d]$. When we receive a Sensor 1's measurement y_m at time t_r, we can create an observation model by interpolating states close to $t_r - \hat{t}_d$ to get an x_{interp}, if our estimation of x is accurate, then $y_m - h(x_{\text{interp}})$ should have a small value.

With the above observation model, we can write time offset estimation problem as

$$\begin{aligned}\min_{x_{k-M:k}} J = \|x_{k-M:k-1} - \hat{x}_{k-M:k-1}\|^2_{\hat{P}^{-1}_{k-M:k-1}} + \sum_{i=k-M}^{k-1} \|x_{i+1} - f(x_i)\|^2_{Q^{-1}} \\ + \|y_m - h_1(x_{\text{interp}}(t_d, \bar{x}_{k-2}, \bar{x}_{k-1}))\|^2_{R_1^{-1}} + \|y'_m - h_2(x_k)\|^2_{R_2^{-1}} \\ \text{s.t.} \quad 0 \le t_d \le t_{max}\end{aligned} \tag{4.9}$$

The y'_m is the sensor measurement from sensor 2. h_1 and h_2 are sensor observation function of the two sensors respectively.

This Problem (4.9) is a nonlinear constrained optimization problem. We can iteratively linearize the problem and solve the linearized version, which is a quadratic program with linear constraints.

4.3.3 Visual-Inertial-Leg Odometry

A typical VILO framework [54, 76, 159] keeps track of the estimation of a list of past N states $\hat{\boldsymbol{x}}_k$ and M camera feature locations $\hat{\lambda}_l$ as $\mathcal{X} = \{\hat{\boldsymbol{x}}_0, \hat{\boldsymbol{x}}_1, \ldots \hat{\boldsymbol{x}}_N, \hat{\lambda}_0, \hat{\lambda}_1, \ldots \hat{\lambda}_M\}$. The robot state is $\hat{\boldsymbol{x}}_k = [\hat{\boldsymbol{p}}_k; \hat{\boldsymbol{q}}_k; \hat{\boldsymbol{v}}_k; \hat{\boldsymbol{b}}_{ak}; \hat{\boldsymbol{b}}_{\omega k}]$, where $\hat{\boldsymbol{p}}_k \in \mathbb{R}^3$ is the robot position in the world frame, $\hat{\boldsymbol{q}}_k$ is the robot's orientation quaternion, and $\hat{\boldsymbol{v}}_k \in \mathbb{R}^3$ is the linear velocity of the robot's body represented in the world frame. $\hat{\boldsymbol{b}}_{ak} \in \mathbb{R}^3$ and $\hat{\boldsymbol{b}}_{\omega k} \in \mathbb{R}^3$ are IMU accelerometer bias and gyroscope bias. A new state $\hat{\boldsymbol{x}}_k$ is created each time t_k when a new camera image arrives. Also, sensors on the robot generate measurements $Z_t = \{\hat{\boldsymbol{a}}_m(t), \hat{\boldsymbol{\omega}}_m(t), \hat{\boldsymbol{\phi}}_j(t), \hat{\dot{\boldsymbol{\phi}}}_j(t)\}$ and Λ_t periodically, where $\hat{\boldsymbol{a}}_m$ and $\hat{\boldsymbol{\omega}}_m$ are IMU linear acceleration and angular velocity, $\hat{\boldsymbol{\phi}}_j$ and $\hat{\dot{\boldsymbol{\phi}}}_j$ are joint angle and joint angle velocity for each leg j, and Λ_t is a set of feature coordinates on the camera images who have known associations with feature locations in $\mathcal{X}$. We denote $\mathcal{Z}$ as all measurements between state $\hat{\boldsymbol{x}}_0$ and $\hat{\boldsymbol{x}}_N$. We also denote subsets $\mathcal{X}_{sub} \subset \mathcal{X}$ and $\mathcal{Z}_{sub} \subset \mathcal{Z}$. The VILO constructs a nonlinear least-squares problem to find $\mathcal{X}$ as the solution of

$$\min_{\mathcal{X}^*} \left\{ \sum_i \left\| \boldsymbol{r}_i(\mathcal{X}_{sub}, \mathcal{Z}_{sub}) \right\|^2_{P_i} \right\}, \tag{4.10}$$

where each term $\boldsymbol{r}_i(\mathcal{X}_{sub}, \mathcal{Z}_{sub})$ defines a measurement residual function. Ideally the cost should be $\mathbf{0}$ at optimal solution $\mathcal{X}^*$. P_i is a weighting matrix that encodes the

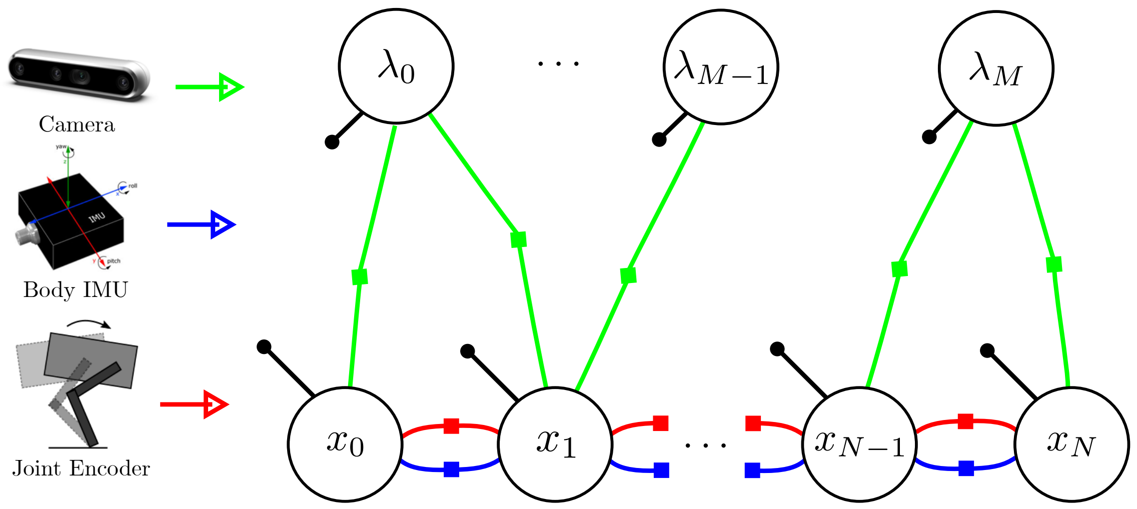


Figure 4-6: The factor graph representation of a VILO. The green factors are related to camera images, blue factors are formulated by IMU sensor data, red factors are from leg kinematic parameters

relative uncertainty in each $\boldsymbol{r}_i$, and also takes the same set of inputs. Problem (4.10) can be solved by nonlinear optimization methods [36]. The core technical challenge is to design cost functions and their uncertainties leveraging all available sensor data. Figure 4-6 illustrates the factor graph formulation, which is obviously an SWE.

Additionally, a VILO estimator usually has other mechanisms to ensure real-time computation, such as visual feature tracking and IMU preintegration. See [118, 158] for more details. We will focus on talking about the preintegration in the next section.

4.3.4 Preintegration

A key technique used in VIO and VILO to improve computation efficiency is preintergration. When fusing camera data and IMU data with different frequencies, preintegration [44] is used to integrate multiple IMU measurements between two camera image times into a single "motion constraint" in the cost function, so the estimator only needs to add states at the camera frequency instead of keeping up with the much higher frequency of the IMU. More importantly, it is well known that IMUs are biased [1], and biases should be estimated along with robot physical states. When the estimator updates IMU biases, IMU preintegration can avoid integrating measurements again by directly updating the integration term using its first order approximation. IMU preintergration is used in several real-time VIO algorithms [118, 139, 149]. Similarly, contact preintegration is used to integrate joint encoder data into motion constraints in VILO [56].

We assume there are L IMU measurements between state $\hat{\boldsymbol{x}}_k$ and $\hat{\boldsymbol{x}}_{k+1}$, and that each IMU measurement arrives δt after the previous one. Let $i \in \{1 \dots L\}$ be the measurement index and $\Delta t = t_{k+1} - t_k$, then $t_1 = t_k$ and $t_L = t_{k+1}$. As shown in Figure 4-7, we can integrate these IMU measurements into a single motion measurement.

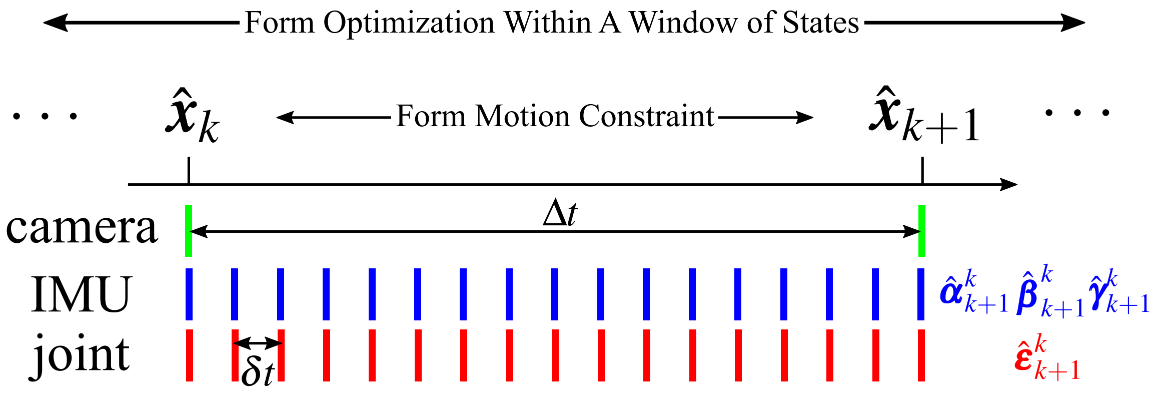
First, let $\hat{\boldsymbol{\gamma}}_i^k$ denote quaternion rotation from frame b_k to frame b_i, the robot body frame at time t_i. Starting from $\hat{\boldsymbol{\gamma}}_k^k = \boldsymbol{q}_I$, we can calculate

$$\hat{\boldsymbol{\gamma}}_{i+1}^k = R(\frac{1}{2}\begin{bmatrix} 0 \\ (\hat{\boldsymbol{\omega}}_m(t_i) - \hat{\boldsymbol{b}}_{\omega k})\delta t \end{bmatrix})\hat{\boldsymbol{\gamma}}_i^k, \tag{4.11}$$

which recursively leads to $\hat{\boldsymbol{\gamma}}_{k+1}^k$, a measurement of the rotation difference between $\hat{\boldsymbol{q}}_k$ and $\hat{\boldsymbol{q}}_{k+1}$. Another two recursive relations can be derived using acceleration data as

$$\hat{\boldsymbol{\alpha}}_{i+1}^k = \hat{\boldsymbol{\alpha}}_i^k + \hat{\boldsymbol{\beta}}_i^k \delta t, \text{ and} \tag{4.12}$$

$$\hat{\boldsymbol{\beta}}_{i+1}^k = \hat{\boldsymbol{\beta}}_i^k + A(\hat{\boldsymbol{\gamma}}_i^k)(\hat{\boldsymbol{a}}_m(t_i) - \hat{\boldsymbol{b}}_{ak})\delta t, \tag{4.13}$$



Form Optimization Within A Window of States
$\hat{\boldsymbol{x}}_k$
Form Motion Constraint
$\hat{\boldsymbol{x}}_{k+1}$
camera
Δt
IMU
$\hat{\boldsymbol{\alpha}}^k_{k+1}\,\hat{\boldsymbol{\beta}}^k_{k+1}\,\hat{\boldsymbol{\gamma}}^k_{k+1}$
joint
δt
$\hat{\boldsymbol{\varepsilon}}^k_{k+1}$

Figure 4-7: Preintergation of IMU or joint measurements. The estimator adds a new state whenever a new camera image arrives. Within the time interval Δt between two consecutive states, there are multiple other sensor measurements with a sub-interval δt. We use k to refer to estimator time indices and i to indicate a measurement within Δt. These measurements are integrated according to equations (4.12), (4.13), (4.11), and (4.20), forming a motion constraint on states as in equation (4.14). The VILO Problem (4.10) involves such motion constraints and other constraints due to visual observations over a window of states.

such that $\hat{\boldsymbol{\alpha}}^k_{k+1}$ and $\hat{\boldsymbol{\beta}}^k_{k+1}$ measure position and velocity differences between two states. These so called preintegration terms [44] describe a cost function on states as [118]

$$\boldsymbol{r}(\hat{\boldsymbol{x}}_k, \hat{\boldsymbol{x}}_{k+1}, Z_{\Delta k}) = \begin{bmatrix} A(\hat{\boldsymbol{q}}_k)^T(\hat{\boldsymbol{p}}_{k+1} - \hat{\boldsymbol{p}}_k + \frac{1}{2} g^w \Delta t^2 - \hat{\boldsymbol{v}}_k \Delta t) - \hat{\boldsymbol{\alpha}}^k_{k+1} \\ \Phi^{-1}(\hat{\boldsymbol{q}}_k^{-1} \otimes \hat{\boldsymbol{q}}_{k+1} \otimes (\hat{\gamma}^k_{k+1})^{-1}) \\ A(\hat{\boldsymbol{q}}_k)^T(\hat{\boldsymbol{v}}_{k+1} + g^w \Delta t - \hat{\boldsymbol{v}}_k) - \hat{\boldsymbol{\beta}}^k_{k+1} \\ \hat{\boldsymbol{b}}_{ak+1} - \hat{\boldsymbol{b}}_{ak} \\ \hat{\boldsymbol{b}}_{\omega k+1} - \hat{\boldsymbol{b}}_{\omega k} \end{bmatrix}, \tag{4.14}$$

where $Z_{\Delta k}$ represents all measurements during Δt.

The error dynamics [118] of $\boldsymbol{r}$ as

$$\boldsymbol{e}_{i+1} = \begin{bmatrix} I & I\delta t & 0 & 0 & 0 \\ 0 & I & -A(\hat{\boldsymbol{\gamma}}_i^k)\lfloor \hat{\boldsymbol{a}}_m(t_i) - \hat{\boldsymbol{b}}_{ak} \rfloor^\times \delta t & -A(\hat{\boldsymbol{\gamma}}_i^k)\delta t & 0 \\ 0 & 0 & I - \lfloor \hat{\boldsymbol{\omega}}_m(t_i) - \hat{\boldsymbol{b}}_{\omega k} \rfloor^\times \delta t & 0 & -I\delta t \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix} \boldsymbol{e}_i$$

$$+ \begin{bmatrix} 0 & 0 & 0 & 0 \\ -A(\hat{\boldsymbol{\gamma}}_i^{b_k})\delta t & 0 & 0 & 0 \\ 0 & -I\delta t & 0 & 0 \\ 0 & 0 & I\delta t & 0 \\ 0 & 0 & 0 & I\delta t \end{bmatrix} \begin{bmatrix} \boldsymbol{n}_a \\ \boldsymbol{n}_\omega \\ \boldsymbol{n}_{ba} \\ \boldsymbol{n}_{b\omega} \end{bmatrix} = F_i \boldsymbol{e}_i + G_i \boldsymbol{n}_{IMU}, \tag{4.15}$$

where $\boldsymbol{n}_a$ and $\boldsymbol{n}_\omega$ are IMU sensor measurement noises and $\boldsymbol{n}_{ba}$ and $\boldsymbol{n}_{b\omega}$ are random walk noises for IMU biases. $\boldsymbol{e}_i = [\delta\boldsymbol{\alpha}_i^k; \delta\boldsymbol{\beta}_i^k; \delta\boldsymbol{\theta}_i^k; \delta\boldsymbol{b}_{ai}; \delta\boldsymbol{b}_{\omega i}]$ is a vector describing the errors between preintegration terms and their "true" values after each IMU measurement integration [118]. $\delta\boldsymbol{\alpha}_i^k = \boldsymbol{\alpha}_i^k - \hat{\boldsymbol{\alpha}}_i^k$, $\delta\boldsymbol{\beta}_i^k$, and $\boldsymbol{\gamma}_i^k = L(\hat{\boldsymbol{\gamma}}_i^k)\Phi(\delta\boldsymbol{\theta}_i^k)$. Details of the derivation can be seen in [118].

Let Q be the noise covairance matrix of $\boldsymbol{n}_{IMU}$. We can also recursively calculate P_{k+1}^k and J_{k+1}, the error jacobian, as follows

$$P_{i+1}^k = F_i P_i^k F_i^T + G_i Q G_i^T, P_1^k = 0, \tag{4.16}$$

$$J_{i+1} = F_i J_i, J_i = I. \tag{4.17}$$

The error jacobian can greatly reduce VILO computation time: When solving Problem (4.10) using numerical methods, a solver iteratively calculates state update

Type & Model	No.	Freq.	Output Description
D435 camera [64]	1	15Hz	A pair of stereo images
Robot built-in IMU	1	500Hz	Linear acceleration & angular velocity
Robot built-in joint encoder	12	500Hz	Joint motor angles & angle velocities
Robot built-in contact sensor	4	500Hz	Binary foot contact flag

Table 4.1: VILO Sensor List

vectors $\delta\boldsymbol{x}$, and the update will change IMU biases. Instead of reintegrating the preintegration terms that depend on IMU biases, with the error jacobian, we can directly update the preintegration terms, for example, as

$$\boldsymbol{\alpha}_{k+1}^{k} = \hat{\boldsymbol{\alpha}}_{k+1}^{k} + J_{a}^{\alpha}\delta\boldsymbol{b}_{a} + J_{\omega}^{\alpha}\delta\boldsymbol{b}_{\omega} \tag{4.18}$$

to get their revised values, where J_{a}^{α} and J_{ω}^{α} are blocks in J_{k+1} that correspond to $\partial\boldsymbol{\alpha}/\partial\boldsymbol{b}_{a}$ and $\partial\boldsymbol{\alpha}/\partial\boldsymbol{b}_{\omega}$.

4.4 Kinematic Calibration In Preintegration

In this section we show, in the Cerberus, how to estimate $\boldsymbol{\rho}$ for each leg discussed in Chapter 3 by including them into the state so $\hat{\boldsymbol{x}}_{k} = [\hat{\boldsymbol{p}}_{k}; \hat{\boldsymbol{q}}_{k}; \hat{\boldsymbol{v}}_{k}; \hat{\boldsymbol{b}}_{ak}; \hat{\boldsymbol{b}}_{\omega k}; \hat{\boldsymbol{\rho}}_{jk}]$, where j is the leg index. For brevity, we only describe the case $j = 1$ but the algorithm can easily apply to robots with more legs.

4.4.1 Contact Preintegration

For a leg that has non-slipping contact with the ground, (3.44) describes body velocity estimation through LO. This velocity can be integrated into a body displacement.

We again focus on integrating measurements between state $\hat{\boldsymbol{x}}_k$ and $\hat{\boldsymbol{x}}_{k+1}$ including sensor data from leg sensors, then have a revised constraint equation

$$\boldsymbol{r}'(\hat{\boldsymbol{x}}_k, \hat{\boldsymbol{x}}_{k+1}, Z_{\Delta k}) = \begin{bmatrix} \boldsymbol{r}(\hat{\boldsymbol{x}}_k, \hat{\boldsymbol{x}}_{k+1}, Z_{\Delta k}) \\ A(\hat{\boldsymbol{q}}_k)^T(\hat{\boldsymbol{p}}_{k+1} - \hat{\boldsymbol{p}}_k) - \hat{\boldsymbol{\epsilon}}^k_{k+1} \\ \hat{\boldsymbol{\rho}}_{k+1} - \hat{\boldsymbol{\rho}}_k \end{bmatrix}, \tag{4.19}$$

where $\hat{\boldsymbol{\epsilon}}^k_{k+1}$ is the integration result of

$$\hat{\boldsymbol{\epsilon}}^k_{i+1} = \hat{\boldsymbol{\epsilon}}^k_i + A(\hat{\boldsymbol{\gamma}}^{b_k}_i)\hat{\boldsymbol{v}}_i \delta t, \text{ where} \tag{4.20}$$

$$\hat{\boldsymbol{v}}_i = -[J(\hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\rho}})\hat{\dot{\boldsymbol{\phi}}} + \lfloor \hat{\boldsymbol{\omega}} - \hat{\mathrm{b}}_{\omega k} \rfloor^{\times} g(\hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\rho}})]. \tag{4.21}$$

Comparing to (4.14), (4.19) introduces the LO velocity integration as a measurement model of body positions. The term $\hat{\boldsymbol{\epsilon}}^k_{k+1}$ depends on sensor measurements, $\hat{\boldsymbol{b}}_{\omega k}$, and $\hat{\boldsymbol{\rho}}_k$. A version without kinematic parameter dependency is previously derived in [54]. The error of this measurement, defined as $\boldsymbol{e}'_i = [\boldsymbol{e}_i; \delta\boldsymbol{\epsilon}^k_i; \delta\boldsymbol{\rho}_i]$, has dynamics

$$\begin{aligned} \boldsymbol{e}'_{i+1} = & \begin{bmatrix} & & F_i & & \mathbf{0} & & \\ 0 & I & -A(\hat{\boldsymbol{\gamma}}^{b_k}_i)\lfloor \hat{\boldsymbol{v}}_i \rfloor^{\times}\delta t & 0 & \boldsymbol{\zeta}\delta t & 0 & \boldsymbol{\kappa}\delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{e}'_i \\ & + \begin{bmatrix} & G_i & & & & \mathbf{0} & & \\ 0 & \boldsymbol{\zeta}\delta t & 0 & 0 & \boldsymbol{\eta}\delta t & A(\hat{\boldsymbol{\gamma}}^{b_k}_i)J\delta t & I\delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & I\delta t \end{bmatrix} \begin{bmatrix} \boldsymbol{n}_{IMU} \\ \boldsymbol{n}_{\phi} \\ \boldsymbol{n}_{\dot{\phi}} \\ \boldsymbol{n}_{v} \\ \boldsymbol{n}_{\rho} \end{bmatrix}, \end{aligned} \tag{4.22}$$

in which J is short for $J(\boldsymbol{\phi}, \hat{\boldsymbol{\rho}})$, the forward kinematics Jacobian. $\boldsymbol{e}_t$, $\boldsymbol{n}_t$, F_t, and

G_t are defined in 4.15. The definitions of $\boldsymbol{\zeta}$, $\boldsymbol{\eta}$, and $\boldsymbol{\kappa}$, along with the derivation of the error dynamics, are in the Appendix. $\boldsymbol{n}_\phi \sim \mathcal{N}(0, \sigma_\phi^2)$ and $\boldsymbol{n}_{\dot{\phi}} \sim \mathcal{N}(0, \sigma_{\dot{\phi}}^2)$ are the measurement noise of joint angle and joint angle velocity. $\boldsymbol{n}_\rho \sim \mathcal{N}(0, \sigma_\rho^2)$ is the kinematic parameter random walk noise. $\boldsymbol{n}_v \sim \mathcal{N}(0, \sigma_v^2)$ is the uncertainty of the contact preintegration motion constraint.

From the error dynamics, we can get P_{k+1}^k and J_{k+1} as in (4.16) and (4.17). Then Jacobians such as $J_\rho^\epsilon = \frac{\partial \epsilon_{k+1}^k}{\partial \rho}$ extracted from J_{k+1}^k can allow fast preintegration updates:

$$\boldsymbol{\epsilon}_{k+1}^k = \hat{\boldsymbol{\epsilon}}_{k+1}^k + J_\omega^\epsilon \delta \boldsymbol{b}_\omega + J_\rho^\epsilon \delta \boldsymbol{\rho}. \tag{4.23}$$

This technique is critical for enabling real-time computation of the Cerberus while doing kinematic calibration.

4.5 Experiments

Our C++ implementation of the Cerberus uses the factor graph optimizer and vision front end of the open source visual-inertial odometry software VINS-Fusion [118]. The IMU factor in VINS-Fusion is replaced with our proposed cost function (4.19). We set $\boldsymbol{\rho} = [l_c]$, the calf length shown in Fig. 2-2 as it is changing during locomotion [163]. We conducted experiments on sensor data collected on two quadruped robot platforms, the Unitree A1 and Go1 [148]. Both robots perform trotting using different controller implementations. The list of sensors that provide data to our state estimator is summarized in Table 4.1.

We focus on comparing the position drift percentages of a Kalman Filter (KF) [19], visual-inertial odometry (VINS) [118], visual-inertia-leg odometry without kinematics calibration (VILO w/o calib), and the Cerberus (VILO with calib). The only difference between the last two is the VILO w/o calib just uses a fixed value $\boldsymbol{\rho} = [0.21m]$ while the Cerberus calibrates the kinematic parameters.

Before focusing on Cerberus, we explain two important parameter estimation function of Cerberus individually.

4.5.1 Parameter Estimation

Kinematic Parameter

We also test our measurement model in the context of an optimization-based sliding-window estimator. We add the kinematic parameters and the body velocity measurement model (3.51) to the open-source VINS-Fusion [118], one of the most popular sliding-window estimators. The modified estimator takes inputs from a single Intel Realsense D435 camera and other legged robot proprioceptive sensors as measurements. The total cost of the sensor hardware is less than \$2000. We run the estimator on the standing-up datasets. The kinematic calibration can be done quickly when the robot stands up. Fig. 4-8 shows the estimated l_c values of different runs with different initial calf length values. The final mean length value after the robot finished standing up (6s-11s) is 0.215m, which agrees with the previous experiment using motion-capture data. However, we observe larger variance with the sliding-window estimator. We attribute this difference to the use of visual sensors, which are less accurate than the motion capture system used in the EKF. This experimental result confirms that we can perform kinematics calibration within an optimization-based estimator using low-cost onboard sensors, while quantitative analysis remains future work.

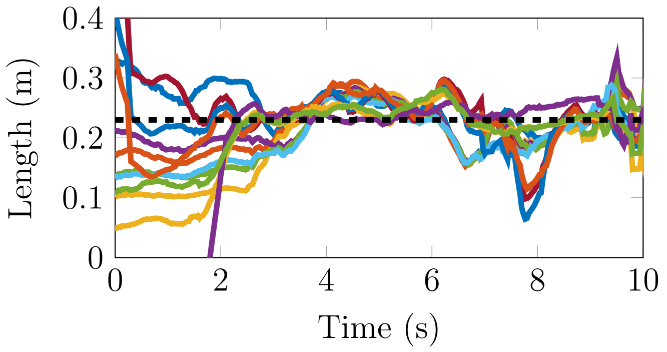
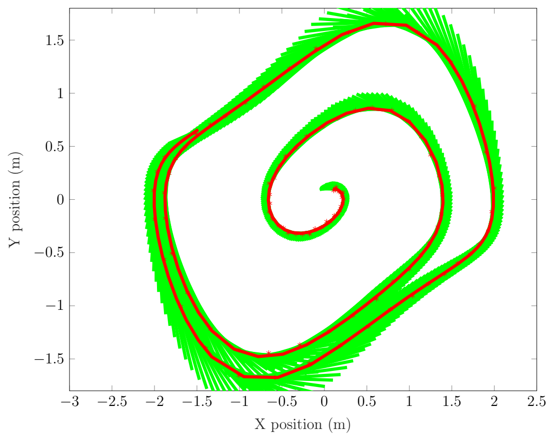


Figure 4-8: The calf length calibration results using Cerberus. For each of the four standing-up datasets, we run the estimator three times with different random initial calf lengths. Each solid curve shows the estimated calf length of each run. The horizontal dash reference line indicates 0.215m, the mean value of estimations between 6s to 11s.

Temporal Parameter

We use SWE method to estimate robot states and sensor delay time for a simulated robot as an example of explaining the temporal parameter estimation. As shown in Figure 4-9, we simulate a planar robot that moves along a certain trajectory. The robot has a position sensor and a velocity sensor that generates measurements at 4Hz and 20Hz. Without the knowledge of the robot dynamics, we just use a planar constant velocity integration model to represent its system dynamics. If the sensor data has no delay, linear KF can estimate the robot position easily. But delay in position sensor data will make the KF estimation worse.

Figure 4-10 and Figure 4-11 show that, without prior knowledge of the delay time, the RMSE (root mean square error) of the estimation using SWE is much smaller than that using conventional KF. Also, the SWE can deal with unknown and changing delays. As shown in Figure 4-11 right, even when the delay time is increasing, the estimated delay time follows the trend of the actual time. The code



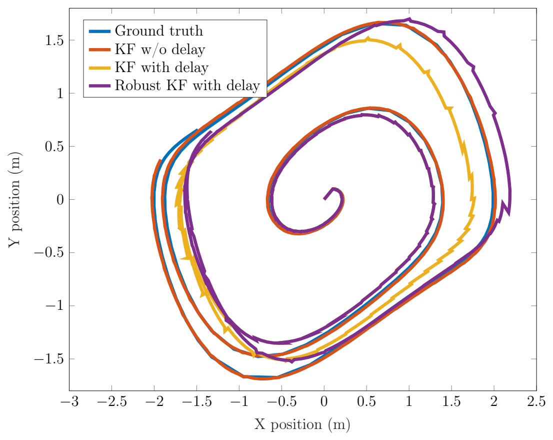


Figure 4-9: Left: simulated robot trajectory and sensor data. Right: If the position sensor has no delay, the normal KF can estimate the robot trajectory very precisely (red). However, if the position sensor data contains delays, the KF estimation result is very bad (yellow). Even if we use outlier rejection in KF (purple), the result is still not good compared to that has no delay.

implementation of this experiment is at `https://github.com/ShuoYangRobotics/MHE_delay_example`.

4.5.2 Indoor Experiments

In a lab space equipped with an OptiTrack[112] motion-capture system, the robot moves on flat ground following different paths with an average speed of 0.5m/s. We record sensor data and ground-truth positions. We then run the Cerberus on a desktop computer with Intel i7-7800X 3.50GHz CPU. The processing time is 50ms per camera frame on average, which is faster than the camera sample rate (66ms). Therefore, the state estimator should run in real time. Figure 4-12 compares the ground truth trajectory (blue) with estimated trajectory using VINS (red), VILO w/o calib (yellow), and VILO with calib (purple) in one dataset. Table 4.2 shows average performance over 10 datasets.

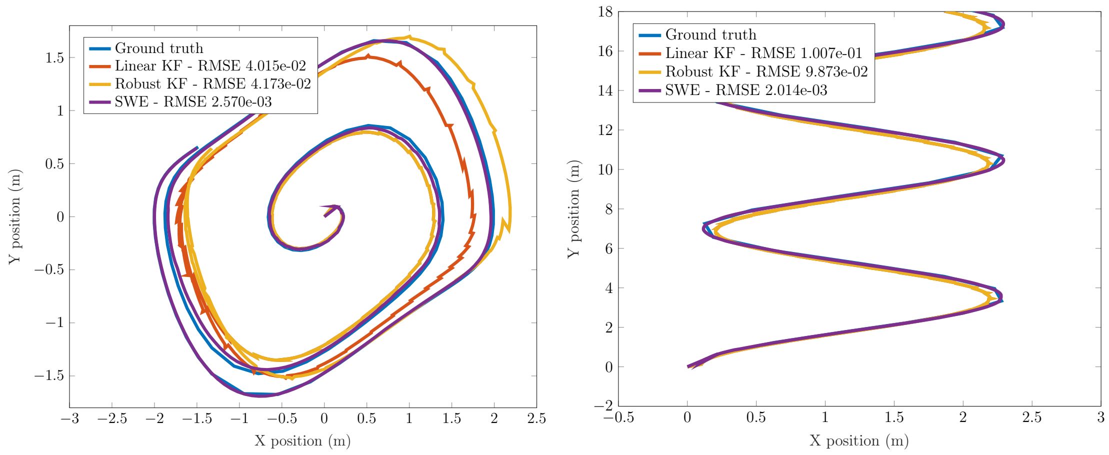


Figure 4-10: We compare different estimators' performance using the root mean square error (RMSE) by comparing the estimated position trajectory with the ground truth. Left and right figures show two different types of robot motion trajectories.

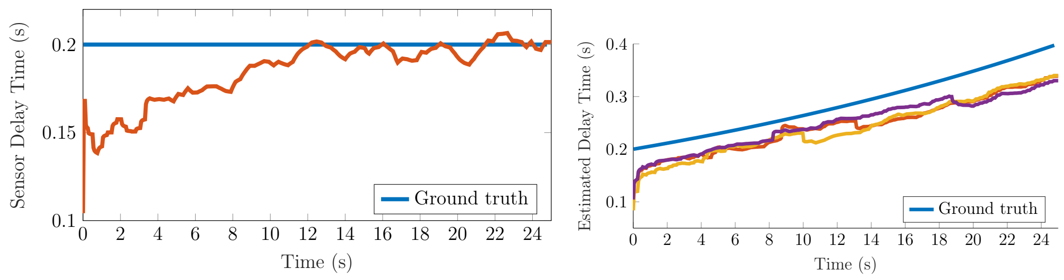


Figure 4-11: Left: when the sensor delay time is constant. The estimation converges to the correct time (0.2s). Results from three different runs with different sensor data and initial delay time values are shown. Right: if the sensor delay time is changing, the estimation keep track of the growing trend of the time. Also, the results of three runs are shown.

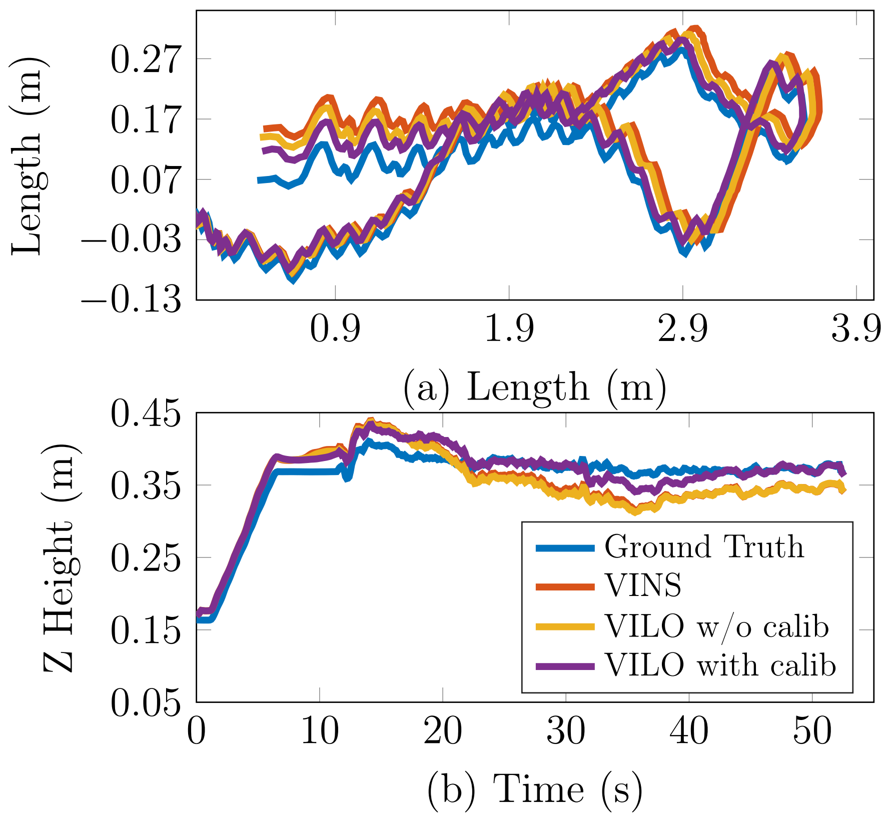


Figure 4-12: Comparing with the mocap ground truth, VILO with calib has smaller drift on all directions. The final drift of the VINS trajectory (red) is 1.73% while the drift of the VILO w/o calib trajectory (yellow) is 1.25% and that of VILO with calib (purple) is 1.13%.

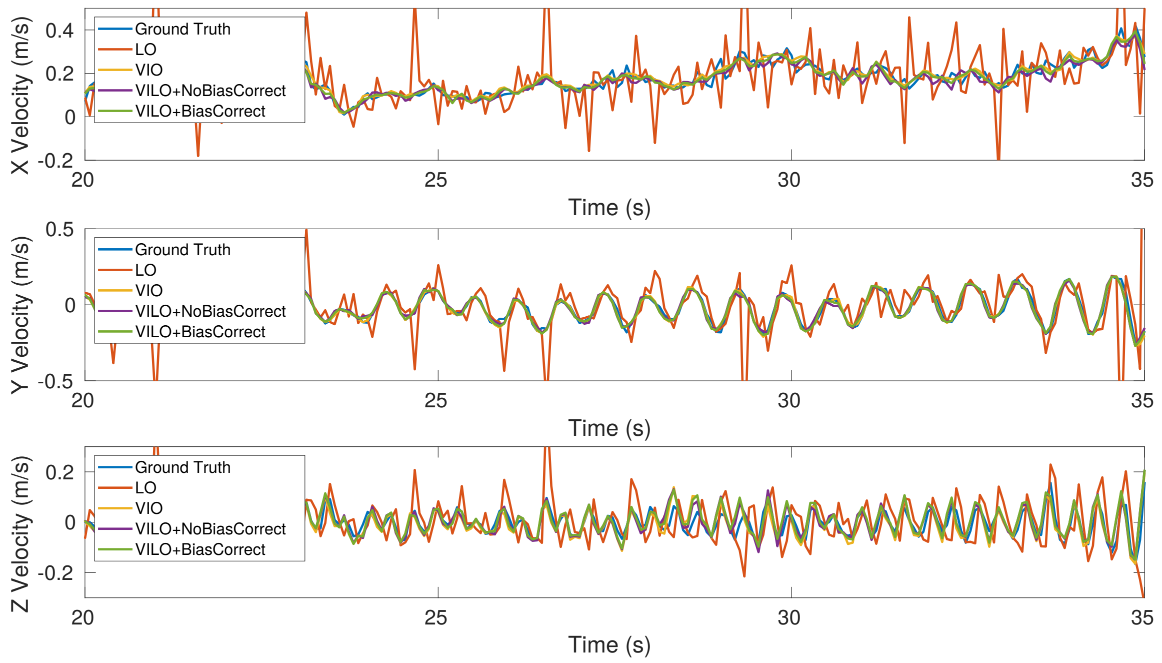


Figure 4-13: Linear velocity of the robot body on X,Y, and Z directions.

Figure 4-13 compares the velocities for one period in dataset 1. The LO velocity is calculated according to [23]. It deviates from the ground truth a lot with rooted mean-square error (RMSE) be (12.9, 16.7, 7.65) on each axis. The velocities estimated by VIO, VILO, and VILO+BiasCorrect are all pretty close to the ground truth. The RMSE of VILO+BiasCorrect velocity is (1.61, 2.83, 2.11) comparing to that of VIO (1.85, 3.56, 2.68) and that of VILO (1.79, 2.79, 2.15).

4.5.3 Outdoor Experiments

The contribution of kinematics calibration to long-term position estimation is verified in outdoor experiments. Two robots collected datasets in several outdoor environments while traveling over 1.5 km with an average velocity of 0.5 m/s. Note that our robots move at a much faster speed than prior works (for example, [76] is 0.125

Dataset	KF-PO	VIO	VILO w/o calib	Cerberus
Indoor (average 10)	6.53%	1.31%	1.02%	**0.92%**
Street	$> 10\%$	0.89%	**0.70%**	0.85%
Track	$> 10\%$	3.9%	2.6%	**0.98%**
Campus	$> 10\%$	break	3.32%	**1.65%**

Table 4.2: Hardware Experiment Final Drifts Comparison

m/s and [161] is 0.25 m/s). In each dataset, the robot moves in a large loop and we evaluate the final position estimation drift after the robot returns to the starting point. We also note that 1% drift is equivalent to $0.1m$ of the 10M Relative Translation Error (RTE) metric used in [161] and [76]. Details of datasets can be found in the open-source code base.

Figures 4-1, 4-14, and 4-15 compare the estimated trajectories for three datasets, "Track", "Campus", and "Street". Table 4.2 contains quantitative analysis of drift percentage for different datasets. The "Campus" dataset is particularly difficult because the robot runs at over 1 m/s on various indoor and outdoor terrains with different slopes. See the supplementary video for its estimation run visualization and kinematic parameter estimation result. VILO with calib outperforms all other methods across all datasets except for "Street", where both methods have very small drift values that have no statistically significant difference. Even though our datasets are longer and contain faster and more challenging dynamics, the Cerberus algorithm achieves $< 1\%$ drift on most of them and 1.65% drift on the hardest case. No prior work has achieved this level of performance.

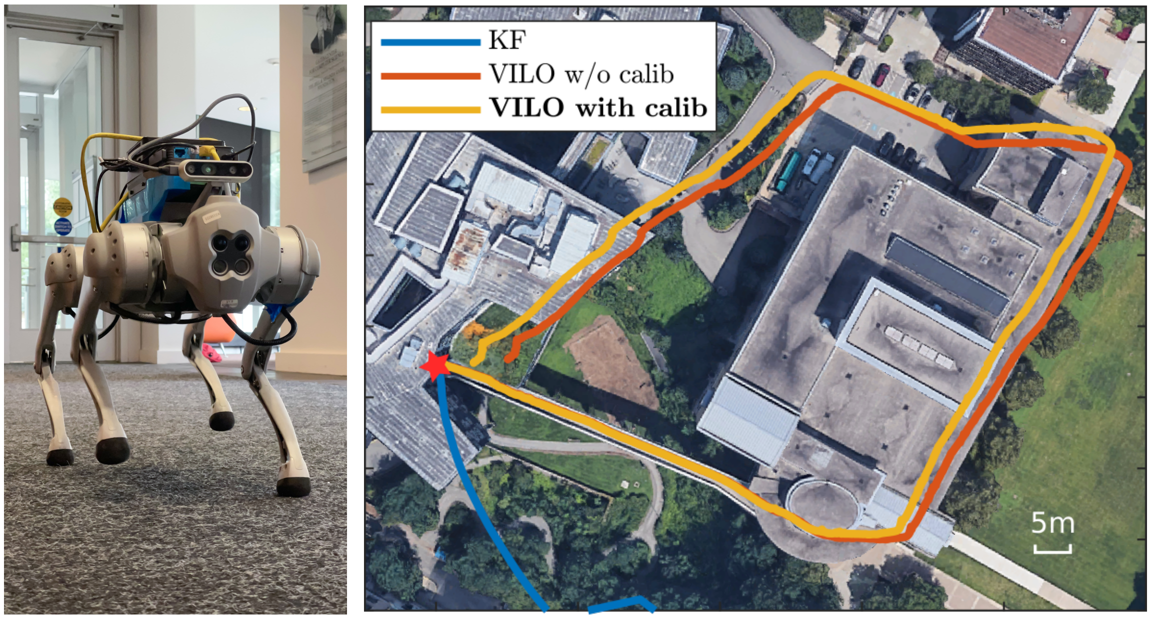


Figure 4-14: During the recording of the "Campus" dataset, the Go 1 robot ran 345 m with an average speed of 1 m/s in indoor and outdoor environments. VINS fails, so no result is shown. VILO with calibration has the smallest final position drift after returning to the starting point (red star).

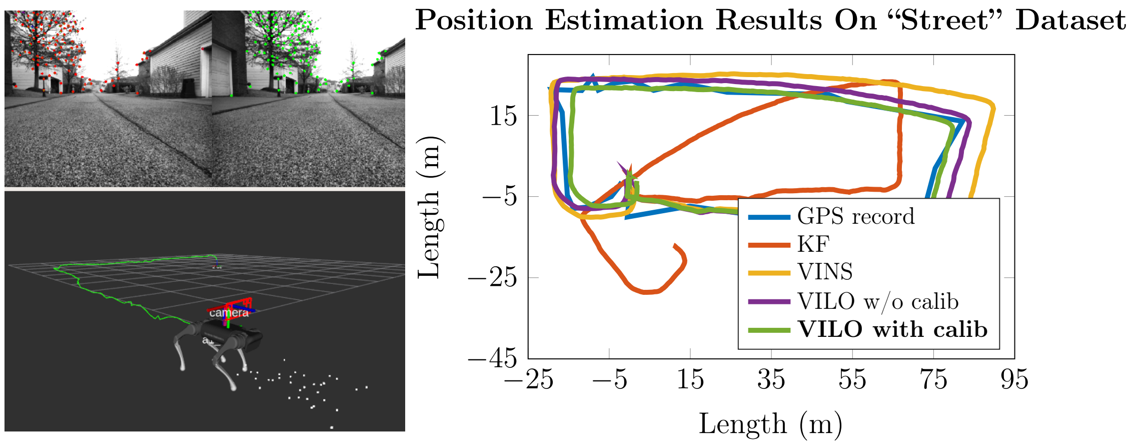


Figure 4-15: Dataset "Street" algorithm run visualization and the estimation result. GPS position reference is collected using iPhone App "Gaia GPS". The final drift of VILO with calib is 2.22m (0.85% after 260m travel) comparing to 1.84m of VILO w/o calib.

4.5.4 Robust Estimation

Since the Cerberus combines various sensor sources, the position estimation is robust against camera occlusion, foot slippage, and excessive body shakiness. The supplementary video contains more challenging scenarios that demonstrate the robustness of the estimator.

4.6 Conclusions

We have presented Cerberus, a VILO algorithm that uses kinematic calibration in contact preintegration and contact outlier rejection to improve performance. Indoor and outdoor experiments on two robots have demonstrated that our state estimator outperforms many existing methods. We believe that kinematic parameter errors, like IMU biases, should always be modeled and calibrated to achieve precise long-term estimation for legged robots. Finally, our open-sourced Cerberus package can serve as a baseline for future work.

4.7 Appendix

We present several important factors used in Ceberus' factor graph.

4.7.1 Leg-IMU Factor Derivation

As defined in Section 4.4, the robot state is $x_k = [p^w_{b_k}, q^w_{b_k}, v^w_{b_k}, b_{ak}, b_{gk}, b_{vk}, \rho_{2k}, \rho_{3k}, \rho_{4k}]$, where $p^w_{b_k}, q^w_{b_k}$ are the robot position and orientation (pose), $v^w_{b_k}$ is the robot COM velocity in world frame. b_{ak}, b_{gk} are the IMU biases. b_{vk} is the leg odometry velocity bias. ρ_{jk} is the kinematic parameter of leg j.

Preintegration Terms

$$\alpha^k_{k+1} = \int\int_{t\in[t_k,t_{k+1}]} R^{b_k}_t (a_{m_t} - b_{a_t} - n_a) dt^2 \tag{4.24}$$

$$\beta^k_{k+1} = \dot{\alpha}^k_{k+1} = \int_{t\in[t_k,t_{k+1}]} R^{b_k}_t (a_{m_t} - b_{a_t} - n_a) dt \tag{4.25}$$

$$\gamma^k_{k+1} = \int_{t\in[t_k,t_{k+1}]} \gamma^{b_k}_t \bigotimes \frac{1}{2} \begin{bmatrix} 0 \\ (\omega_{m_t} - b_{g_t} - n_g) dt \end{bmatrix} \tag{4.26}$$

$$\epsilon^k_{k+1} = \int_{t\in[t_k,t_{k+1}]} R^{b_k}_t (v_{m_t} - b_{vk} - n_{vt}) dt \tag{4.27}$$

$$\begin{aligned} v_{m_t} &= \sum_j w_j/W * v_{m_t,j} \\ &= -\sum_j w_j/W * [R_{br} J(\phi_{jt} - n_{\phi t}, \rho_j)(\dot{\phi}_{jt} - n_{\dot{\phi} t}) \\ &\quad + (\omega_{m_t} - b_{g_t} - n_g) \times [p_{br} + R_{br} g(\phi_{jt} - n_{\phi t}, \rho_j)]] \end{aligned} \tag{4.28}$$

Residual

$z_t = [a_{m_t}, \omega_{m_t}, \phi_{jt}, \dot{\phi}_{jt}, c_{jt}]$ as the sensor measurements at time t and $Z_{k,k+1} = \{z_t | t \in [t_k, t_{k+1}]\}$ be all sensor measurements between two time steps, we define the baseline IMU-leg residual r as (the dimension is $7 \times 3 + 4 = 25$)

$$r(x_k, x_{k+1}, Z_{k,k+1}) = \begin{bmatrix} R_w^{b_k}(p_{b_{k+1}}^w - p_{b_k}^w + \frac{1}{2} g^w \Delta t^2 - v_{b_k}^w \Delta t) - \alpha_{k+1}^k \\ \log(q_{b_k}^{w^{-1}} \bigotimes q_{b_{k+1}}^w \bigotimes \gamma_{k+1}^{k^{-1}}) \\ R_w^{b_k}(v_{b_{k+1}}^w + g^w \Delta t - v_{b_k}^w) - \beta_{k+1}^k \\ R_w^{b_k}(p_{b_{k+1}}^w - p_{b_k}^w) - \epsilon_{k+1}^k, \\ b_{ak+1} - b_{ak} \\ b_{gk+1} - b_{gk} \\ b_{vk+1} - b_{vk} \\ \rho_{jk+1} - \rho_{jk}, \quad \text{for } j = 1,2,3,4 \end{bmatrix} \tag{4.29}$$

Continuos Time Error Dynamics

The eqs. (4.24) to (4.27) are the results of an ideal integration process. On real system we can only get estimated result because the noise terms are unkown

$$
\begin{aligned}
\hat{\alpha}_{k+1}^{k} &= \int\int_{t\in[t_k,t_{k+1}]} \hat{R}_t^{b_k}(a_{m_t}-\hat{b}_{a_t})dt^2 \\
\hat{\beta}_{k+1}^{k} &= \dot{\hat{\alpha}}_{k+1}^{k} = \int_{t\in[t_k,t_{k+1}]} \hat{R}_t^{b_k}(a_{m_t}-\hat{b}_{a_t})dt \\
\hat{\gamma}_{k+1}^{k} &= \int_{t\in[t_k,t_{k+1}]} \hat{\gamma}_t^{b_k} \bigotimes \frac{1}{2}\begin{bmatrix} 0 \\ (\omega_{m_t}-\hat{b}_{g_t})dt \end{bmatrix} \\
\hat{\epsilon}_{k+1}^{k} &= \int_{t\in[t_k,t_{k+1}]} \hat{R}_t^{b_k}(\hat{v}_{m_t}-\hat{b}_{v_t})dt
\end{aligned}
$$

where

$$
\hat{v}_{m_t} = \sum_j w_j/W*\hat{v}_{m_t,j} = -\sum_j w_j/W*[R_{br}J(\phi_{jt},\hat{\rho}_j)(\dot{\phi}_{jt})+(\omega_{m_t}-\hat{b}_{g_t})\times[p_{br}+R_{br}g(\phi_{jt},\hat{\rho}_j)]] \tag{4.30}
$$

The $v_{m_t,j}$ can be simplified as

$$
\begin{aligned}
v_{m_t,j} =& - R_{br}J(\phi-n_\phi,\rho)(\dot{\phi}-n_{\dot{\phi}}) - (\omega_{m_t}-b_{g_t}-n_g)\times[p_{br}+R_{br}g(\phi-n_\phi,\rho)] \\
=& - R_{br}J(\phi,\rho)(\dot{\phi}-n_{\dot{\phi}}) + R_{br}(\dot{\phi}^T \circledast I_3)\frac{\partial vec(J(\phi,\rho))}{\partial\phi}n_\phi \\
& - (\omega_{m_t}-b_{g_t}-n_g)\times[p_{br}+R_{br}(g(\phi,\rho)-J(\phi,\rho)n_\phi)] \\
=& - R_{br}J(\phi,\rho)\dot{\phi} - (\omega_{m_t}-b_{g_t})\times[p_{br}+R_{br}g(\phi,\rho)] \\
& - \lfloor p_{br}+R_{br}g(\phi,\rho)\rfloor^\times n_g + R_{br}J(\phi,\rho)n_{\dot{\phi}} \\
& + R_{br}(\dot{\phi}^T \circledast I_3)\frac{\partial vec(J(\phi,\rho))}{\partial\phi}n_\phi + \lfloor\omega_{m_t}-b_{g_t}\rfloor^\times R_{br}J(\phi,\rho)n_\phi \\
=& - R_{br}J(\phi,\rho)\dot{\phi} - (\omega_{m_t}-b_{g_t})\times[p_{br}+R_{br}g(\phi,\rho)] + n_{lt,j}
\end{aligned}
$$

where

$$n_{lt,j} = -\lfloor p_{br}+R_{br}g(\phi,\rho)\rfloor^{\times}n_g+R_{br}J(\phi,\rho)n_{\dot{\phi}}+R_{br}(\dot{\phi}^T\widehat{\otimes}I_3)\frac{\partial vec(J(\phi,\rho))}{\partial\phi}n_{\phi}+\lfloor\omega_{m_t}-b_{g_t}\rfloor^{\times}R_{br}J(\phi,\rho)n_{\phi}$$

Define

$$\kappa_1 = (\dot{\phi}^T \widehat{\otimes} I_3)\frac{\partial vec(J(\phi,\hat{\rho}))}{\partial\phi}$$

$$\kappa_2 = (\dot{\phi}^T \widehat{\otimes} I_3)\frac{\partial vec(J(\phi,\hat{\rho}))}{\partial\hat{\rho}}$$

Also notice $b_{g_t} = \hat{b}_{g_t} + \delta b_{gt}$, $\rho = \hat{\rho} + \delta\rho$, and define $\hat{g}_{bj} = p_{br} + R_{br}g_j(\phi_j,\hat{\rho})$ and $g_{bj} = p_{br} + R_{br}g_j(\phi_j,\rho)$ $(g_{bj} = \hat{g}_{bj} + R_{br}\frac{\partial g(\phi,\hat{\rho})}{\partial\hat{\rho}}\delta\rho)$, so

$$v_{m_t,j} = \hat{v}_{m_t,j} - R_{br}\kappa_2\delta\rho - \lfloor\hat{g}_{bj}\rfloor^{\times}\delta b_g - \lfloor\omega_{m_t} - \hat{b}_{g_t}\rfloor^{\times}R_{br}\frac{\partial g(\phi,\hat{\rho})}{\partial\hat{\rho}}\delta\rho + n_{lt,j}$$

Define

$$\kappa_3 = R_{br}\kappa_2 + \lfloor\omega_{m_t} - \hat{b}_{g_t}\rfloor^{\times}R_{br}\frac{\partial g(\phi,\hat{\rho})}{\partial\hat{\rho}}$$

So we have

$$v_{m_t} = \hat{v}_{m_t} + \sum_j \frac{w_j}{W}n_{lt,j} - \{\sum_j \frac{w_j}{W}\lfloor g_{bj}\rfloor^{\times}\}\delta b_{gt} \tag{4.31}$$

$$- \{\sum_j \frac{w_j}{W}\kappa_3\delta\rho_j\} \tag{4.32}$$

We notice

$$R_t^{b_k} = \hat{R}_t^{b_k}(I + \lfloor\delta\theta\rfloor^{\times})$$

Derive error dynamics

$$\begin{aligned}
\delta\dot{\epsilon}_{k+1}^{k} =&\dot{\epsilon}_{k+1}^{k} - \dot{\hat{\epsilon}}_{k+1}^{k} = R_t^{b_k}(v_{m_t} - b_{vk} - n_{vt}) - \hat{R}_t^{b_k}(\hat{v}_{m_t} - \hat{b}_{v_t}) \qquad (4.33)\\
=&\hat{R}_t^{b_k}(I + \lfloor\delta\theta\rfloor^{\times})v_{m_t} - \hat{R}_t^{b_k}(I + \lfloor\delta\theta\rfloor^{\times})b_{vk} - \hat{R}_t^{b_k}(I + \lfloor\delta\theta\rfloor^{\times})n_{vt} - \hat{R}_t^{b_k}\hat{v}_{m_t} + \hat{R}_t^{b_k}\hat{b}_{v_t}\\
=&\hat{R}_t^{b_k}v_{m_t} - \hat{R}_t^{b_k}\lfloor v_{m_t}\rfloor^{\times}\delta\theta - \hat{R}_t^{b_k}b_{vk} + \hat{R}_t^{b_k}\lfloor b_{vk}\rfloor^{\times}\delta\theta - \hat{R}_t^{b_k}n_{vt}\\
&- \hat{R}_t^{b_k}\hat{v}_{m_t} + \hat{R}_t^{b_k}\hat{b}_{v_t}\\
=&\hat{R}_t^{b_k}(\sum_j \frac{w_j}{W}n_{lt,j} - \{\sum_j \frac{w_j}{W}\lfloor g_{bj}\rfloor^{\times}\}\delta b_{gt} - \{\sum_j \frac{w_j}{W}\kappa_3\delta\rho_j\}) \qquad (4.34)\\
&-\hat{R}_t^{b_k}\lfloor\hat{v}_{m_t} - \hat{b}_{vk}\rfloor^{\times}\delta\theta - \hat{R}_t^{b_k}\delta b_{vk} - \hat{R}_t^{b_k}n_{vt}
\end{aligned}$$

Define

$$\kappa_4 = R_{br}\kappa_1 + \lfloor\omega_{m_t} - b_{g_t}\rfloor^{\times}R_{br}J(\phi,\rho)$$

then

$$n_{lt,j} = -\lfloor\hat{g}_{bj}\rfloor^{\times}n_g + R_{br}J(\phi,\rho)n_{\dot{\phi}} + \kappa_4 n_{\phi}$$

We can drop the term $\hat{R}_t^{b_k}$ for all noise term.

$$\begin{aligned}
\delta\dot{\epsilon}_{k+1}^{k} =&\sum_j \frac{w_j}{W}n_{lt,j} - \hat{R}_t^{b_k}\{\sum_j \frac{w_j}{W}\lfloor g_{bj}\rfloor^{\times}\}\delta b_{gt} - \{\hat{R}_t^{b_k}\sum_j \frac{w_j}{W}\kappa_3\delta\rho_j\} \qquad (4.35)\\
&-\hat{R}_t^{b_k}\lfloor\hat{v}_{m_t} - \hat{b}_{vk}\rfloor^{\times}\delta\theta - \hat{R}_t^{b_k}\delta b_{vk} - n_{vt}\\
=&-\hat{R}_t^{b_k}\lfloor\hat{v}_{m_t} - \hat{b}_{vk}\rfloor^{\times}\delta\theta - \hat{R}_t^{b_k}\{\sum_j \frac{w_j}{W}\lfloor g_{bj}\rfloor^{\times}\}\delta b_{gt}\\
&- \hat{R}_t^{b_k}\delta b_{vk} - \{\hat{R}_t^{b_k}\sum_j \frac{w_j}{W}\kappa_3\delta\rho_j\}\\
&- \sum_j \frac{w_j}{W}\lfloor\hat{g}_{bj}\rfloor^{\times}n_g + \sum_j \frac{w_j}{W}R_{br}J(\phi,\rho)n_{\dot{\phi}} + \sum_j \frac{w_j}{W}\kappa_4 n_{\phi} - n_{vt} \qquad (4.36)
\end{aligned}$$

4.7.2 Midpoint Method Discretize Dynamics

To improve numerical conditions of dynamics terms, we use the midpoint method to rewrite discretized dynamics in the previous section as follows.

$\delta\theta$

$$\delta\theta_{i+1} = \kappa_9 \delta\theta_i - \frac{\delta t}{2} n_{gi} - \frac{\delta t}{2} n_{gi+1} - \delta t \delta b_{gt} \tag{4.37}$$

$$\kappa_9 = [I - \lfloor \frac{\omega_{mi+1} + \omega_{mi}}{2} - \hat{b}_{gk} \rfloor^{\times} \delta t]$$

$\delta\beta$

$$\delta\beta_{i+1} = \delta\beta_i + \kappa_5 \delta\theta_i - \frac{\delta t}{2}(\hat{R}^{b_k}_{i+1} + \hat{R}^{b_k}_i)\delta b_{at} + \kappa_6 \delta b_{gt} - \frac{\delta t}{2}(\hat{R}^{b_k}_i n_{ai} + \hat{R}^{b_k}_{i+1} n_{ai+1}) + \kappa_7 (n_{gi} + n_{gi+1}) \tag{4.38}$$

where

$$\kappa_5 = -\frac{\delta t}{2}\hat{R}^{b_k}_i \lfloor a_{m_i} - \hat{b}_{a_t} \rfloor^{\times} - \frac{\delta t}{2}\hat{R}^{b_k}_{i+1} \lfloor a_{m_{i+1}} - \hat{b}_{a_t} \rfloor^{\times} [I - \lfloor \frac{\omega_{mi+1} + \omega_{mi}}{2} - \hat{b}_{gk} \rfloor^{\times} \delta t]$$

$$\kappa_6 = \frac{\delta t^2}{2} \hat{R}^{b_k}_{i+1} \lfloor a_{m_{i+1}} - \hat{b}_{a_t} \rfloor^{\times}$$

$$\kappa_7 = \frac{\delta t^2}{4} \hat{R}^{b_k}_{i+1} \lfloor a_{m_{i+1}} - \hat{b}_{a_t} \rfloor^{\times}$$

$\delta\alpha$

$$\delta\alpha_{i+1} = \delta\alpha_i + \delta t\delta\beta_i + \frac{\delta t}{2}\kappa_5\delta\theta_i - \frac{\delta t^2}{4}(\hat{R}^{b_k}_{i+1} + \hat{R}^{b_k}_i)\delta b_{at} + \frac{\delta t}{2}\kappa_6\delta b_{gt} - \frac{\delta t^2}{4}(\hat{R}^{b_k}_i n_{ai} + \hat{R}^{b_k}_{i+1} n_{ai+1}) + \frac{\delta t}{2}\kappa_7(n_{gi} + n_{gi+1})$$

$\delta\epsilon$

Finally, the discretization of (4.36) is

$$\begin{aligned}
&\delta\epsilon_{i+1,j} = \delta\epsilon_{i,j} + \\
&\kappa_8\delta\theta_i + \frac{\delta t^2}{4}\hat{R}^{b_k,i+1}_t\lfloor\hat{v}_{m_t,i+1} - \hat{b}_{vk}\rfloor^{\times}(n_{gi} + n_{gi+1}) + \frac{\delta t^2}{2}\hat{R}^{b_k,i+1}_t\lfloor\hat{v}_{m_t,i+1} - \hat{b}_{vk}\rfloor^{\times}\delta b_{gt} \\
&- \frac{\delta t}{2}\hat{R}^{b_k,i}_t\{\sum_j \frac{w_{j,i}}{W}\lfloor g_{bj,i}\rfloor^{\times}\}\delta b_{gt} - \frac{\delta t}{2}\hat{R}^{b_k,i}_t\delta b_{vk} - \frac{\delta t}{2}\{\hat{R}^{b_k,i}_t\sum_j \frac{w_{j,i}}{W}\kappa_{3,i}\delta\rho_j\} \\
&- \frac{\delta t}{2}\hat{R}^{b_k,i+1}_t\{\sum_j \frac{w_{j,i+1}}{W}\lfloor g_{bj,i+1}\rfloor^{\times}\}\delta b_{gt} - \frac{\delta t}{2}\hat{R}^{b_k,i+1}_t\delta b_{vk} - \frac{\delta t}{2}\{\hat{R}^{b_k,i+1}_t\sum_j \frac{w_{j,i+1}}{W}\kappa_{3,i+1}\delta\rho_j\} \\
&- \frac{\delta t}{2}\sum_j \frac{w_{j,i}}{W}\lfloor\hat{g}_{bj,i}\rfloor^{\times}n_g + \frac{\delta t}{2}\sum_j \frac{w_{j,i}}{W}R_{br}J(\phi,\rho)n_{\dot{\phi}} + \frac{\delta t}{2}\sum_j \frac{w_{j,i}}{W}\kappa_{4,i}n_{\phi} - \frac{\delta t}{2}n_{vt,i} \\
&- \frac{\delta t}{2}\sum_j \frac{w_{j,i+1}}{W}\lfloor\hat{g}_{bj,i+1}\rfloor^{\times}n_g + \frac{\delta t}{2}\sum_j \frac{w_{j,i+1}}{W}R_{br}J(\phi,\rho)n_{\dot{\phi}} + \frac{\delta t}{2}\sum_j \frac{w_{j,i+1}}{W}\kappa_{4,i+1}n_{\phi} - \frac{\delta t}{2}n_{vt,i+1}
\end{aligned}$$

$$\kappa_8 = -\frac{\delta t}{2}\hat{R}^{b_k}_i\lfloor\hat{v}_{m_i} - \hat{b}_{vk}\rfloor^{\times} - \frac{\delta t}{2}\hat{R}^{b_k}_{i+1}\lfloor\hat{v}_{m_{i+1}} - \hat{b}_{vk}\rfloor^{\times}[I - \lfloor\frac{\omega_{mi+1} + \omega_{mi}}{2} - \hat{b}_{gk}\rfloor^{\times}\delta t]$$

4.7.3 Error dynamics equation

The discretized error dynamics is

$$\delta E^k_{t+\delta t} = F_i \delta E^k_t + G_i n_t =$$

$$\begin{bmatrix} I & \frac{\delta t}{2}\kappa_5 & \delta t & 0 & -\frac{\delta t^2}{4}(\hat{R}^{b_k}_{i+1} + \hat{R}^{b_k}_{i}) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \kappa_9 & 0 & 0 & 0 & -\delta t & 0 & 0 & 0 & 0 & 0 \\ 0 & \kappa_5 & I & 0 & -\frac{\delta t}{2}(\hat{R}^{b_k}_{i+1} + \hat{R}^{b_k}_{i}) & \kappa_6 & 0 & 0 & 0 & 0 & 0 \\ 0 & \kappa_8 & 0 & I & 0 & \kappa_{10} & -\frac{\delta t}{2}(\hat{R}^{b_k}_{i+1} + \hat{R}^{b_k}_{i}) & -\frac{\delta t}{2}(\hat{R}^{b_k}_{i+1}\frac{w_{1,i+1}}{W}\kappa_{3,i+1} + \hat{R}^{b_k}_{i}\frac{w_{1,i}}{W}\kappa_{3,i}) & * & * & * \\ 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \delta\alpha^k_t \\ \delta\theta^k_t \\ \delta\beta^k_t \\ \delta\epsilon^k_t \\ \delta b_{a_t} \\ \delta b_{g_t} \\ \delta b_{v_t} \\ \delta\rho_1 \\ \delta\rho_2 \\ \delta\rho_3 \\ \delta\rho_4 \end{bmatrix} +$$

$$\begin{bmatrix} -\frac{\delta t^2}{4}\hat{R}^{b_k}_{i} & \frac{\delta t}{2}\kappa_7 & -\frac{\delta t^2}{4}\hat{R}^{b_k}_{i+1} & \frac{\delta t}{2}\kappa_7 & 0 & 0 & 0 \\ 0 & -\frac{\delta t}{2} & 0 & -\frac{\delta t}{2} & 0 & 0 & 0 \\ -\frac{\delta t}{2}\hat{R}^{b_k}_{i} & \kappa_7 & -\frac{\delta t}{2}\hat{R}^{b_k}_{i+1} & \kappa_7 & 0 & 0 & 0 \\ 0 & \kappa_{11} - \frac{\delta t}{2}\hat{R}^{b_k}_{i}\sum_j \frac{w_j}{W}\lfloor g_{bij}\rfloor^{\times} & 0 & \kappa_{11} - \frac{\delta t}{2}\hat{R}^{b_k}_{i+1}\sum_j \frac{w_j}{W}\lfloor g_{bi+1j}\rfloor^{\times} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \dots$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{\delta t}{2}\hat{R}_i^{b_k}\sum_j \frac{w_j}{W}\kappa_{4,i} & \frac{\delta t}{2}\hat{R}_{i+1}^{b_k}\sum_j \frac{w_j}{W}\kappa_{4,i+1} & \frac{\delta t}{2}\hat{R}_i^{b_k}R_{br}\sum_j \frac{w_j}{W}J(\phi_{ij}) & \frac{\delta t}{2}\hat{R}_{i+1}^{b_k}R_{br}\sum_j \frac{w_j}{W}J(\phi_{i+1j}) & -\delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \delta t \end{bmatrix} \begin{bmatrix} n_{ai} \\ n_{gi} \\ n_{ai+1} \\ n_{gi+1} \\ n_{b_{at}} \\ n_{b_{gt}} \\ n_{b_{vt}} \\ n_{\phi i} \\ n_{\phi i+1} \\ n_{\dot{\phi} i} \\ n_{\dot{\phi} i+1} \\ n_{vi} \\ n_{\rho_1} \\ n_{\rho_2} \\ n_{\rho_3} \\ n_{\rho_4} \end{bmatrix}$$

$$\kappa_{10} = \frac{\delta t^2}{2}\hat{R}_{i+1}^{b_k}\lfloor \hat{v}_{m_{i+1}} - \hat{b}_{vk}\rfloor^{\times} - \frac{\delta t}{2}(\hat{R}_i^{b_k}\sum_j \frac{w_j}{W}\lfloor g_{bij}\rfloor^{\times} + \hat{R}_{i+1}^{b_k}\sum_j \frac{w_j}{W}\lfloor g_{bi+1j}\rfloor^{\times})$$

$$\kappa_{11} = \frac{\delta t^2}{4}\hat{R}_{i+1}^{b_k}\lfloor \hat{v}_{m_{i+1}} - \hat{b}_{vk}\rfloor^{\times}$$

4.7.4 Residual Jacobian

In this section we list Jacobians of (4.29) w.r.t to $x_k = [p^w_{b_k}, q^w_{b_k}, v^w_{b_k}, b_{ak}, b_{gk}, b_{vk}, \rho_{1k}, \rho_{2k}, \rho_{3k}, \rho_{4k}]$ and

$x_{k+1} = [p^w_{b_{k+1}}, q^w_{b_{k+1}}, v^w_{b_{k+1}}, b_{ak+1}, b_{gk+1}, b_{vk+1}, \rho_{1k+1}, \rho_{2k+1}, \rho_{3k+1}, \rho_{4k+1}]$ as follows:

$$J[0]^{25\times7} = \begin{bmatrix} \frac{\partial r}{\partial p^w_{b_k}} & \frac{\partial r}{\partial q^w_{b_k}} \end{bmatrix} = \begin{bmatrix} -R^{b_k}_w & \lfloor R^{b_k}_w (p^w_{b_{k+1}} - p^w_{b_k} + \frac{1}{2} g^w \Delta t^2 - v^w_{b_k} \Delta t) \rfloor^\times \\ 0 & -[\mathcal{L}[{q^w_{b_{k+1}}}^{-1} \bigotimes q^w_{b_k}] \mathcal{R}(\gamma^k_{k+1})]_{3\times3} \\ 0 & \lfloor R^{b_k}_w (v^w_{b_{k+1}} + g^w \Delta t - v^w_{b_k}) \rfloor^\times \\ -R^{b_k}_w & \lfloor R^{b_k}_w (p^w_{b_{k+1}} - p^w_{b_k}) \rfloor^\times \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{4.39}$$

$$J[1]^{25\times 9} = \begin{bmatrix} \frac{\partial r}{\partial v_{b_k}^w} & \frac{\partial r}{\partial b_{ak}} & \frac{\partial r}{\partial b_{gk}} \end{bmatrix} = \begin{bmatrix} -R_w^{b_k}\Delta t & -J_{b_a}^{\alpha} & -J_{b_g}^{\alpha} \\ 0 & 0 & -\mathcal{L}[{q_{b_{k+1}}^w}^{-1} \bigotimes q_{b_k}^w \bigotimes \gamma_{k+1}^k]_{3\times 3} J_{b_g}^{\gamma} \\ -R_w^{b_k} & -J_{b_a}^{\beta} & -J_{b_g}^{\beta} \\ 0 & 0 & -J_{b_g}^{\epsilon} \\ 0 & -I & 0 \\ 0 & 0 & -I \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$J[2]^{25\times 7} = \begin{bmatrix} \frac{\partial r}{\partial b_{vk}} & \frac{\partial r}{\partial \rho_1} & \frac{\partial r}{\partial \rho_2} & \frac{\partial r}{\partial \rho_3} & \frac{\partial r}{\partial \rho_4} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -J_{b_{vk}}^{\epsilon} & -J_{\rho_1 k}^{\epsilon} & -J_{\rho_2 k}^{\epsilon} & -J_{\rho_3 k}^{\epsilon} & -J_{\rho_4 k}^{\epsilon} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -I & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$J[3]^{25\times 7} = \begin{bmatrix} \frac{\partial r}{\partial p^w_{b_{k+1}}} & \frac{\partial r}{\partial q^w_{b_{k+1}}} \end{bmatrix} = \begin{bmatrix} R^{b_k}_w & 0 \\ 0 & \mathcal{L}[{\gamma^k_{k+1}}^{-1} \bigotimes {q^w_{b_k}}^{-1} \bigotimes q^w_{b_{k+1}}]_{3\times 3} \\ 0 & 0 \\ R^{b_k}_w & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{4.40}$$

$$J[4]^{25\times 9} = \begin{bmatrix} \frac{\partial r}{\partial v^w_{b_{k+1}}} & \frac{\partial r}{\partial b_{k+1}} & \frac{\partial r}{\partial b_{k+1}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ R^{b_k}_w & 0 & 0 \\ 0 & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

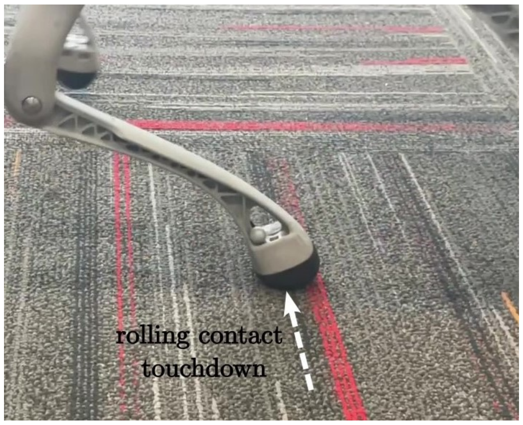
$$J[5]^{25\times 7} = \begin{bmatrix} \frac{\partial r}{\partial b_{vk+1}} & \frac{\partial r}{\partial \rho_1} & \frac{\partial r}{\partial \rho_2} & \frac{\partial r}{\partial \rho_3} & \frac{\partial r}{\partial \rho_4} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ I & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Chapter 5

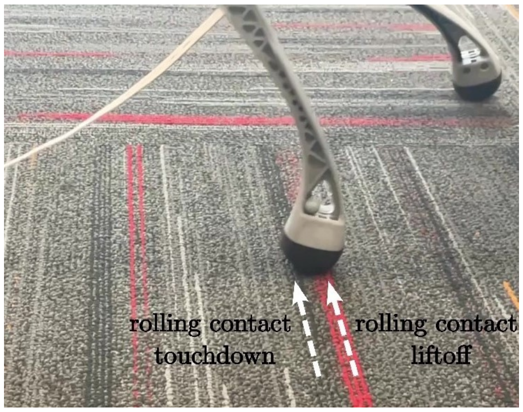
Multi-IMU Proprioceptive Odometry

In Chapter 3 and Chapter 4 we mainly leveraged a model where we treat foot deformations as leg kinematic parameter changes. This model become less effective if we adopt a foot design that has harder surface. In this way the foot is less likely to deform but more easily to roll on the ground. But we decided to take this design and add more sensors to better capture the rolling motion. In addition to the conventional sensor set used in standard Proprioceptive Odometry, including one body Inertial Measurement Unit (IMU) and joint encoders, we attach an additional IMU to each calf link of the robot, just above the foot.

We modify the Kalman filter formulation to fuse data from all sensors to estimate the robot's body and foot positions in the world frame. By using the additional IMUs, the filter can reliably determine foot contact modes and detect foot slips without tactile or pressure-based foot contact sensors. This sensing solution is validated in various hardware experiments, which confirm that it can reduce position drift by nearly an order of magnitude compared to conventional approaches, with only a very modest increase in hardware and computational costs.



(a) Touchdown

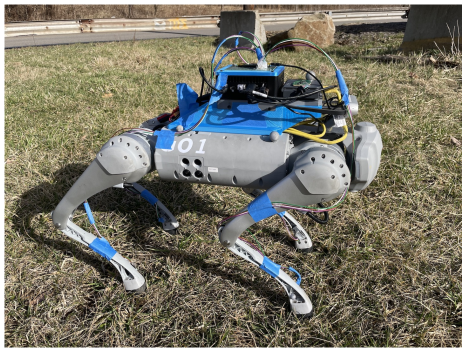


(b) Liftoff

Figure 5-1: The point foot has obvious rolling contact during stance.

5.1 Introduction

Our goal is to develop a PO solution that can achieve low position drift while requiring minimal additional hardware and computational resources. To achieve this, we propose Multi-IMU Proprioceptive Odometry (MIPO), a sensing solution that uses multiple IMUs and leg-joint encoders to significantly improve upon KF-based PO methods that only use a single IMU. Compared to visual sensors such as cameras and lidar, an IMU has much lower cost, energy consumption, and physical size. Therefore, we are able to add IMUs to the robot's calf links near the feet with minimal impact on the overall design and cost. To fuse data from additional IMUs, We include world-frame foot positions and velocities in the state of an Extended Kalman Filter (EKF) and design a prediction model to update foot velocities using foot IMU data. More importantly, we also leverage the foot IMUs to detect contact and foot slip in a novel measurement model, overcoming a fundamental error source in conventional PO, where feet are assumed to have zero velocity relative to the ground while in contact [14, 19, 55, 163].



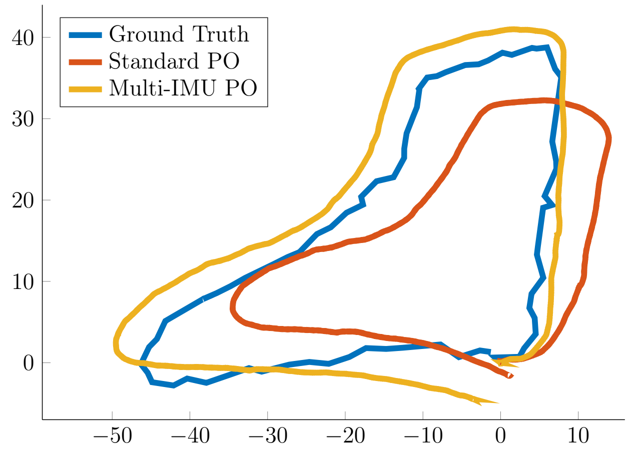


Figure 5-2: **Left:** A Unitree Go1 robot equipped with foot IMUs. **Right:** Position estimates while walking over a 160m loop trajectory: Standard Proprioceptive Odometry (PO) has an average XY position drift of 18.4%, while our Multi-IMU PO solution achieves 2.59% average drift, a significant improvement achieved with minimal additional hardware and computational cost.

Our state-estimation approach is validated on a quadrupedal robot in a variety of conditions, and we make comparisons to several existing baseline methods. Our specific contributions include:

- A novel low-cost multi-IMU sensing solution for proprioceptive odometry on legged robots.
- An Extended Kalman Filter (EKF) with a prediction model that uses foot IMU data to update foot velocities and a measurement model that uses foot IMU data to determine foot contact modes and slip.
- Ablation studies and comparison experiments on hardware demonstrating significant reductions in position drift.

The remainder of this chapter is organized as follows: Section 5.2 provides a review of related work. Section 5.3 presents the basics of legged robot state estimation.

Section 5.4 presents our main technical contributions. Section 5.5 compares our solution with baseline methods across a variety of experiments. An ablation study is also provided. Finally, Section 5.6 concludes the chapter.

5.2 Related Work

Deploying a suite of IMUs, rather than just one, to enhance estimation performance is common in several domains, such as pedestrian navigation systems [5], augmented and virtual-reality applications [66], and general VIO [40]. In legged robot state estimation, a PO solution is proposed in [162] using a network of IMUs installed on different parts of a humanoid robot for better joint-velocity sensing. A similar approach uses redundant accelerometers to improve joint information estimation [83]. A multi-IMU Kalman filter is developed for an exoskeleton [41]. It has also been shown that multiple IMUs can help quantify link flexibilities [152].

5.3 Background

Key building blocks of our work, such as quaternion representation, leg odometry, and Kalman filtering, have been extensively discussed in the previous chapters. Additional to standard Kalman filtering, we further leverage an idea that uses "factor" like measurement model in EKF.

5.3.1 Implicit Measurement Model & Quaternion Measurement

An intuitive understanding of Kalman filter is, we would expect the innovation to be zero after seeing the sensor data. If the innovation is not zero, we transform the nonzero residual in an educated way and use it to correct the estimated state. This leads to a measurement model as "implicit measurement". We can write Eqn (3.33) as

$$0 = h'(x_t, z_t) + v = h(x_t) - z_t + v \tag{5.1}$$

and define a new residual

$$y'_t = h'(\hat{x}_t, z_t) = -s_t(\hat{x}_t). \tag{5.2}$$

Doing Taylor expansion at $x_{t-1} = \hat{x}_{t-1} + \delta x_{t-1}$ of Eqn. (5.1) shows that

$$s_t(\hat{x}_t) = \left.\frac{\partial h'}{\partial x}\right|_{\hat{x}_t} \delta x_t + v \tag{5.3}$$

We notice that if $h'(x_t, z_t) = h(x_t) - z_t$, then $\partial h/\partial x = \partial h'/\partial x$. And nothing changes for ESKF steps. It seems unnecessary and confusing to do so. However, this implicit measurement is very important for complex nonlinear measurements.

For example, if the legged robot estimator state involves body orientation q and foot orientation q_L, the robot kinematics model observes their differences as

$$q_{FK} = q^{-1} \otimes q_L. \tag{5.4}$$

In this case, the notion of subtraction $(z_t - h(x_t))$ is ill-defined because we cannot directly subtract two quaternions. On the other hand, the implicit measurement idea is more straightforward. Given an estimated state $\hat{x}_{t-1}$ and a sensor observation q_y, we calculate an innovation term

$$y_t = \mathrm{Log}(q_y^{-1} \otimes \hat{q}^{-1} \otimes \hat{q}_L) \tag{5.5}$$

and then use s_t in Eqn. (5.2) and Eqn. (5.3) for ESKF update steps.

5.4 Technical Approach

The standard single-IMU PO method presented in Section 3.3.11 has two fundamental limitations: First, bias estimation, which usually must be included if IMU hardware quality is not good, would drift when the robot stands still with colinear contacts [19]. It will lead to wrong orientation estimation. Second, the zero-velocity assumption used when deriving (3.47) is seldom true on hardware. The LO velocity always underestimates the true velocity if the foot is rolling during contact, as shown in Figure 5-3. Both of these limitations can be addressed by adding additional IMUs to the robot's feet. We refer to this sensor-and-algorithm combination as Multi-IMU Proprioceptive Odometry (Multi IMU PO). The hardware sensor setup will be explained in Section 5.5, this section focuses on the algorithm. Compared to the previous publication [164], we extend the filter state to include foot orientations and use a simplified quaternion small-rotation approximation technique to achieve fast and precise computation. We also add a new measurement model to keep the bias from drifting.

We define the multi-IMU estimator state as $\boldsymbol{x} = [\boldsymbol{p}; \boldsymbol{v}; \boldsymbol{q}; \boldsymbol{b}_a; \boldsymbol{b}_g; [\boldsymbol{s}; \dot{\boldsymbol{s}}; \boldsymbol{q}_f; \boldsymbol{b}_s; \boldsymbol{b}_t]_j]$ where $\boldsymbol{p}$, $\boldsymbol{v}$, $\boldsymbol{q}$, and $\boldsymbol{s}$ are the same as that defined in Section 3.3.11. Additionally, we also estimate body IMU accelerometer biases $\boldsymbol{b}_a \in \mathbb{R}^3$ and gyroscope biases $\boldsymbol{b}_g \in \mathbb{R}^3$, foot velocity $\dot{\boldsymbol{s}} \in \mathbb{R}^3$ and foot orientation $\boldsymbol{q}_f \in SO(3)$ as a quaternion, and foot IMU accelerometer biases $\boldsymbol{b}_s \in \mathbb{R}^3$ and foot IMU gyroscope biases $\boldsymbol{b}_q \in \mathbb{R}^3$. The subscript j means that the second part of the state has E copies, one for each leg. The estimator state dimension increases considerably (for quadrupeds, the dimension is 80, while the single PO has a dimension of 28 even with biases added). But this dimension increase is necessary. As we will show, including foot orientation is important for keeping the entire system observable, and foot velocity is critical for low drift body position estimation. To keep a low computation load, we would like to calculate analytical Jacobians and leverage Jacobian sparsity as much as possible.

For brevity, again in the subsequent discussion, we consider $E = 1$ and drop leg index j. We assume all sensors are synchronized and produce data at the same frequency. In addition to sensor measurements introduced in Section 3.3.11, we also get $\boldsymbol{a}_f$ and $\boldsymbol{\omega}_f$, the foot acceleration reading and the foot angular velocity reading from an IMU installed on the foot in the foot frame.

5.4.1 EKF Process Model

We use $\hat{\boldsymbol{x}}'$ to represent $\hat{\boldsymbol{x}}_{k+1|k}$ in order to simplify notation. The process model is changed from (3.45) to

$$\hat{\boldsymbol{x}}' = \begin{bmatrix} \hat{\boldsymbol{p}}' \\ \hat{\boldsymbol{v}}' \\ \hat{\boldsymbol{q}}' \\ \hat{\boldsymbol{b}}'_a \\ \hat{\boldsymbol{b}}'_g \\ \hat{\boldsymbol{s}}' \\ \dot{\hat{\boldsymbol{s}}}' \\ \hat{\boldsymbol{q}}'_f \\ \hat{\boldsymbol{b}}'_s \\ \hat{\boldsymbol{b}}'_t \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{p}} + \Delta t \hat{\boldsymbol{v}} \\ \hat{\boldsymbol{v}} + \Delta t (R(\hat{\boldsymbol{q}})(\boldsymbol{a}_b - \hat{\boldsymbol{b}}_a) - \boldsymbol{g}^w) \\ M(\hat{\boldsymbol{q}})\mathrm{Exp}((\boldsymbol{\omega}_b - \hat{\boldsymbol{b}}_g)\Delta t) \\ \hat{\boldsymbol{b}}_a \\ \hat{\boldsymbol{b}}_g \\ \hat{\boldsymbol{s}} + \Delta t \dot{\hat{\boldsymbol{s}}} \\ \dot{\hat{\boldsymbol{s}}} + \Delta t (R(\hat{\boldsymbol{q}}_f)(\boldsymbol{a}_f - \hat{\boldsymbol{b}}_s) - \boldsymbol{g}^w) \\ M(\hat{\boldsymbol{q}}_f)\mathrm{Exp}((\boldsymbol{\omega}_f - \hat{\boldsymbol{b}}_t)\Delta t) \\ \hat{\boldsymbol{b}}_s \\ \hat{\boldsymbol{b}}_t \end{bmatrix}. \tag{5.6}$$

Compared to (3.45), although the dimension increases, the update equations are structurally identical and self-contained for body and foot, which are two rigid links with installed IMUs. Therefore, the Jacobian is fairly sparse. Moreover, the noise covariance of this process model does not depend on the contact flag because the dynamics are continuous. The foot velocity noise covariance is constant as long as the foot IMUs do not saturate their readings, which can be guaranteed by proper controller design and IMU hardware selection.

5.4.2 The Pivoting Contact Model

A key observation that significantly enhances the estimation of body velocity is the relationship between the foot's contact point and its velocity. For any legged robot, whether it has point feet or flat feet, the foot pivots around the contact point at any given moment. This occurs irrespective of whether the contact foot is stationary or in motion, the type of terrain, or any deformation that might occur on the foot surface or the ground. Thus, the contact foot's linear velocity should equal the cross product of the foot angular velocity vector $\boldsymbol{\omega}$ and a pivoting vector $\boldsymbol{d}$ pointing from the contact point to the foot frame origin. For robots with flat feet, the contact point is at the center of pressure (CoP), which can be measured directly by foot contact sensors [41]. However, for robots equipped with spherical "point" feet, directly measuring the contact location on each foot is difficult. We discovered that a suitable approximation can be achieved by using the point on the foot's surface that aligns with a line drawn from the body frame origin to the foot frame origin. This approximation is effective because the body frame also pivots around the contact point. This specific point and the corresponding equations are depicted in Figure 5-3, in which

$$\dot{\hat{\boldsymbol{s}}} = \boldsymbol{\omega}_p(\hat{\boldsymbol{x}}, \boldsymbol{\omega}_f) \times \boldsymbol{d}_p(\hat{\boldsymbol{x}}, \boldsymbol{\alpha}), \text{and} \tag{5.7}$$

$$\boldsymbol{\omega}_p(\hat{\boldsymbol{x}}, \boldsymbol{\omega}_f) = R(\hat{\boldsymbol{q}}_f)(\boldsymbol{\omega}_f - \hat{\boldsymbol{b}}_t), \tag{5.8}$$

$$\boldsymbol{d}_p(\hat{\boldsymbol{x}}, \boldsymbol{\alpha}) = -d_0 \cdot \boldsymbol{n}/\|\boldsymbol{n}\|. \tag{5.9}$$

In the calculation, d_0 is the distance between the foot center and the foot surface. If the foot does not deform much during locomotion, this distance can be treated as a constant and measured from the CAD model, which is the case in our experiments; if the foot deforms a lot, d_0 can be obtained using data-driven methods or a calibration method such as that described in [163]. $\boldsymbol{n} = R(\hat{\boldsymbol{q}})\boldsymbol{g}(\boldsymbol{\alpha})$ is the contact normal vector expressed in the world frame. This "pivoting model" captures the rolling contact better than [163].

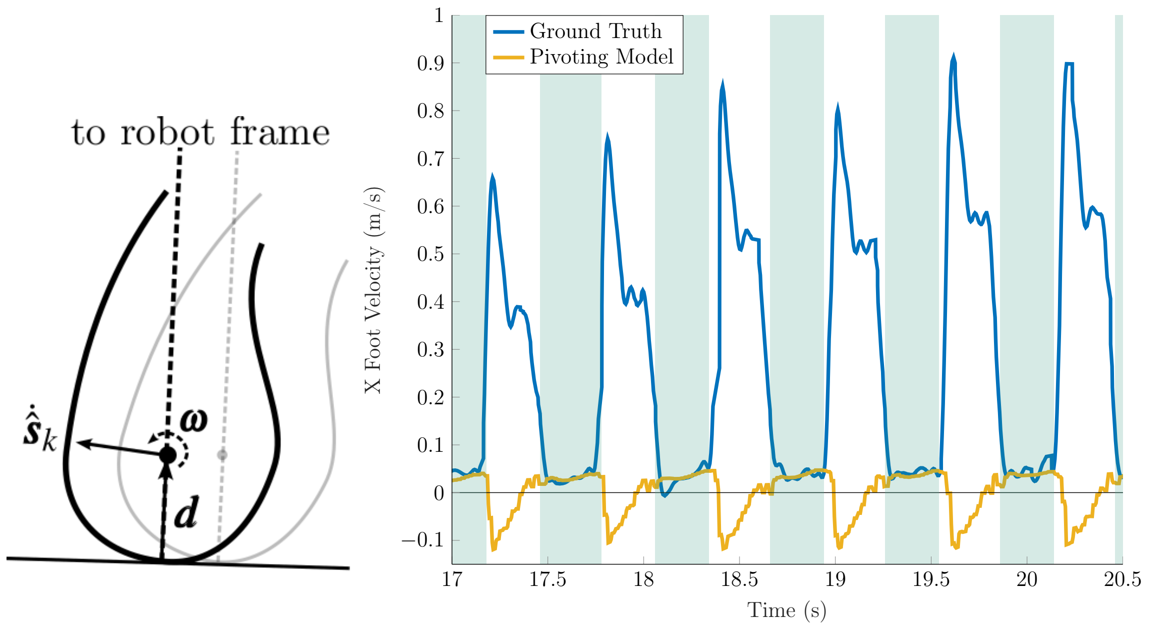


Figure 5-3: **Left:** Illustration of the pivoting model for a foot that has rolling contact with the ground. The estimated foot velocity $\dot{\boldsymbol{s}}_k$ depends on $\boldsymbol{\omega}$ and $\boldsymbol{d}$, as defined in (5.8) and (5.9). **Right:** Comparison of ground-truth foot velocity captured with a motion capture system to estimation using the pivoting model (5.12). When the foot has non-zero rolling velocity during contacts (shaded regions), the pivoting model agrees with the ground truth velocity very well while the zero-velocity model treats foot velocity as zero.

5.4.3 EKF Measurement Model

We define a measurement residual $y(\hat{\boldsymbol{x}}, \boldsymbol{\alpha}, \boldsymbol{a}_f, \boldsymbol{\omega}_f) = \bar{\boldsymbol{z}}(\boldsymbol{\alpha}) - \boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{\alpha}, \boldsymbol{a}_f, \boldsymbol{\omega}_f)$, where

$$\bar{\boldsymbol{z}}(\boldsymbol{\alpha}) = \left[\boldsymbol{g}(\boldsymbol{\alpha}); \mathbf{0}; \mathbf{0}; \mathbf{0}; \boldsymbol{a}_f\right] \tag{5.10}$$

and

$$\boldsymbol{h}(\hat{\boldsymbol{x}}, \boldsymbol{\alpha}, \boldsymbol{a}_f, \boldsymbol{\omega}_f) = \tag{5.11}$$

$$\begin{bmatrix} R(\hat{\boldsymbol{q}})^T(\hat{\boldsymbol{s}} - \hat{\boldsymbol{p}}) \\ \mathrm{Log}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1} \otimes \hat{\boldsymbol{q}}^{-1} \otimes \hat{\boldsymbol{q}}_f) \\ J(\boldsymbol{\alpha})\dot{\boldsymbol{\alpha}} - \lfloor \boldsymbol{\omega}_b - \hat{\boldsymbol{b}}_g \rfloor^{\times} \boldsymbol{g}(\boldsymbol{\alpha}) - R(\hat{\boldsymbol{q}})^T(\hat{\boldsymbol{v}} - \dot{\hat{\boldsymbol{s}}}) \\ \dot{\hat{\boldsymbol{s}}} - \boldsymbol{\omega}_p(\hat{\boldsymbol{x}}, \boldsymbol{\omega}_f) \times \boldsymbol{d}_p(\hat{\boldsymbol{x}}, \boldsymbol{\alpha}) \\ R(\hat{\boldsymbol{q}}_f)^T \boldsymbol{g}^w + \hat{\boldsymbol{b}}_s \end{bmatrix}. \tag{5.12}$$

Several terms in the sensor measurement vector $\bar{\boldsymbol{z}}$ are $\boldsymbol{0}$ and the sensor data appears in the measurement model. We call these "implicit measurements."

The first term of the residual y is the same as in (3.48). The second term uses forward kinematics to observe the rotational difference between the foot orientation and the body orientation, which is an implicit measurement because of the special structure of the quaternion. The term $\boldsymbol{q}(\boldsymbol{\alpha})$ represents a rotation between the foot frame and the body frame calculated from the forward kinematics [91]. The third term comes from (3.44) without assuming $\boldsymbol{v}_f^w = 0$. In contrast to (3.48), these three terms related to leg forward kinematics do not have varying measurement noise since they stay valid across foot contact switches.

We refer to the fourth term, which is based on the pivoting model, as a pivoting measurement. The pivoting measurement is only valid when a foot is in contact with the ground and its residual value is drastically different in stance and swing phases, which allows us to use a better technique to change its contribution to the estimation.

The last term commonly appears in the IMU complementary filter [95, 150], where the IMU accelerometer reading is used for observing the gravity direction. This measurement model assumes the gravitational field dominates the low frequency response of the accelerometer. This assumption is often not valid for IMUs on the body of legged robots where horizontal linear accelerations are significant. However, when IMUs are on their feet and contact can be reliably detected, this measurement model becomes accurate and can contribute to robot orientation estimation significantly.

Since the yaw orientation of the robot is never observable [19], whenever a better yaw observation is available — for example, a vision-based method from VIO where global landmarks can be observed — we can introduce it into the EKF. It can be formulated in different ways, one of the simplest observations is calculating the yaw angle from the orientation quaternion as [3]

$$\mathrm{YAW}(\boldsymbol{q}) = \arctan(2(q_x q_y + q_w q_z), 1 - 2(q_y q_y + q_z q_z)) \tag{5.13}$$

5.4.4 Foot Contact and Slip Detection

The last two terms in the measurement model (5.12) are only accurate when the robot foot has contact with the ground. Instead of doing the covariance-scaling heuristic (3.46) for them, we use a statistical test based on the Mahalanobis norm [17]:

$$\|\boldsymbol{y}\|_S < \sigma, \tag{5.14}$$

where σ is a hyperparameter, $\boldsymbol{y} = \dot{\hat{\boldsymbol{s}}} - \boldsymbol{\omega}_p \times \boldsymbol{d}_p$ as defined in (5.12), and S is its corresponding covariance matrix calculated in the Kalman filter Algorithm (1). If (5.14) is satisfied, we treat the foot as being in non-slipping contact and include the corresponding pivoting measurement in the update (5.12).

Due to substantial differences in foot velocity between swing and stance phases (see Figure 5-3 unshaded regions), (5.14) can effectively distinguish foot phases without a dedicated contact sensor. Notably, analogous mechanisms for the zero-velocity model have been employed in previous works such as [19] and [75]. In Section 5.5, we further demonstrate the importance of the statistical test in the pivoting model through an ablation study.

5.4.5 Analytical Jacobian

It is well known that when the EKF state includes quaternions, the filter should operate on the error state [93, 163], in which orientation error is parameterized by a three-parameter rotation representation. Let the true state of the robot be $\boldsymbol{x}$ and the estimated state be $\hat{\boldsymbol{x}}$, then we denote the error state as $\delta\boldsymbol{x} = [\delta\boldsymbol{p}; \delta\boldsymbol{v}; \delta\boldsymbol{\theta}; \delta\boldsymbol{b}_a; \delta\boldsymbol{b}_g; \delta\boldsymbol{s}; \delta\dot{\boldsymbol{s}}; \delta\boldsymbol{\theta}_f; \delta\boldsymbol{b}_s; \delta\boldsymbol{b}_t]$, in which Euclidean vectors are in the form of true state minus estimated state, such as $\delta\boldsymbol{p} = \boldsymbol{p} - \hat{\boldsymbol{p}}$. But the error rotation is $\delta\boldsymbol{\theta} = \mathrm{Log}(\hat{\boldsymbol{q}}^{-1} \otimes \boldsymbol{q}) \in \mathbb{R}^3$ as discussed in Section 3.3.5. In this section, we give the analytical form of the Multi-IMU PO's EKF model Jacobians F and H introduced in the Kalman filter Algorithm (1), with detailed derivations of key terms in the Appendix.

The process Jacobian F is a sparse block matrix

$$F(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{\omega}, \boldsymbol{a}_f, \boldsymbol{\omega}_f) = \begin{bmatrix} F_s(\boldsymbol{q}, \boldsymbol{b}_a, \boldsymbol{b}_g, \boldsymbol{a}, \boldsymbol{\omega}) & 0 \\ 0 & F_s(\boldsymbol{q}_f, \boldsymbol{b}_s, \boldsymbol{b}_t, \boldsymbol{a}_f, \boldsymbol{\omega}_f) \end{bmatrix} \tag{5.15}$$

where

$$F_s(\boldsymbol{q}, \boldsymbol{b}_a, \boldsymbol{b}_g, \boldsymbol{a}, \boldsymbol{\omega}) = \tag{5.16}$$
$$\begin{bmatrix} I & I\Delta t & 0 & 0 & 0 \\ 0 & I & -R(\boldsymbol{q})\lfloor(\boldsymbol{a}-\boldsymbol{b}_a)\Delta t\rfloor^{\times} & 0 & -R(\boldsymbol{q})\Delta t \\ 0 & 0 & I - \lfloor(\boldsymbol{\omega}-\boldsymbol{b}_g)\Delta t\rfloor^{\times} & -I\Delta t & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix}.$$

We write the measurement Jacobian H as

$$H(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{a}_f, \boldsymbol{\omega}_f) = [H_b \;\; H_f], \tag{5.17}$$

where the first block H_b is Jacobian concerning robot body-related state variables while the second block relates to foot H_f. Let R be short for $R(\boldsymbol{q})$ and $R_f = R(\boldsymbol{q}_f)$, then we have:

$$H_b = \tag{5.18}$$
$$\begin{bmatrix} -R^T & 0 & \lfloor R^T(\boldsymbol{s}-\boldsymbol{p})\rfloor^{\times} & 0 & 0 \\ 0 & 0 & -\mathcal{L}(\boldsymbol{q}_f^{-1}\otimes\boldsymbol{q})\mathcal{R}(q(\boldsymbol{\alpha}))_3 & 0 & 0 \\ 0 & -R^T & \lfloor R^T(\dot{\boldsymbol{s}}-\boldsymbol{v})\rfloor^{\times} & 0 & -\lfloor g(\boldsymbol{\alpha})\rfloor^{\times} \\ 0 & 0 & \lfloor\boldsymbol{\omega}_p\rfloor^{\times}\lfloor\boldsymbol{d}_p\rfloor^{\times} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and $H_f =$ (5.19)

$$\begin{bmatrix} R^T & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathcal{L}(q(\boldsymbol{\alpha}) \otimes \boldsymbol{q}^{-1} \otimes \boldsymbol{q}_f)_3 & 0 & 0 \\ 0 & R^T & 0 & 0 & 0 \\ 0 & I & -\lfloor \boldsymbol{d}_p \rfloor^\times \lfloor \boldsymbol{\omega}_p \rfloor^\times & 0 & -\lfloor \boldsymbol{\omega}_p \rfloor^\times R_f \\ 0 & 0 & \lfloor R_f^T \boldsymbol{g}^w \rfloor^\times & I & 0 \end{bmatrix}.$$

In above equations, $\mathcal{L}()_3$ means the bottom right 3-by-3 sub-matrix of Equation (3.8) and $\mathcal{R}()_3$ is the bottom right 3-by-3 sub-matrix of Equation (3.9). These Jacobians are sparse by nature. And their elements can be calculated efficiently.

In the literature, some formulations also treat joint-angle and joint-angle velocity measurements as noise corrupted and calculate a measurement Jacobian for $\boldsymbol{\alpha}$ and $\dot{\boldsymbol{\alpha}}$ [55, 159]. We found this Jacobian can often be approximated as a constant matrix since, during locomotion, joint angles stay within small ranges so we can precalculate the Jacobian using the average joint angles as a part of Σ_w.

5.4.6 Observability Analysis

The nonlinear observability analysis of the dynamical system we have formulated is nontrivial because of the complicated measurement model. It is easier to do local observability analysis through the Jacobians. Prior work has pointed out that such analysis would lead to the conclusion that the yaw angle of the orientation is observable [19], which it is not. Intuitively, it is obvious that, even with additional IMUs, the global positions and yaw angles of the robot are still not observable. Therefore, our goal for observability analysis is to prove other state terms are observable. We can verify that for the process Jacobian (5.15) and the measurement Jacobian (5.17), the Observability Matrix [57]

$$\mathcal{O}_1 = \begin{bmatrix} H_0 \\ H_1 F_0 \\ \vdots \\ H_T F_T F_{T-1} \cdots F_0 \end{bmatrix} \tag{5.20}$$

and the local Observability Gramian [80]

$$\mathcal{O}_2 = \sum_{i=0}^{T} (F_i^T)^i H_i^T H_i (F_i)^i \tag{5.21}$$

both have rank $15 \times (1+E) - 3$, where the Jacobians are evaluated along an estimated state trajectory $[x_0, x_1, \ldots, x_T]$ with enough time steps. 15 is the dimension of state variables related to either the robot body link or a foot link. E is the total number of legs. This result aligns with the conclusion that global position is not observable and the yaw angle is rendered observable locally. Since the most important job of the state estimator is to provide pitch and roll orientation as well as body pose rate (linear velocity and angular velocity) to the controller, be sure that these key variables are observable is enough to ensure overall system stability.

The rank condition is satisfied for any state and measurement value, even when the robot stands still with zero angular velocity. The standard PO's observability matrix would drop rank when body angular velocity is $\mathbf{0}$, while the Multi-IMU PO will keep the rank condition consistent.

	Frequency	Solve Time	Median Drift
Standard PO	200Hz	1.81ms	11.05%
MIPO	200Hz	2.50ms	2.61%

Table 5.1: Performance Summary

5.4.7 Cramér-Rao Lower Bound

The Cramér-Rao Lower Bound (CRLB) is a metric that indicates the theoretical lower bound of the covariance of the output of a state estimator [144]. By computing and contrasting the CRLBs of various filters, we can validate our hypothesis that Multi-IMU PO is expected to outperform the conventional PO method due to a richer set of sensor information.

According to [144], the CRLB for a Kalman filter is defined as a matrix P^* that at any time instance k, we have estimation covariance $P_k \geq P^*$. This lower bound can be computed recursively along with the EKF procedure as:

$$(P_k^*)^{-1} = (F_k P_{k-1}^* F_k^T)^{-1} + H_k^T \Sigma_w^{-1} H_k \tag{5.22}$$

where P_k^* is the lower bound calculated at time k. F_k, H_k, and Σ_w are defined in Algorithm 1. These matrices will have different values for Multi-IMU PO and the standard PO, but since the states of both filters contain the position of the robot, we can examine the covariance of position in their respective CRLBs, as will be shown in Section 5.5.

5.5 Experiments

We conducted a series of indoor and outdoor experiments on hardware to compare our MIPO with the baseline method, standard single-IMU PO [14], and Cerberus

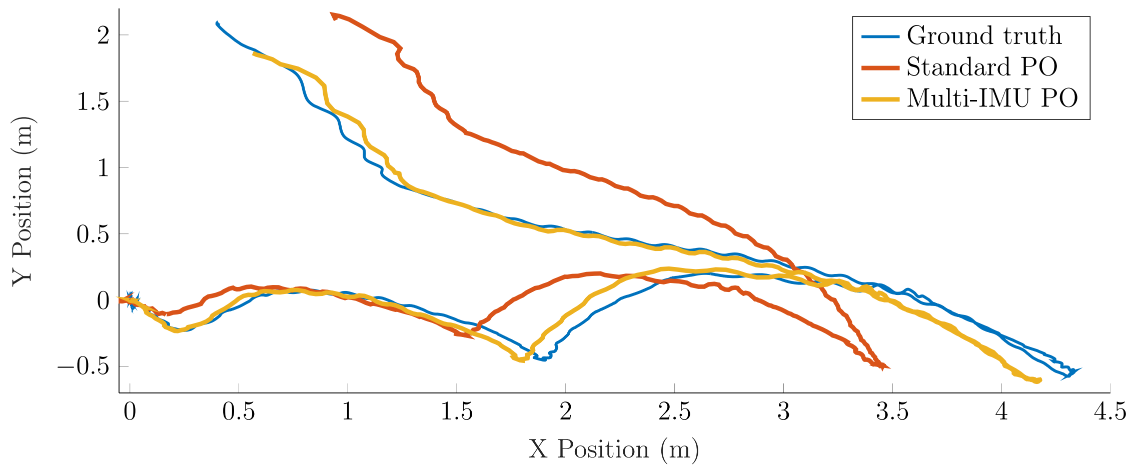


Figure 5-4: The estimated XY position trajectory comparison of the standard PO and MIPO. The total trajectory length is 10.5m. The standard PO estimation drifts 11.39% on average, and its maximum RSE is 1.04m. While the result of MIPO has 2.31% average drift and 0.25m maximum RSE.

Visual-Inertial-Leg Odometry (VILO) [165]. We focused on evaluating their position estimation performance, especially XY position. Common evaluation metrics used in the state-estimation literature include Root-Square-Error (RSE) and drift percentage. RSE is defined as the Euclidean distance between the ground truth and the estimated position at a given time instance. Drift percentage is defined as the ratio of the RSE to the total distance traveled. The data and code used to generate results are available on Github[1].

5.5.1 Sensor Hardware Design

The MIPO hardware does not significantly alter the form factor of the Go1 robot. The Go1 robot's built-in proprietary MEMS IMU sensor and joint motor encoder data can be obtained at 200Hz via Ethernet. The robot also has four pressure

[1]`https://github.com/ShuoYangRobotics/Multi-IMU-Proprioceptive-Odometry`

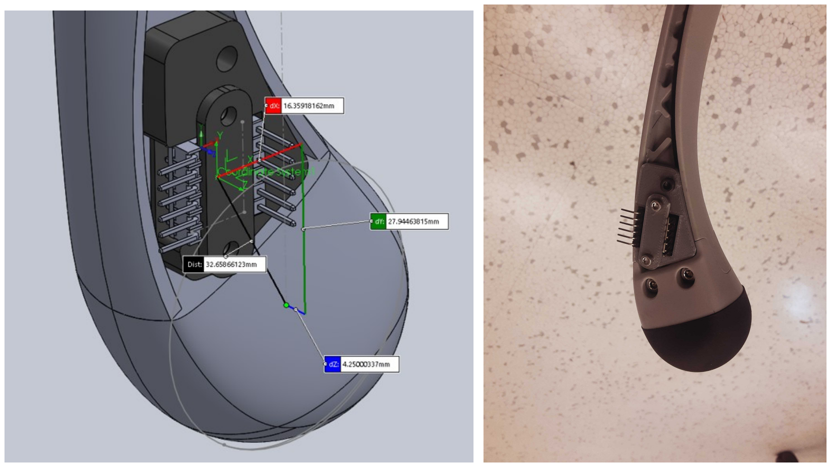


Figure 5-5: **Left:** The CAD design of foot IMU mount. The offset from IMU to foot center is measured from the CAD drawing. **Right:** The IMU installation on one foot of a Unitree Go1 robot.

contact sensors on the feet that can be thresholded to obtain binary contact flags for the baseline method. Our MIPO, however, does not use contact sensor data. Instead, four MPU9250 IMUs are installed on each foot, right above the foot, as shown in Fig. 5-5. Since it is difficult to directly install an IMU at the exact foot center, we transform foot IMU outputs to the foot center frame using the transformation measured in the CAD model. An Arduino Teensy board communicates with the foot IMUs, acquiring their outputs at 200Hz. An Intel NUC mini computer finally collects sensor data from the Go1 robot and the Teensy board to run the MIPO algorithm. The computer also runs a nonlinear predictive control locomotion controller [42].

5.5.2 Position Estimation Evaluation

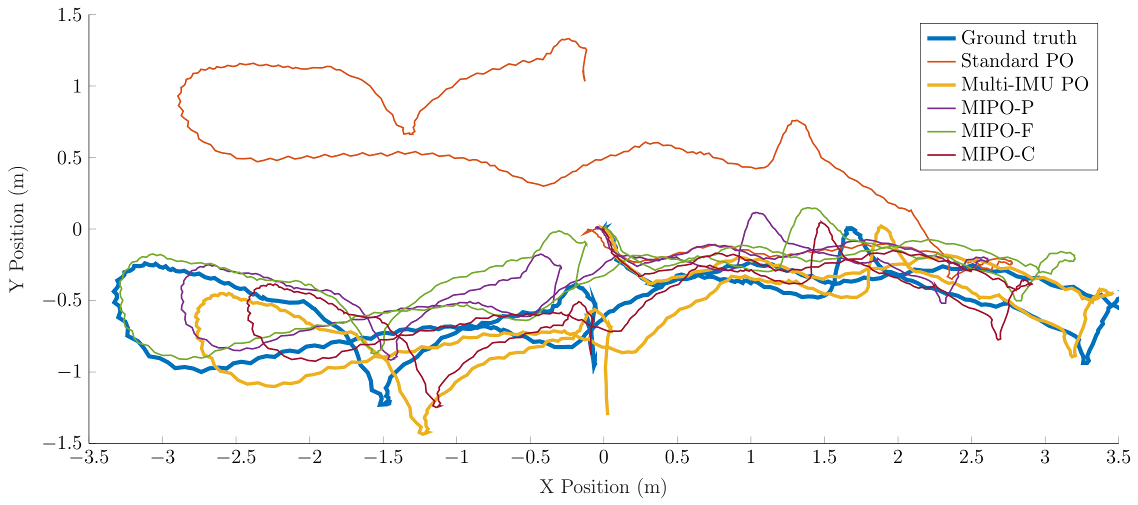
We first compare the results of MIPO against standard PO in an indoor environment. The robot operates in a lab space equipped with a motion capture (MoCap) system which provides the ground truth pose. The robot uses the trotting gait or flying trotting gait (with a full airborne phase between leg switching) to locomote in arbitrary directions with a speed of 0.4-1.0 m/s on flat ground. Fig. 5-4 compares the estimated position trajectories. Table 5.1 compares their per-loop solve times and average drifts across five different datasets. MIPO has larger state dimensions so the computation time is slightly longer, but the time is still within the budget (5 ms). Moreover, the drift percentage is significantly lower.

In outdoor environments where neither MoCap or reliable GPS signals are available, we use the Cerberus [165] VILO algorithm as ground truth to compare MIPO and standard PO. As can be seen in Fig. 5-2, MIPO achieves a much smaller position drift than standard PO after traveling 160m over varrying terrains.

Lastly, we compare the CRLBs of Multi-IMU PO and standard PO. As discussed in Section 5.4.7, the CRLB represents the lowest possible uncertainty of a state estimator. For any given trajectory, we can calculate the CRLB recursively as Equation (5.22) from an initial covariance. In Figure 5-8, it is shown that Multi-IMU PO has a much smaller CRLB than standard PO. This result aligns with our intuition and previous experiment results that Multi-IMU has better estimation performance.

5.5.3 Multi-IMU PO Orientation Estimation

Robot state estimation can be challenging in some special cases during locomotion. For example, when a legged robot performs fast in-place rotation with rapidly changing angular accelerations using the trotting gait, the robot could suffer from several error sources. First, robot feet deform more than usual, making leg kinematics less accurate. Second, the foot-supporting line gets close to roll axis so the roll angle becomes less observable. Lastly, angular acceleration makes IMU have elevated noise.



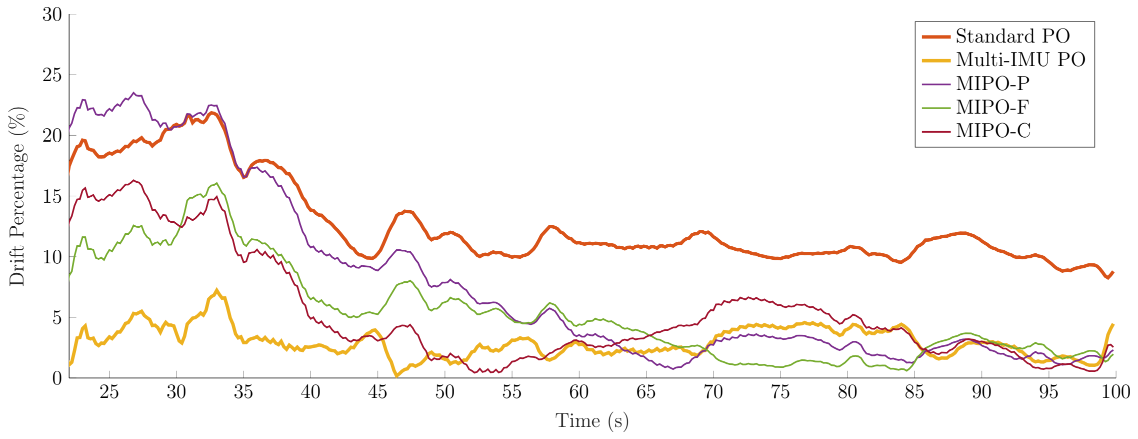


Figure 5-6: **Top:** XY position trajectory estimation results of the standard PO, MIPO, and three MIPO variants used in the ablation study. The total trajectory length is 21.5m. **Bottom:** The drift percentages over time of all methods. MIPO has the smallest (4.21%), followed by MIPO-F (7.75%), MIPO-C (11.3%), MIPO-P (12.04%), and the standard PO (16.87%).

These problems can be effectively solved by including foot orientations to the filter and adding gravity direction observation as shown in Equation 5.10. We observe that even during fast in-place rotation and foot-rolling contacts, the IMUs on feet capture gravity direction reasonably well during contacts. We show its contribution in Figure 5-7. In this experiment, the operator arbitrarily gave the robot a fast fast-changing yaw rate command to rotate the robot. We collect sensor data and offline run different estimation methods to get body orientation estimations. Unlike in the previous section, we do not use Mocap ground truth orientation to correct any estimation. Then we convert all estimated orientations to Euler Angle and compare pitch and roll angles with MoCap ground truth. In the plots, the "Complementary Filter" is an implementation of [95], where the accelerometer and user command are used to estimate gravity direction to correct orientation drift. Among all methods, the Multi-IMU PO has the smallest orientation estimation error. In the most extreme case (time 13.6s, where the robot did a fast stop and start of rotation), ground truth roll orientation is 2.49deg, while standard PO estimates it to be 1.74deg and the Multi-IMU PO estimation is 2.11deg. The Multi-IMU's orientation is 50.6% more accurate at this time and has 30% less estimation error on average.

5.5.4 Ablation Study

In this section, we study the individual contribution of the pivoting model and the statistical test introduced in Section 5.4.3. We create three algorithm variants: 1) MIPO-P, where the fourth term in (5.12) is $\dot{\boldsymbol{s}}_k$ instead of $\dot{\boldsymbol{s}}_k - \boldsymbol{\omega} \times \boldsymbol{d}$. We vary its measurement noise according to the contact flag but do not perform the statistical test (5.14). In this way, MIPO-P is essentially a standard PO that uses MIPO's process model. 2) MIPO-F, which adopts the pivoting model from MIPO, but differs

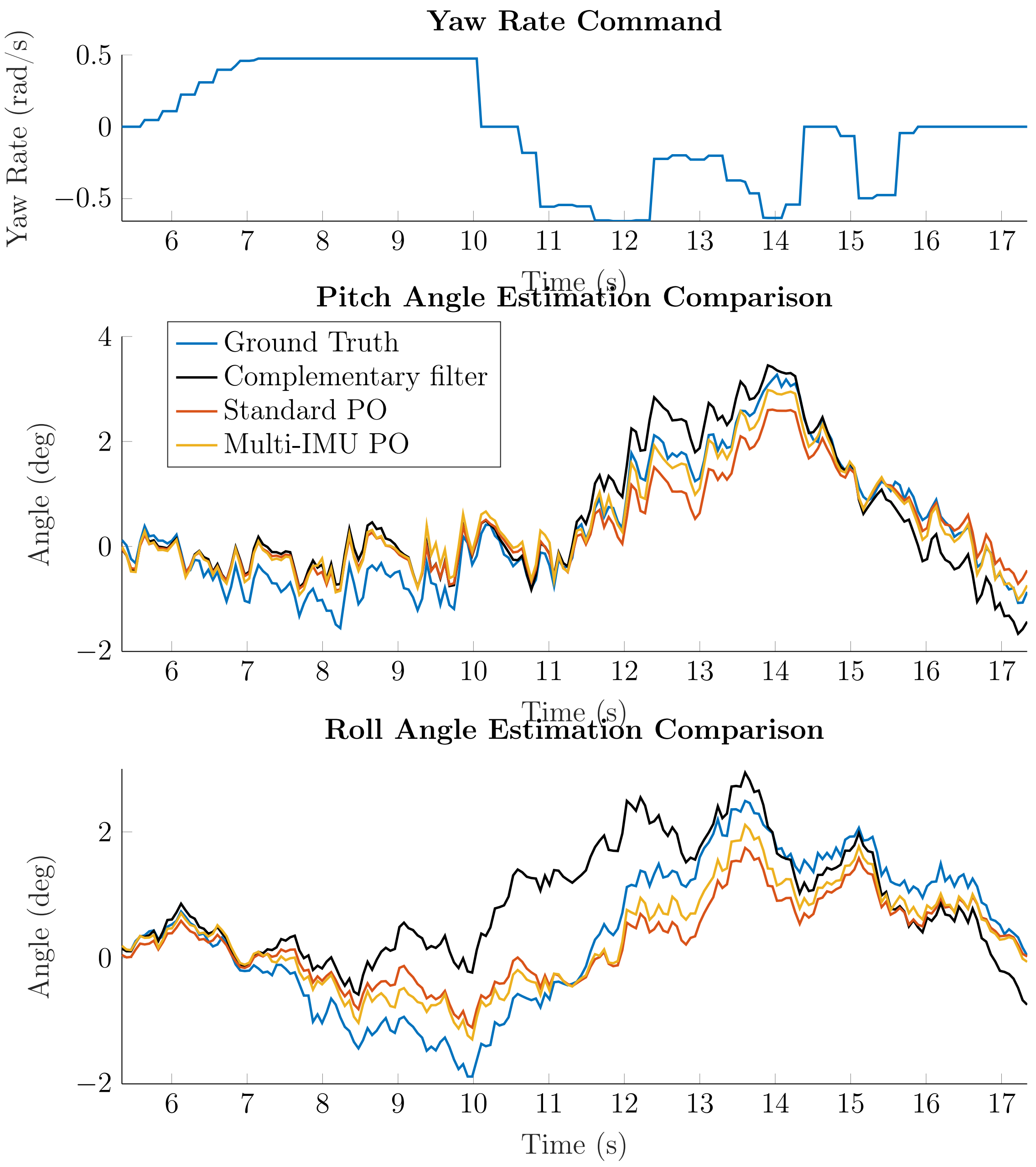


Figure 5-7: To show how different orientation estimation methods perform differently during fast in-place rotation, we compare the user input yaw rate command to the robot (**Top**) and estimated body pitch (**Middle**) and roll angles (**Bottom**). Positive command lets the robot spin counterclockwise. The faster the robot spins and the more abrupt the angular acceleration is, the more the estimated orientation will be wrong. Compared with the MoCap ground truth, average pitch and roll angular estimation errors (in deg) of different methods are: Complementary filter (0.019, 0.040), Standard PO (0.0185, 0.025), Multi-IMU PO (**0.015**, **0.018**).

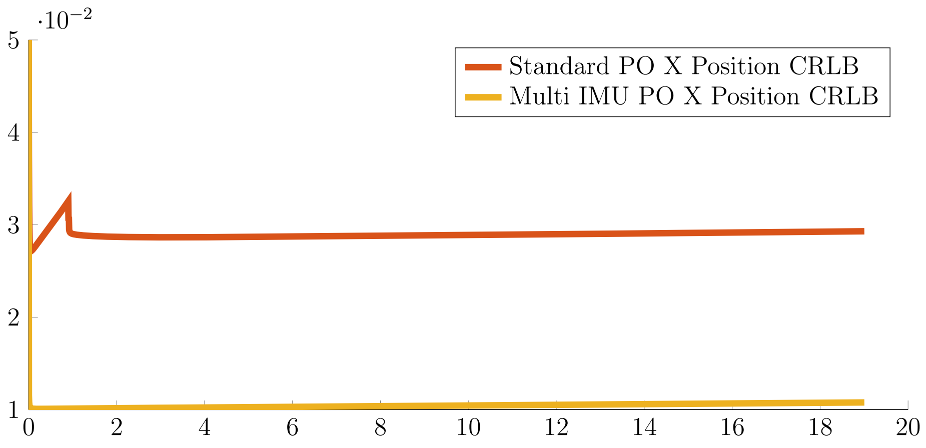


Figure 5-8: Plots of the element value corresponding to X position in the CRLB matrix for standard PO and Multi-IMU PO. From the same initial covariance value, both filters' CRLB first decreases and then increases gradually. The CRLB of Multi-IMU PO is much smaller than that of standard PO.

by replacing the statistical test with the measurement noise adjustment mechanism based on the contact flag. And 3) MIPO-C, which uses MIPO's process model, MIPO-P's measurement model, and the statistical test on the last measurement term instead of varying measurement noise. The results of standard PO, MIPO, and all three variants are shown in Fig. 5-6.

MIPO-F has performance closer to MIPO than standard PO, MIPO-P, and MIPO-C, showing that the largest performance contributor is the pivoting constraint. This result suggests that the zero-velocity model fundamentally limits the capability of standard PO to achieve low-drift position estimation, even if more accurate contact-flag generation methods can be used to avoid contact-detection errors in standard PO.

5.6 Conclusion

We have presented the Multi-IMU Proprioceptive Odometry (MIPO), a legged robot state estimation solution with IMUs in both the body and feet. Experiments have shown that additional IMUs in feet can significantly reduce position drift and improve overall estimation accuracy while keeping computation and hardware costs low. Moreover, MIPO provides an alternative method for detecting foot contact modes and foot slip without using contact sensors. It is a compelling replacement for conventional single-IMU PO. The process and measurement models in MIPO can be easily added to other EKF-based PO methods [14, 54]. MIPO's accuracy should further improve if IMU biases [19], kinematic parameters [163], and other calibration errors are addressed. Moreover, MIPO can estimate ground deformation or moving velocity. We also plan to replace standard PO with MIPO in the open-source Cerberus [165] odometry to improve its performance and robustness. Moreover, the contact flag generated by MIPO may also benefit downstream control algorithms. We will also study whether the contact flag generated by MIPO can benefit robot control. We also conjecture that for any contact-rich multi-rigid body systems, installing IMUs on each link could improve the state estimation performance of the entire system. A general estimation theory for such systems remains future work.

5.7 Appendix

The derivation of the process Jacobian has been presented in the VIO and legged robot state-estimation literature such as [41, 161] and Chapter 4. We will use Equation 3.16 to derive two key terms in the measurement Jacobian.

We define true orientation as $\boldsymbol{q}$ and our estimation as $\hat{\boldsymbol{q}}$, so the estimation error quaternion is $\delta\boldsymbol{q} = \hat{\boldsymbol{q}}^{-1} \otimes \boldsymbol{q}$. From standard quaternion kinematics [118]

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \begin{bmatrix} \boldsymbol{\omega} - \boldsymbol{b}_g \\ 0 \end{bmatrix} \tag{5.23}$$

we have

$$(\hat{\boldsymbol{q}}\dot{\otimes}\delta\boldsymbol{q}) = \dot{\hat{\boldsymbol{q}}} \otimes \delta\boldsymbol{q} + \hat{\boldsymbol{q}} \otimes \delta\dot{\boldsymbol{q}} = \frac{1}{2}\hat{\boldsymbol{q}} \otimes \delta\boldsymbol{q} \otimes \begin{bmatrix} \boldsymbol{\omega} - (\hat{\boldsymbol{b}}_g + \delta\boldsymbol{b}_g) \\ 0 \end{bmatrix} \tag{5.24}$$

which can be simplified to

$$\begin{aligned} 2\delta\dot{\boldsymbol{q}} &= \mathcal{R}(\begin{bmatrix} \boldsymbol{\omega} - (\hat{\boldsymbol{b}}_g + \delta\boldsymbol{b}_g)) \\ 0 \end{bmatrix})\delta\boldsymbol{q} - \mathcal{L}(\begin{bmatrix} \boldsymbol{\omega} - \hat{\boldsymbol{b}}_g \\ 0 \end{bmatrix})\delta\boldsymbol{q} \\ &= \begin{bmatrix} -\lfloor 2(\boldsymbol{\omega} - \hat{\boldsymbol{b}}_g) + \delta\boldsymbol{b}_g \rfloor^{\times} & -\delta\boldsymbol{b}_g \\ \delta\boldsymbol{b}_g & 0 \end{bmatrix} \delta\boldsymbol{q} \end{aligned} \tag{5.25}$$

Because $\delta\boldsymbol{q} = \mathrm{Exp}(\delta\boldsymbol{\theta}) = \begin{bmatrix} \frac{1}{2}\delta\boldsymbol{\theta} \\ 1 \end{bmatrix}$, then $\delta\dot{\boldsymbol{q}} = \begin{bmatrix} \frac{1}{2}\delta\dot{\boldsymbol{\theta}} \\ 0 \end{bmatrix}$. Ignoring any product of two error terms, we get

$$\begin{bmatrix} \delta\dot{\boldsymbol{\theta}} \\ 0 \end{bmatrix} = \begin{bmatrix} -\lfloor 2(\boldsymbol{\omega} - \hat{\boldsymbol{b}}_g) + \delta\boldsymbol{b}_g \rfloor^{\times} & -\delta\boldsymbol{b}_g \\ \delta\boldsymbol{b}_g & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{2}\delta\boldsymbol{\theta} \\ 1 \end{bmatrix} \tag{5.26}$$

$$\delta\dot{\boldsymbol{\theta}} = -\lfloor \boldsymbol{\omega} - \hat{\boldsymbol{b}}_g \rfloor^{\times}\delta\boldsymbol{\theta} - \delta\boldsymbol{b}_g \tag{5.27}$$

Alternatively, the result can be derived using rotation matrices. Write $\mathrm{Exp}(\delta\boldsymbol{\theta}) =$

$I + \lfloor \delta\boldsymbol{\theta} \rfloor^{\times}$. We notice

$$R = \hat{R}(I + \lfloor \delta\boldsymbol{\theta} \rfloor^{\times}) = \hat{R} + \hat{R}\lfloor \delta\boldsymbol{\theta} \rfloor^{\times} \tag{5.28}$$

then

$$\dot{R} = \dot{\hat{R}} + \dot{\hat{R}}\lfloor \delta\boldsymbol{\theta} \rfloor^{\times} + \hat{R}\lfloor \delta\dot{\boldsymbol{\theta}} \rfloor^{\times} \tag{5.29}$$

Then we can expand $\dot{R} = R\lfloor \boldsymbol{\omega} - \boldsymbol{b}_g \rfloor^{\times}$, the matrix form of the rotation dynamics [91] as

$$\dot{\hat{R}} + \dot{\hat{R}}\lfloor \delta\boldsymbol{\theta} \rfloor^{\times} + \hat{R}\lfloor \delta\dot{\boldsymbol{\theta}} \rfloor^{\times} = (\hat{R} + \hat{R}\lfloor \delta\boldsymbol{\theta} \rfloor^{\times})\lfloor \boldsymbol{\omega} - \boldsymbol{b}_g \rfloor^{\times} \tag{5.30}$$

$$\begin{aligned} \lfloor \delta\dot{\boldsymbol{\theta}} \rfloor^{\times} &= \lfloor \delta\boldsymbol{\theta} \rfloor^{\times}\lfloor \boldsymbol{\omega} - \hat{\boldsymbol{b}}_g \rfloor^{\times} \\ &\quad - \lfloor \boldsymbol{\omega} - \hat{\boldsymbol{b}}_g \rfloor^{\times}\lfloor \delta\boldsymbol{\theta} \rfloor^{\times} - \lfloor \delta\boldsymbol{b}_g \rfloor^{\times} \end{aligned} \tag{5.31}$$

Since $\lfloor \boldsymbol{a} \times \boldsymbol{b} \rfloor^{\times} = \lfloor \boldsymbol{a} \rfloor^{\times}\lfloor \boldsymbol{b} \rfloor^{\times} - \lfloor \boldsymbol{b} \rfloor^{\times}\lfloor \boldsymbol{a} \rfloor^{\times}$, then again

$$\lfloor \delta\dot{\boldsymbol{\theta}} \rfloor^{\times} = \lfloor -(\boldsymbol{\omega} - \hat{\boldsymbol{b}}_g) \times \delta\boldsymbol{\theta} \rfloor^{\times} - \lfloor \delta\boldsymbol{b}_g \rfloor^{\times} \tag{5.32}$$

$$\delta\dot{\boldsymbol{\theta}} = -\lfloor \boldsymbol{\omega} - \hat{\boldsymbol{b}}_g \rfloor^{\times}\delta\boldsymbol{\theta} - \delta\boldsymbol{b}_g, \tag{5.33}$$

The result is the same as Equation 5.27.

5.7.1 Measurement Jacobian Terms

For term in Equation 5.10

$$\boldsymbol{h}_2 = \mathrm{Log}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1} \otimes \hat{\boldsymbol{q}}^{-1} \otimes \hat{\boldsymbol{q}}_f)$$

First, we calculate the body orientation derivative.

$$\begin{aligned}\frac{\partial \boldsymbol{h}_2}{\partial \delta\theta} &= \lim_{\delta\theta\to 0}\frac{1}{\delta\theta}(\mathrm{Log}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1}\otimes[\hat{q}\otimes\mathrm{Exp}(\delta\theta)]^{-1}\otimes\hat{q}_f)\\ &\quad -\mathrm{Log}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1}\otimes\hat{q}^{-1}\otimes\hat{q}_f))\\ &= \lim_{\delta\theta\to 0}\frac{1}{\delta\theta}(\mathrm{Log}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1}\otimes\mathrm{Exp}(\delta\theta)^{-1}\otimes\hat{q}^{-1}\otimes\hat{q}_f)\end{aligned}$$

$$\begin{aligned}&\quad -\mathrm{Log}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1}\otimes\hat{q}^{-1}\otimes\hat{q}_f))\\ &= \lim_{\delta\theta\to 0}\frac{1}{\delta\theta}(2\Bigg[\mathcal{R}(\hat{q}^{-1}\otimes\hat{q}_f))\mathcal{L}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1})\begin{bmatrix}-\frac{1}{2}\delta\theta\\ 1\end{bmatrix}\\ &\quad -\mathcal{R}(\hat{q}^{-1}\otimes\hat{q}_f)\mathcal{L}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1})\begin{bmatrix}0\\ 1\end{bmatrix}\Bigg]_{3\times 3})\\ &= \lim_{\delta\theta\to 0}\frac{-[\mathcal{R}(\hat{q}^{-1}\otimes\hat{q}_f)\mathcal{L}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1})]_{3\times 3}\delta\theta}{\delta\theta}.\end{aligned} \tag{5.34}$$

An interesting property of the multiplicative maps is $\mathcal{L}(q)_{3\times 3} = \mathcal{R}(q^{-1})_{3\times 3}$. So

$$\frac{\partial \boldsymbol{h}_2}{\partial \delta\theta} = -[\mathcal{R}(\hat{q}^{-1}\otimes\hat{q}_f)\mathcal{L}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1})]_{3\times 3} \tag{5.35}$$

$$= -[\mathcal{L}(\hat{q}_f^{-1}\otimes\hat{q})\mathcal{R}(\boldsymbol{q}(\boldsymbol{\alpha}))]_{3\times 3}. \tag{5.36}$$

Secondly, we compute the foot orientation derivative.

$$\begin{aligned}
\frac{\partial \boldsymbol{h}_2}{\partial \delta\theta_f} =& \lim_{\delta\theta_f \to 0} \frac{1}{\delta\theta_f}(\mathrm{Log}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1} \otimes \hat{q}^{-1} \otimes \hat{q}_f \otimes \mathrm{Exp}(\delta\theta_f)) \\
& - \mathrm{Log}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1} \otimes \hat{q}^{-1} \otimes \hat{q}_f)) \qquad (5.37) \\
=& \lim_{\delta\theta_f \to 0} \frac{1}{\delta\theta_f}(2\Bigg[\mathcal{L}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1} \otimes \hat{q}^{-1} \otimes \hat{q}_f) \begin{bmatrix} \frac{1}{2}\delta\theta_f \\ 1 \end{bmatrix} \\
& - \mathcal{L}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1} \otimes \hat{q}^{-1} \otimes \hat{q}_f) \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Bigg]_{3\times 3}) \qquad (5.38) \\
=& \mathcal{L}(\boldsymbol{q}(\boldsymbol{\alpha})^{-1} \otimes \hat{q}^{-1} \otimes \hat{q}_f)_{3\times 3}. \qquad (5.39)
\end{aligned}$$

Chapter 6

Multi-IMU Visual-Inertial-Leg Odometry

The fact that adding more IMUs to PO can significantly improve estimation performance inspires us to consider incorporating more IMUs into VILO as well. In this chapter, we explore multi-IMU VILO formulations, validate them through various indoor and outdoor experiments, and use a multi-IMU VILO to provide feedback to the robot controller. Additionally, we qualitatively study how different controller settings can affect state estimation performance.

6.1 Cerberus 2.0

Building upon the Multi-IMU PO, we present Cerberus 2.0, a novel Multi-IMU VILO state estimator that fuses camera images, body IMU, joint encoders, and foot IMUs to achieve low-drift position estimation and precise orientation and velocity estimation. This estimator follows the general structure of the optimization-based

VILO described in Section 4.3.3, but the contact preintegration considers the foot IMU data. We present two different approaches to do so. The first one is to integrate the velocity estimation from a Multi-IMU PO directly as the contact preintegration term. The second approach is to process raw sensor information in a way similar to (4.20). We will explain them in this section and study their robustness in Section 6.2. Additionally, the estimator also leverages a Multi-IMU PO that runs in parallel with the VILO to provide contact estimation. Cerberus 2.0 is an extended and improved version of our previous contribution Cerberus [165].

6.1.1 Software Architecture

The overview of the software Cerberus 2.0 is shown in Fig. 6-1. Key building blocks and links among them are highlighted. The most important parts of the VILO consist of the rightmost three blocks and the Multi-IMU PO is on the bottom left. In VILO, as Section 4.3.3 explains, solving the nonlinear least square problem represented by a factor graph is a well-established technique. Using the marginalization to reduce factor graph size hence maintaining constant computation consumption is also very standard. It is how different factors are constructed that affects estimation accuracy the most. The Multi-IMU PO interacts with the VILO in two ways. First, the VILO uses the contact flag generated by (5.14) in the Multi-IMU PO. Second, Multi-IMU PO corrects its yaw angle using the VILO output orientation estimation. Compared to conventional VILO, our method includes special contact preintegration terms that relate to sensor data from joint encoders and foot IMUs.

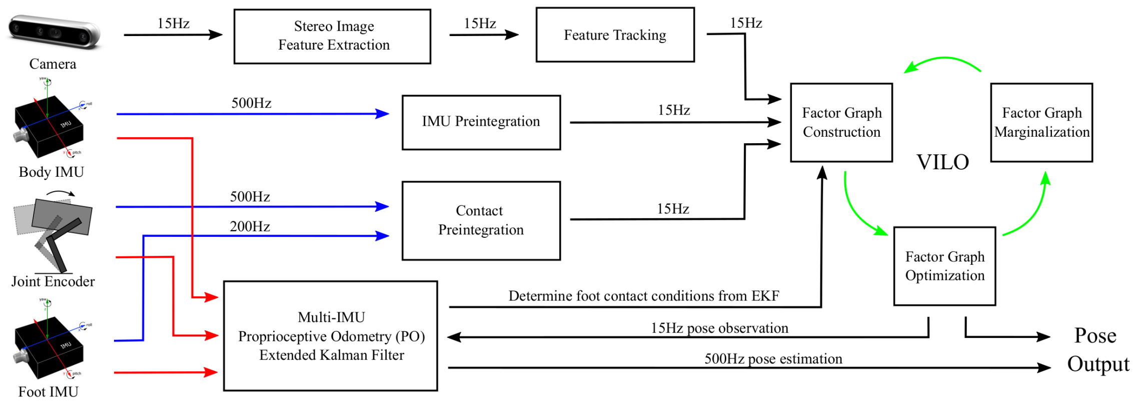

Camera
Body IMU
Joint Encoder
Foot IMU
15Hz
Stereo Image Feature Extraction
15Hz
Feature Tracking
15Hz
500Hz
IMU Preintegration
15Hz
500Hz
200Hz
Contact Preintegration
15Hz
Factor Graph Construction
Factor Graph Marginalization
VILO
Factor Graph Optimization
Multi-IMU Proprioceptive Odometry (PO) Extended Kalman Filter
Determine foot contact conditions from EKF
15Hz pose observation
500Hz pose estimation
Pose Output

Figure 6-1: The software architecture of Cerberus 2.0.

6.1.2 Loosely Coupled Leg Residual

An easier way to incorporate information from leg sensors into a visual and inertial-based factor graph is to leverage the existing PO. As explained in Section 3.3.11, the PO, either MIPO or SIPO, can output pose estimation at the frequency of IMU. Therefore along with the IMU preintegration, we can directly use the velocity estimation from PO to form a residual [158]. Again we focus on the period between state $\hat{\boldsymbol{x}}_k$ and $\hat{\boldsymbol{x}}_{k+1}$, let ν_i be the PO velocity estimation at time t_i, the associated covariance of which is σ_i. The covariance can be extracted from the estimation covariance of the PO output. A contact preintegration term is defined as:

$$\hat{\boldsymbol{\epsilon}}^k_{k+1} = \sum_{i=1}^{L} \nu_i \delta t \tag{6.1}$$

This term is the total body displacement measured by the leg odometry. It is in the world frame, so a residual is defined as:

$$\begin{aligned} &\boldsymbol{r}''(\hat{\boldsymbol{x}}_k, \hat{\boldsymbol{x}}_{k+1}, Z_{\Delta k}) = \\ &\left[(\hat{\boldsymbol{p}}_{k+1} - \hat{\boldsymbol{p}}_k) - \hat{\boldsymbol{\epsilon}}^k_{k+1}\right], \end{aligned} \tag{6.2}$$

which replaces the leg residual (4.19) in Problem (4.10). The residual covariance for it is simply:

$$\hat{P}^k_{k+1} = \sum_{i=1}^{L} \sigma_i \delta t \tag{6.3}$$

This formulation is deemed as "loosely coupled" [130] because the raw sensor data from foot IMUs and leg sensors are processed by the PO first. So the VILO cannot denoise or remove bias from the raw sensor information. Intuitively, in this context, Problem (4.10) calculates an educated weighted average of the output velocities from a VIO and a PO. It is worth noting that, in this scenario, calling (6.1) "contact preintegration" is not entirely descriptive, as it doesn't explicitly involve contact information in the calculation but we maintain this terminology for the sake of consistency.

6.1.3 Tightly Coupled Leg Residual

On the other hand, the "tightly coupled" approach devises a contact preintegration term using raw sensor information directly. Biases in foot IMUs and leg kinematic parameters [163] are explicitly considered and estimated in the VILO framework.

We augment the robot state to be $\hat{\boldsymbol{x}}_k = [\hat{\boldsymbol{p}}_k; \hat{\boldsymbol{q}}_k; \hat{\boldsymbol{v}}_k; \hat{\boldsymbol{b}}_{ak}; \hat{\boldsymbol{b}}_{\omega k}; \hat{\boldsymbol{b}}_{fk}; \hat{\boldsymbol{d}}_k]$. Compared to the state definition in Section 4.3.3, this augmented state contains foot IMU gyroscope bias $\hat{\boldsymbol{b}}_{fk}$ and leg kinematic parameter $\hat{\boldsymbol{d}}_k$.

From (3.44) and (4.21) we can write a body velocity measurement model taking body IMU angular velocity, foot IMU angular velocity, joint angles, joint angle velocities, and biases in the robot state as inputs:

$$\begin{aligned}\boldsymbol{v}_w =& - R(\hat{\boldsymbol{q}}_k)(J(\boldsymbol{\phi})\dot{\boldsymbol{\phi}} + \lfloor \boldsymbol{\omega}_b - \hat{\boldsymbol{b}}_w \rfloor^\times g(\boldsymbol{\phi})) \\ & -R(\hat{\boldsymbol{q}}_k)R_f^b(\boldsymbol{\phi})(\boldsymbol{\omega}_f - \hat{\boldsymbol{b}}_f) \times dR(\hat{\boldsymbol{q}}_k)\boldsymbol{g}(\boldsymbol{\phi})/\|R(\hat{\boldsymbol{q}}_k)\boldsymbol{g}(\boldsymbol{\phi})\| \end{aligned} \tag{6.4}$$

Noticing that rotation preservers norm and $R\boldsymbol{a} \times R\boldsymbol{b} = R(\boldsymbol{a} \times \boldsymbol{b})$ for any rotation matrix R and vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ [103], we can convert (6.4) into $\boldsymbol{v}_w = R(\hat{\boldsymbol{q}}_k)\boldsymbol{v}_b$, where $\boldsymbol{v}_b$ is a body frame vector represents a velocity observation:

$$\boldsymbol{v}_b = -J(\boldsymbol{\phi})\dot{\boldsymbol{\phi}} - \lfloor \boldsymbol{\omega}_b - \hat{\boldsymbol{b}}_w \rfloor^\times \boldsymbol{g}(\boldsymbol{\phi}) \tag{6.5}$$

$$-d_0 \lfloor R_f^b(\boldsymbol{\phi})(\boldsymbol{\omega}_f - \hat{\boldsymbol{b}}_f) \rfloor^\times \frac{\boldsymbol{g}(\boldsymbol{\phi})}{\|\boldsymbol{g}(\boldsymbol{\phi})\|} \tag{6.6}$$

Given this body velocity observation $\boldsymbol{v}_b$ that directly uses raw sensor data, we define a tightly coupled contact preintegration term as:

$$\hat{\boldsymbol{\epsilon}}'^k_{k+1} = \sum_{i=1}^{L} R(\hat{\gamma}_i^k)\boldsymbol{v}_b(i)\delta t \tag{6.7}$$

This term measures a body displacement vector represented in frame $\boldsymbol{q}_k$. Hence we can define a residual:

$$\boldsymbol{r}'''(\hat{\boldsymbol{x}}_k, \hat{\boldsymbol{x}}_{k+1}, Z_{\Delta k}) = \begin{bmatrix} R(\hat{\boldsymbol{q}}_k)^T(\hat{\boldsymbol{p}}_{k+1} - \hat{\boldsymbol{p}}_k) - \hat{\boldsymbol{\epsilon}}'^{k}_{k+1} \\ \hat{\boldsymbol{b}}_{\omega k+1} - \hat{\boldsymbol{b}}_{\omega k} \\ \hat{\boldsymbol{b}}_{fk+1} - \hat{\boldsymbol{b}}_{fk} \\ \hat{\boldsymbol{d}}_{k+1} - \hat{\boldsymbol{d}}_k \end{bmatrix}, \tag{6.8}$$

The residual covariance is calculated in a similar way as (4.15). We put detailed derivations in the Appendix.

6.2 Implementation & Experiments

After implementing Cerberus 2.0 as a highly efficient C++ package, we conducted extensive hardware experiments to evaluate it.

We first describe the sensor hardware implementation and experimental setup. Then, we evaluate the Multi-IMU PO component individually, highlighting its significant performance gain over standard PO. Third, we present the results of various indoor and outdoor experiments for Cerberus 2.0. The two contact preintegration methods proposed in Section 6.1 are carefully compared under different locomotion conditions to examine their robustness. Also, we study how different gait types, gait frequencies, locomotion speeds, and terrain types affect the estimator accuracy. To the best of the authors' knowledge, this is the first systematic study of legged robot state estimator performance under these conditions.

6.2.1 Sensor Hardware

We built the necessary sensor hardware using off-the-shelf low-cost components. A Unitree Go 1 robot is selected as the locomotion platform, but the state estimator should work on any quadruped robot system.

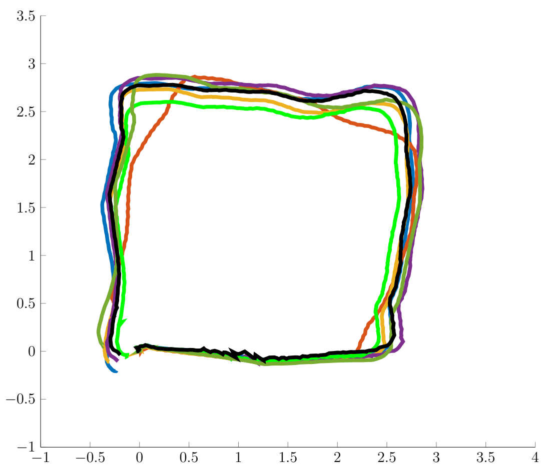
The Cerberus 2.0 hardware does not significantly alter the form factor of the Go1 robot. Additional to robot built-in sensor and foot IMUs described in Fig. 5-5, We also installed an Intel Realsense camera on the robot to provide 15Hz stereo vision RGB images. An Intel NUC computer with an i7-1165G7 CPU finally collects all sensor data from the Go1 robot, camera, and the Teensy board to run the Cerberus 2.0 algorithm. The computer also runs a nonlinear predictive control locomotion controller [42] which has a number of locomotion gaits available.

6.2.2 Evaluation Metrics

It is worth describing the performance metric we are going to use in the experiments in detail first. We use drift percentage to determine how much the estimated position deviates from the actual position over the course of long-distance traveling. We also note that the position to be compared only contains XY directions, for absolute Z position can be readily measured by fusing barometer pressure data. Along the travel trajectory, at each time instance t we can calculate the total travel distance s from the ground truth data. Also, we denote the estimated position as $\hat{\boldsymbol{p}}$ and the ground truth position as $\boldsymbol{p}$. Then the drift percentage is:

$$DRIFT(t) = \frac{\|\boldsymbol{p} - \hat{\boldsymbol{p}}\|}{s} \times 100\%$$

When the robot moves in simple trajectories such as a line or a single loop,



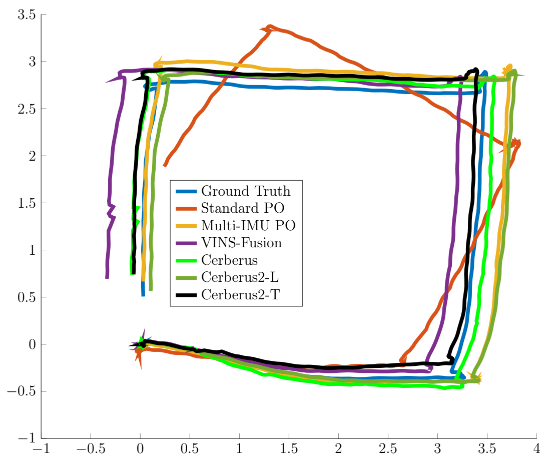


Figure 6-2: A comparison of several PO or vision-based baseline methods and Cerberus2 variants on two indoor data sequences. In the left figure, the robot uses the standing trot gait, 0.40s gait time, and 0.2m/s speed target. The right figure is the result of the robot's moves using the trot gait, 0.44s gait time, and $0.5m/s$ speed. The average drifts of all methods in these two runs are Standard PO - 7.56%, Multi-IMU PO - 1.86%, VINS-Fusion - 2.43%, Cerberus - 1.65%, Cerberus2-L **1.15%**, and Cerberus2-T **1.01%**.

then the final drift percentage $DRIFT(T)$, where T is the final trajectory time, is already informative. If the motion trajectory is more complicated, the average drift percentage:

$$AVR\_DRIFT = \frac{1}{N}\sum_{t=t_0}^{T} DRIFT(t)$$

captures the drift performance over time better, where N is the total number of time instances. Similarly, the median drift or MED_DRIFT can be calculated following the standard definition.

6.2.3 Cerberus 2.0 Evaluation

In this section, we evaluate the position estimation performance of Cerberus 2.0. The two main formulations of LO preintegration as discussed in Section 6.1.2 and 6.1.3 are compared through experiments on various data sequences. In addition to comparing with the ground truth, for each data sequence, we also make comparisons against baseline methods including Multi-IMU PO, VIO (VINS-Fusion), VILO (Cerberus). The two Cerberus 2.0 variants are Cerberus2-L for the loosely-coupled formulation and Cerberus2-T for the tightly-coupled formulation.

Indoor Drift & Robustness Evaluation

We first did the same experiment as discussed in Section 5.5. Estimated trajectories of two indoor runs are shown in Figure 6-2 as examples. For moderate speeds ($\leq 0.5m/s$, the Multi-IMU PO is comparable with vision-based methods and Cerberus2-T has better accuracy than the other Cerberus2 variants. But we came to notice when the robot moves faster, the estimation results become worse. Besides, different gait types and gait frequencies also cause position drift performance to change.

We aim to understand how different locomotion conditions affect position estimation drifts through a large number of controlled experiments. The three independent variables we adjust in these experiments are gait type (trot, standing trot, and flying trot), target moving speed ($0.2m/s$, $0.4m/s$, $0.6m/s$, $0.8m/s$, $1.0m/s$, and $1.2m/s$), and gait time (0.32s, 0.36s, 0.40s, 0.44s, 0.48s, and 0.52s). The foot contact schedule of each gait type is shown in Fig. 6-3. The trot gait, containing two alternating double support phases, is widely used in the community as one of the most reliable gaits. The standing trot gait adds a 50ms stance period after each trot foot switching, while the flying trot gait adds a 50ms full flight phase. The gait time is defined as the total duration of double support phases. The moving speed is a target speed that the locomotion controller wants to reach as soon as possible in each experiment run. A human operator provides the speed target during experiments. For faster target linear moving speed, the target yaw rate is also scaled up to ensure smooth turning. We would like to emphasize that no prior work has studied the effect of this wide range of moving speeds and gait variations on state estimation performance.

57 experiment runs are conducted in the same lab space mentioned in Section 5.5. In each run, we select a set of variable combinations for the robot controller, and then the robot travels a square trajectory that is used across experiments, as shown in Fig. 6-2. So during all experiment runs, the visual features along the trajectory are roughly the same. Although the trajectories are short, the robot experiences fast rotations, foot slippages, and unexpected contacts when moving at high speed. Since long gait time will lead to controller instability at high moving speeds, not all variable combinations are tested. We collect sensor data from one run into one rosbag and run all baseline methods and Cerberus2 methods using the sensor data offline to make a fair comparison. All methods use consistent parameters even though controller parameters change so that we can examine their parameter robustness. After obtaining the estimation results for all runs, these results were then categorized into three groups for focused analysis of each independent variable.

In Fig. 6-3 we compare the average drifts of different gaits. Among the three gaits, the trot gait has the lowest drift and the flying trot has the largest. Because of the airborne phase, the flying trot makes the robot experience large impacts upon touchdown so the worst performance is not surprising. Between the trot gait and the standing trot, the trot has a shorter contact time and fewer rolling contacts, which could explain its better performance. Therefore standard trot gait is preferred for achieving the best position estimation performance.

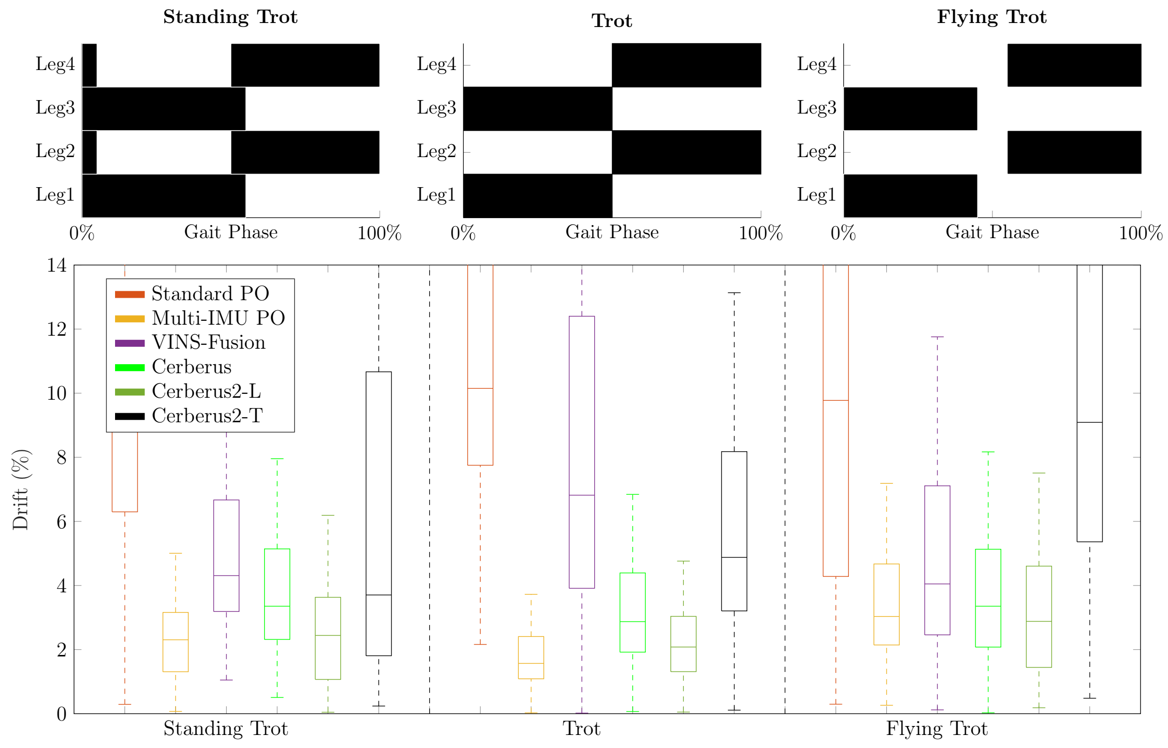


Figure 6-3: **Top:** Leg contact schedules of each gait type. Black represents time periods where the leg is in contact with the ground. **Bottom:** Box plots of average drifts of different methods. Results are categorized by gait types. For each type, we show the drift statistics from all experiment runs for each individual method.

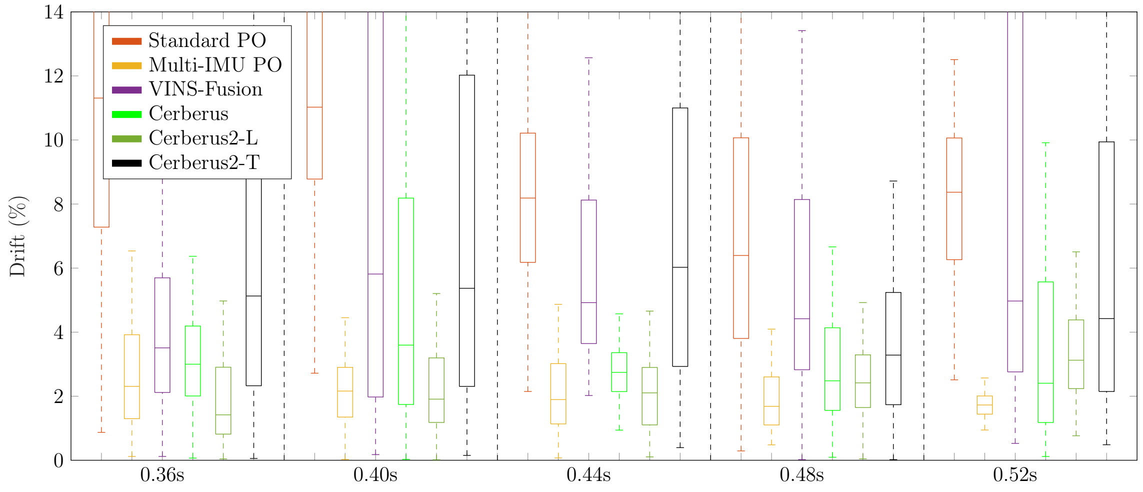


Figure 6-4: Box plots for estimation results of different gait frequencies.

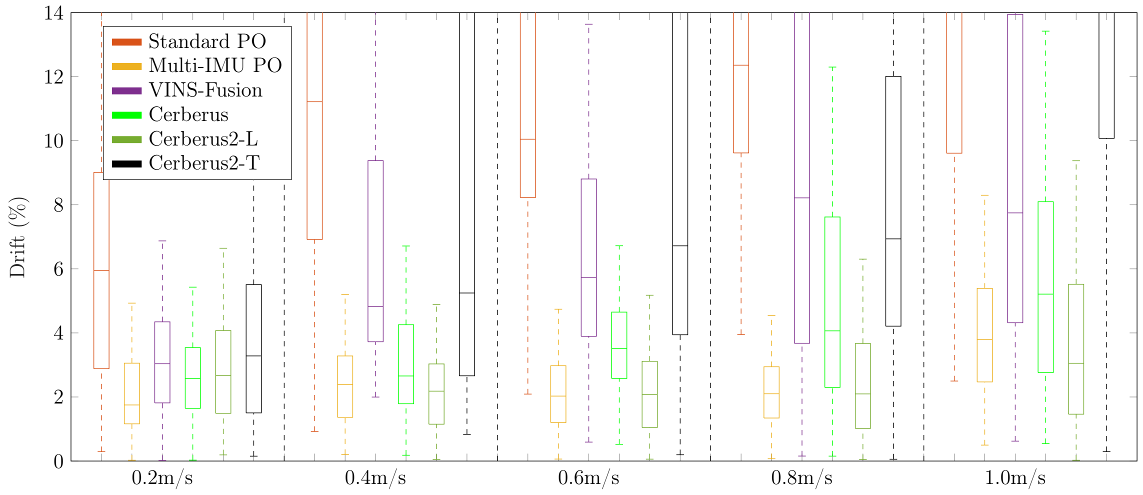


Figure 6-5: Box plots for estimation results of different movement speeds.

In Fig. 6-4, we can see that as gait time increases, the drift estimation results of the Multi-IMU PO monotonically decrease while other vision-based methods do not show a clear changing pattern. The VINS-Fusion even got much worse with a gait time of 0.52s. The reason is as gait time increases, the robot will have larger body displacement per stride hence larger joint angle velocity. According to Kalman filter observability analysis [163], larger joint angle velocity will improve the numerical condition of the Kalman gain. On the other hand, as long as the robot controls its orientation well, gait time does not affect camera observation accuracy. However, if the gait time is too long and the robot target velocity is also high, the robot legs may exceed the workspace limit, resulting in unstable locomotion, which leads to rapid body rotation and tampers the cameras' imaging quality. Around 0.44s gait time, Multi-IMU PO (min drift 1.31%, average drift 1.50%) and Cerberus2-L (min drift 1.08%, average 2.07%) achieve the best drift performance.

As shown in Fig. 6-5, when robot target speed increases, VINS-Fusion's drift monotonically increases, so do Standard PO, Cerberus, and Cerberus2-T. However, Multi-IMU PO and Cerberus2-L's drift performances are always around 2% until the target speed is higher than $1m/s$. By looking at sensor data we notice at $1m/s$ some foot IMUs' accelerometer readings occasionally exceed the range limit ($\pm 150m/s^2$). Our filter implementation does not consider sensor saturation, therefore it will diverge if saturation happens often. Another interesting phenomenon is in the speed range $0.6m/s$-$0.8m/s$, Multi-IMU PO and Cerberus2-L perform the best. The reason could be faster speed leads to fewer foot contact switches along a fixed travel distance, which reduces the effect of locomotion disturbances in the same way as increasing gait time.

Between Cerberus2-L and Cerberus2-T, across all experiments, Cerberus2-T demonstrates a much higher variance. Although in some cases Cerberus2-T achieves better

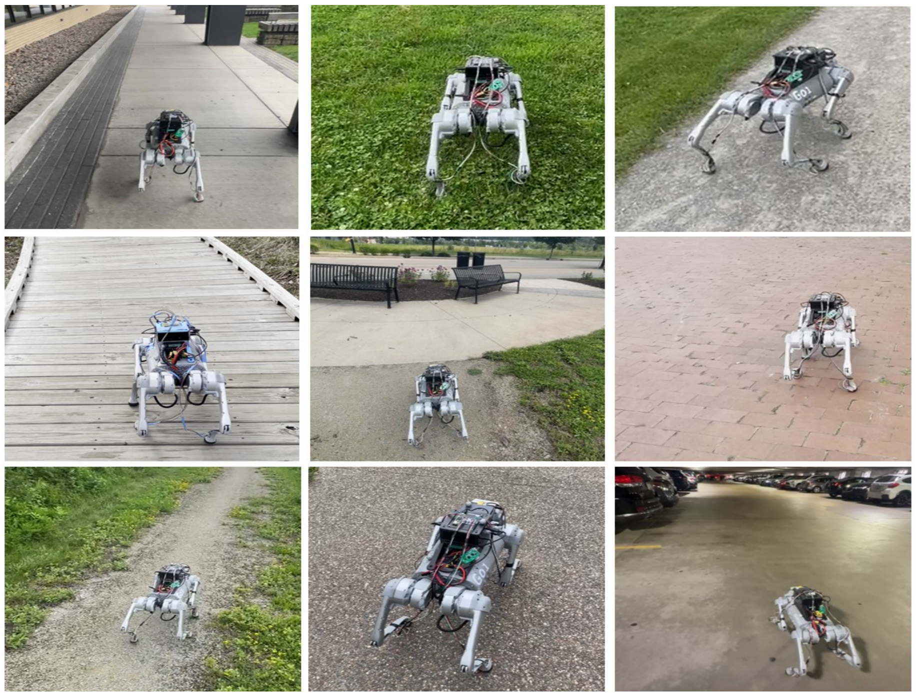


Figure 6-6: Different terrains during outdoor experiments.

performance, most of the time it is worse than Ceberus2-L or other baseline methods. This is because adding tightly coupled leg residuals not only introduces additional noise parameters to tune but also increases the problem's nonlinearity. Besides, because of the outlier rejection mechanism of the Multi-IMU PO, the velocity in the loosely coupled leg residual is less affected by foot impact and slippage, while the tightly coupled leg residual can only rely on the Huber norm in the factor graph for outlier rejection. Thus, our conclusion is that Cerberus2-L is the preferred choice for Multi-IMU VILO, even though it is a loosely coupled method. We also notice in the indoor environment, Multi-IMU PO, being a proprioceptive sensing method, is very competitive compared to the vision-based methods. This indicates that if we have a submodule in the system that already generates precise estimations, using it in a loosely coupled way could outperform the tightly coupled approaches that do not use this submodule.

Outdoor Drift & Terrain Robustness Evaluation

To fully test the accuracy, adaptability, and robustness of Cerberus 2.0, we let the robot travel over uneven terrains for long distances (see Figure 6-6 for screenshots showing the terrains). A dataset that includes 9 experiment runs with a total length of over 2.5km is collected. During each run, in addition to collecting sensor data, we also mounted an iPhone on the robot, from which we measured a separate sequence of GPS and IMU data from the cellphone. Then we use an inertial navigation filter to recover the ground truth.

For each sequence, we compare Cerberus 2.0 with VINS-Fusion, Cerberus, and ORB-SLAM3, an open-source localization library that has a multi-map loop closure ability that should improve estimation accuracy. In Figure 6-7, we visualize the estimation results of 4 outdoor runs.

Figure 6-9 demonstrates the robustness of Cerberus 2.0 during two indoor/outdoor switchings. In Figure 6-8, estimation results for the robot traveling along a hiking trail are shown. Because of gravel and plants on the trail surface, the robot often experiences foot slippage and unexpected contact.

Since our robot sensor is low cost, robot movement is agile, and experiment runs involve large-scale terrain variations, baseline vision-based methods, and PO methods often fail or have large drift. Among all methods and across all experiments, Cerberus2-L has the best performance.

Run Time Discussion

Cerberus 2.0 can run in real-time on robot hardware with moderate computation resources. In our implementation, either Cerberus2-L or Cerberus2-T takes less than 60ms to finish a single factor graph solve on the Intel i7-1165G7 CPU, which is shorter than the camera image arriving interval (1/15s or 66.7ms).

To further ensure run time consistency, Cerberus 2.0 runs on a Linux kernel with the real-time patch, thus the main factor graph optimization thread can be set to a higher priority to minimize operation system thread switching interruption. More implementation details and run time tips can be viewed in our open-source codebase.

6.3 Limitation & Future Work

The biggest limitation of Multi-IMU PO and VILO is we do not have the capability of dealing with sensor saturation. This is the primary reason for estimator failures if the robot motion controller is not well designed. Incorporating some signal reconstruction or saturation detection mechanism to further improve robustness is a

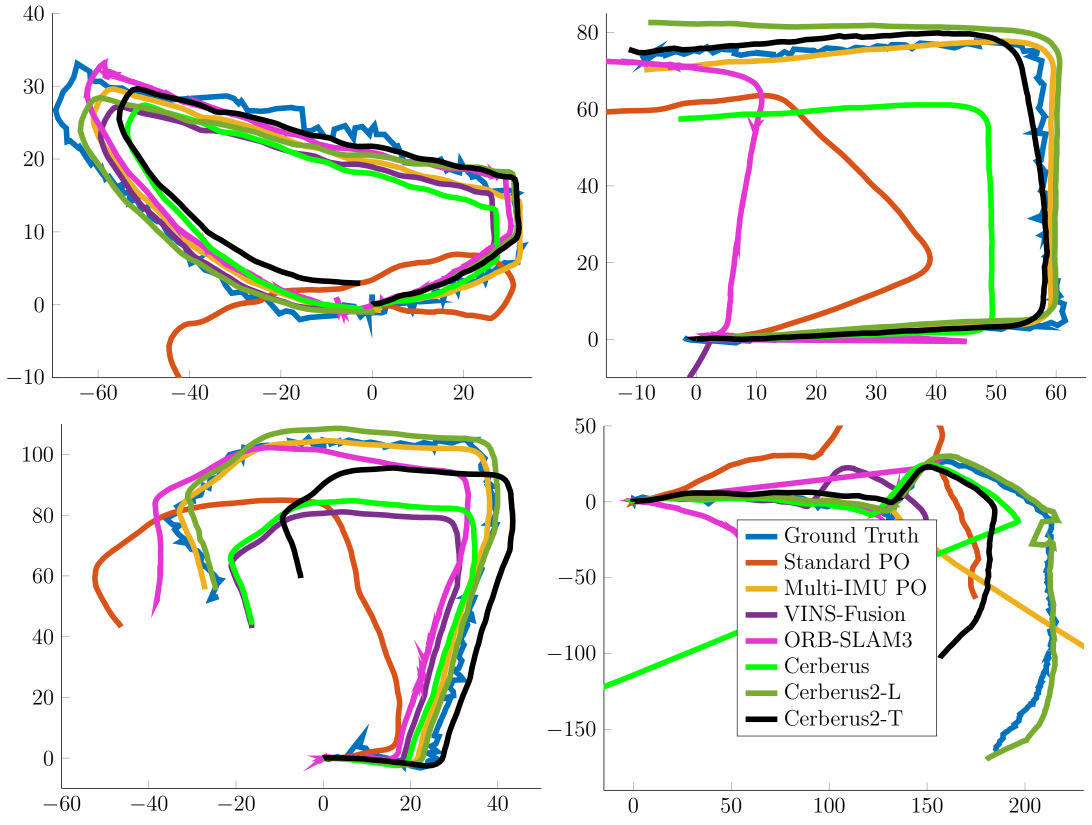


Figure 6-7: Estimation results of four outdoor runs. The average drift percentages of each method in each run are summarized in Table 6.1.

	Top-left	Top-right	Bottom-left	Bottom-right
Standard PO	17.67%	19.62%	14.15%	26.24%
Multi-IMU PO	2.70%	**1.11%**	**1.07%**	break
VINS-Fusion	3.04%	break	9.75%	19.61
ORB-SLAM3	1.83%	break	5.71%	break
Cerberus	4.95%	8.41%	8.57%	break
Cerberus2-L	**1.76%**	2.84%	2.04%	**2.73%**
Cerberus2-T	4.52%	1.88%	7.53%	12.77%

Table 6.1: The drift value of all outdoor experiment runs. Each column lists the results of one run. Across all results, Multi-IMU PO achieves the lowest drift values, while Cerberus2-L has comparable or better performance and improved robustness.

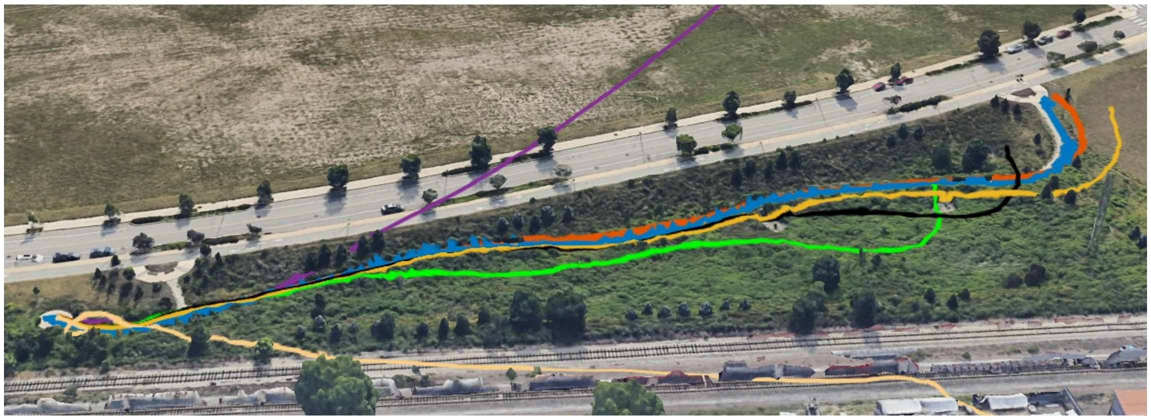


Figure 6-8: On a challenging hiking trail, Cerberus2-L (red) achieves lower than 10% drift performance while all other methods either fail or have drift larger than 10%. The estimation result of Cerberus2-L agrees with the ground truth (blue) and underlying satellite image very well. The map can be viewed online.

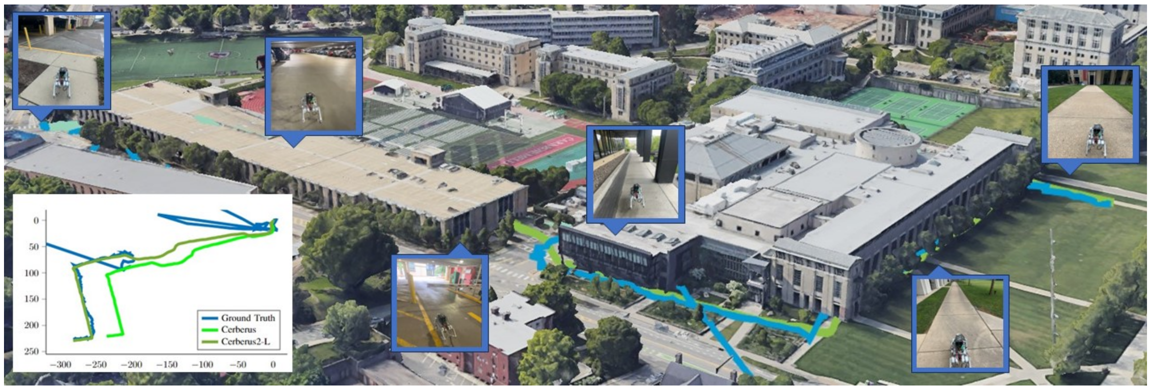


Figure 6-9: In an over 500m experiment run, a Unitree Go1 robot travels through a parking garage over various terrain types. The proposed Cerberus 2.0 algorithm demonstrates great estimation robustness and accuracy. Except for baseline method Cerberus and the Cerberus2-L method, all other methods break along the way. The ground truth trajectory is also wrong inside the garage due to weak GPS signals. After the robot exited the garage, the GPS was restored. Cerberus2-L has $< 0.5\%$ drift even after traveling for long distances with indoor/outdoor switching. The map can be viewed online.

promising future work.

In VILO, the selection of residual covariance values is mostly determined by trial and error. Since the factor graph optimizer only weighs residual terms relatively, the absolute value of covariances may not have physical meaning. Although there exist auto-tuning methods for Kalman filters, auto-tuning for factor graphs remains unexplored. Besides, due to vibrations and impacts, the actually sensor noise model may be heteroscedastic instead of constant. A more careful study of sensor noise characteristics in legged robot state estimation could not only benefit legged locomotion but also shed light on other sensor fusion problems.

6.4 Conclusions

In this chapter, we studied a multi-sensor visual-inertial-leg odometry hardware and algorithm solution that fuses information from one camera, multiple inertial sensors, and multiple motor encoders. The inclusion of inertial measurement units on the robot's feet significantly improves odometry accuracy. By comparing various formulations of visual-inertial-leg odometry under different locomotion conditions, we gained insights into both the theory and best implementation practices. In addition to this thesis chapter, we open-sourced our code implementation, Cerberus 2.0, along with a large dataset.

Chapter 7

Equality Constrained LQR With The Factor Graph

In the previous chapters, we have explored visualizing trajectory optimization as a graphical model and using the factor graph as the backbone for state estimation. Clearly, they are mathematically closely related because the underlying optimization algorithm is the same. In this chapter, we focus on the factor graph to conduct a theoretical analysis, demonstrating that this graphical model can be a powerful tool to solve not only state estimation problems but also control problems.

7.1 Motivation

The Equality Constrained Linear Quadratic Regulator (EC-LQR) is an important extension [81, 133] of the Linear Quadratic Regulator (LQR) [71]. The standard finite-horizon discrete-time LQR problem contains (1) quadratic costs on the state trajectory and the control input trajectory and (2) *system dynamics constraints* which enforce that the current state is determined by a linear function of the previous state and control. In the EC-LQR, *auxiliary constraints* are introduced to enforce additional linear equality relationships on one or more state(s) and/or control(s). Such auxiliary constraints occur commonly in robotics applications [81], especially in iterative nonlinear controllers or as local controllers for open-loop trajectories from trajectory optimization approaches [117].

In many important problems auxiliary constraints violate the Markov assumption, yet such constraints are rarely considered in existing EC-LQR approaches. We classify auxiliary constraints in EC-LQR problems into two categories which we term *local constraints* and *cross-time-step constraints.* A local constraint only contains a state and/or control from the same time step. Examples of local constraints include initial and terminal conditions on states, contact constraints, and states along a predefined curve. In contrast, a cross-time-step constraint involves multiple states and controls at different time instances. While transcribing cross-time-step constraints into local ones is possible by adding additional "book-keeping" states or other manual augmentations, doing so may be unsuitable for online operation and many cross-time-step constraints. Such non-Markovian constraints are pervasive in many robotics applications. For example, a legged robot's leg configuration must return to the same state after a period of time during a periodic gait [43]. In optimal allocation with resource constraints [20], the sum of control inputs is constrained to be some constant. Our goal is to solve for both optimal trajectories and optimal feedback control policies in EC-LQR problems with local and cross-time-step constraints in linear time with respect to the trajectory length, which no existing EC-LQR methods can achieve.

Reformulating control problems as inference problems [72, 85, 146, 155] is a growing alternative to common trajectory optimization [34, 74, 117] and dynamic programming (DP) approaches for optimal control [81, 88, 133] . While trajectory opti-

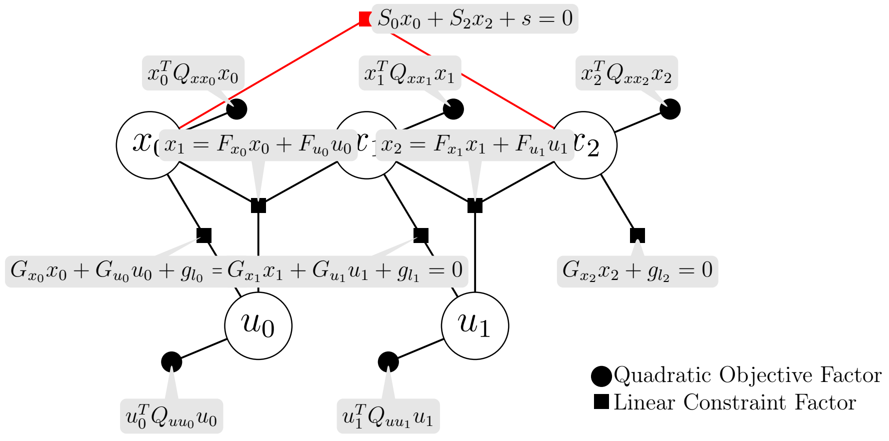

$S_0x_0 + S_2x_2 + s = 0$
$x_0^TQ_{xx_0}x_0$
$x_1^TQ_{xx_1}x_1$
$x_2^TQ_{xx_2}x_2$
x_0
$x_1 = F_{x_0}x_0 + F_{u_0}u_0$
x_1
$x_2 = F_{x_1}x_1 + F_{u_1}u_1$
x_2
$G_{x_0}x_0 + G_{u_0}u_0 + g_{l_0} = 0$
$G_{x_1}x_1 + G_{u_1}u_1 + g_{l_1} = 0$
$G_{x_2}x_2 + g_{l_2} = 0$
u_0
u_1
$u_0^TQ_{uu_0}u_0$
$u_1^TQ_{uu_1}u_1$
● Quadratic Objective Factor
■ Linear Constraint Factor

Figure 7-1: The factor graph representation of an Equality Constrained Linear Quadratic Regular (EC-LQR) problem. Circles with letters are states or controls. Filled squares and circles represent objectives and constraints that involve the state or controls to which they are connected. The red square represents a cross-time-step constraint.

mization focuses on open-loop trajectories rather than feedback laws, and a method using DP to handle cross-time-step constraints has yet to be proposed, control as inference may offer the advantages of both. Factor graphs, in particular, are a common tool for solving inference problems [36] and have recently been applied to optimal control [27, 38].

We propose a novel formulation using factor graphs [36] to efficiently solve the EC-LQR problem with both local and cross-time-step constraints in linear time with respect to trajectory length. We demonstrate how to represent the EC-LQR problem as a factor graph (shown in Fig. 7-1), In the graph, the cost function is Gaussian factors; the system dynamics are hard constraint factors; the equality constraints are also hard constraint factors. and apply the variable elimination (VE) algorithm [12] on the factor graph to solve for the optimal trajectories and optimal feedback control policies. The flexibility of the factor graph representation allows cross-time-step constraints with arbitrary numbers of variables to be seamlessly handled. As long as the maximum time index difference of variables involved in each constraint is bounded, the computational complexity stays linear with trajectory length. The approach in this chapter matches the computational complexity of standard dynamic programming techniques [2], but also has the added benefit of handling cross-time-step constraints.

7.2 Related Work

Trajectory optimization methods typically transcript a problem into a Quadratic Programming (QP) [6] or NonLinear Programming (NLP) [74] problem which can be efficiently solved to obtain open-loop trajectories of nonlinear systems. Local controllers can be used to track the open-loop trajectories generated [117]. Designing local controllers that obey equality constraints motivates EC-LQR problems.

For EC-LQR problems with just local constraints, DP-based approaches can generate both the optimal trajectories and feedback control policies. Solving standard LQR using DP is well understood in control theory [71]. [117] tackles EC-LQR with state-only local constraints by projecting system dynamics onto the constraint manifold. [133] extends the DP approach by using Karush-Kuhn-Tucker (KKT) conditions [20] to absorb auxiliary constraints into the cost function, but its computation time grows with the cube of the trajectory length for certain auxiliary constraints. [81] solves the EC-LQR with *local* constraints in linear complexity by adding a new auxiliary constraint dubbed "*constraint_to_go*" at each time step during DP steps.

Control as inference, in which a control problem is reformulated and solved as an inference problem, has gained considerable attention [85, 146]. Probabilistic Graphical Models (PGMs), which are commonly used for inference, have been applied to optimal control problems [72, 155] because they describe dependencies among vari-

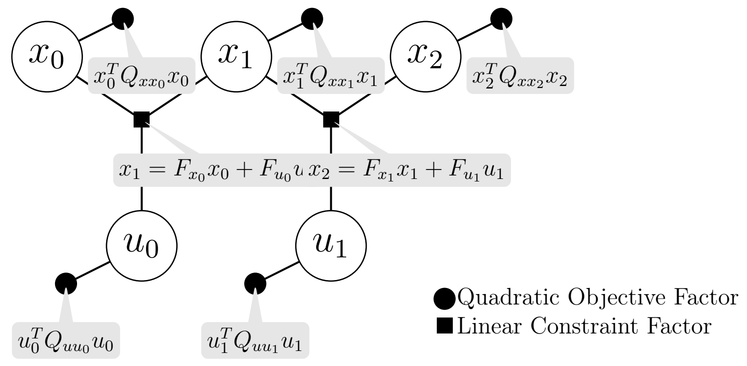
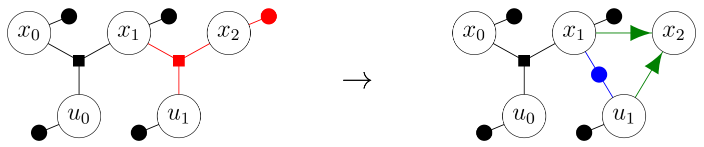


Figure 7-2: Factor graph of a standard LQR problem with trajectory length $T = 2$.

ables while maintaining sparsity in the graphical representation. Therefore, PGMs can solve for variable distributions efficiently by exploiting sparsity [78]. The Markov assumption gives optimal control problems a "chain" structure when represented as PGMs allowing linear computational complexity with respect to trajectory length [4, 27, 146, 155], but PGMs can also exploit sparsity for more complex (non-chain) structures which motivates using PGMs for cross-time-step constraints.

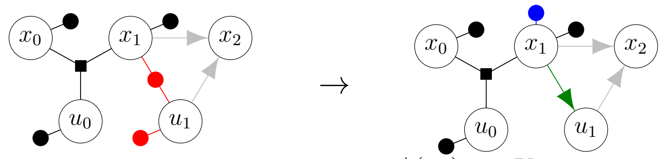
Factor graphs, a type of PGM, have been successfully applied to robot perception and state estimation [36]. Prior works have demonstrated that the variable elimination (VE) algorithm [12] on factor graphs can efficiently factorize the graphs' equivalent matrix representations in order to infer the posterior distributions of random variables. This procedure is called *factor graph optimization*. Moreover, factor graphs can encode constraints [33]. Other than estimation, factor graphs can be used to do motion planning [38, 141]. Standard LQRs with factor graphs are considered in [27, 155] without auxiliary constraints.



$$\begin{aligned} x_2^*(x_1,u_1) &= \arg\min_{x_2}\; x_2^T Q_{xx_T} x_2 \\ \text{s.t. }\; x_2 &= F_{x_1}x_1 + F_{u_1}u_1 \end{aligned} \;\rightarrow\; \begin{aligned} x_2^*(u_1,x_1) &= F_{x_1}x_1 + F_{u_1}u_1 \\ \phi^*_{x_2}(u_1,x_1) &= x_2^{*T} Q_{xx_T} x_2^* \end{aligned}$$

$$\begin{array}{c} W_{x_2} \\ \begin{bmatrix} I \\ \infty \end{bmatrix} \end{array} \begin{array}{c} \begin{array}{ccc} x_2 & u_1 & x_1 \end{array} \\ \left[\begin{array}{ccc|c} Q^{1/2}_{xx_T} & 0 & 0 & 0 \\ I & -F_{u_1} & -F_{x_1} & 0 \end{array}\right] \end{array} \rightarrow \begin{array}{c} W'_{x_2} \\ \begin{bmatrix} \infty \\ I \end{bmatrix} \end{array} \begin{array}{c} \begin{array}{ccc} x_2 & u_1 & x_1 \end{array} \\ \left[\begin{array}{ccc|c} I & -F_{u_1} & -F_{x_1} & 0 \\ 0 & Q^{\frac{1}{2}}_{xx_T}F_{u_1} & Q^{\frac{1}{2}}_{xx_T}F_{x_1} & 0 \end{array}\right] \end{array}$$

(a) Eliminate x_2



$$\begin{aligned} u_1^*(x_1) = \arg\min_{u_1}\; & \phi^*_{x_2}(x_1,u_1) \\ & + u_1^T Q_{uu_1} u_1 \end{aligned} \;\rightarrow\; \begin{aligned} u_1^*(x_1) =& -K_1x_1 \\ \phi^*_{u_1}(x_1) =& (K_1x_1)^T Q_{uu_1}(K_1x_1) \\ & + \phi^*_{x_2}(x_1,-K_1x_1) \end{aligned}$$

$$\begin{array}{c} W_{u_1} \\ \begin{bmatrix} I \\ I \end{bmatrix} \end{array} \begin{array}{c} \begin{array}{cc} u_1 & x_1 \end{array} \\ \left[\begin{array}{cc|c} Q^{\frac{1}{2}}_{xx_T}F_{u_1} & Q^{\frac{1}{2}}_{xx_T}F_{x_1} & 0 \\ Q^{\frac{1}{2}}_{uu_1} & 0 & 0 \end{array}\right] \end{array} \rightarrow \begin{array}{c} W'_{u_1} \\ \begin{bmatrix} R_1 \\ I \end{bmatrix} \end{array} \begin{array}{c} \begin{array}{cc} u_1 & x_1 \end{array} \\ \left[\begin{array}{cc|c} I & K_1 & 0 \\ & E_1 & 0 \end{array}\right] \end{array}$$

(b) Eliminate u_1

Figure 7-3: Two variable eliminations for the LQR problem. Each sub-figure consists of three rows showing three equivalent representations: the factor graph (top), constrained optimization (middle), and modified Gram-Schmidt process on $[A_i|b_i]$ (bottom). The arrows in the factor graphs show variable dependencies. The thin horizontal arrows separate cases before and after elimination. Terms and symbols in the same color correspond to the color-coded variable elimination steps in Section 7.3. Note that the matrix factorization representation consists of the weight vector, W_i, next to the sub-matrix $[A_i|b_i]$.

7.3 Preliminary

We first demonstrate how to represent standard LQR, Problem 7.2 with only constraint (7.2b), as the factor graph shown in Fig. 7-2 and subsequently obtain the optimal trajectory and optimal feedback control policy using VE.

Factor graphs can be interpreted as describing either a joint probability distribution with conditional independencies or, as we focus on in this chapter, an equivalent least-squares problem derived from minimizing the negative log-likelihood. A factor graph is a bipartite graph consisting of variables and factors connected by edges, where a factor can be viewed either as a joint probability density or least squares objective over the variables it is connected to.

We begin by showing how the probabilistic view of factor graphs is equivalent to a least squares minimization [36]. We construct factor graph to describe a joint probability distribution of the variables $X = [\mathbf{x}; \mathbf{u}]$. For Gaussian distributions, the probability distribution for a single objective or constraint factor ϕ_k can be written in matrix form as

$$\phi_k(X_k) \propto \exp\left\{-\tfrac{1}{2}\|A_k X_k - b_k\|^2_{\Sigma_k}\right\}$$

where exp is the exponential function and X_k contains the variables connected to the factor. A_k and b_k are a matrix and a vector with problem-specific values, Σ_k is the covariance of the probability distribution, and $||\cdot||^2_\Sigma \coloneqq (\cdot)^T\Sigma^{-1}(\cdot)$ denotes the square of the Mahalanobis norm. A_k, b_k, and Σ_k together define the probability density of the factor.

The product of all factors is the posterior distribution of X whose MAP estimate solves the least squares problem [36]:

$$X^{MAP} = \arg\max_X \ \phi(X) = \arg\min_X \ -\log(\prod_k \phi_k(X_k))$$
$$= \arg\min_X \ \sum_k \|A_k X_k - b_k\|^2_{\Sigma_k} = \arg\min_X \ \|AX - b\|^2_\Sigma \tag{7.1}$$

where A and Σ contain A_k and Σ_k on the block diagonal respectively and b stacks all b_k vertically. In this formulation, each factor ϕ_k corresponds to a block row in $[A|b]$. Defining the weight matrix $W \coloneqq \Sigma^{-1}$, X^{MAP} minimizes a weighted least squares expression $(AX - b)^T W(AX - b)$.

The objective factors in Fig. 7-2 are $\phi_{objx}(x_t) \propto \exp\{-\frac{1}{2}\|Q^{1/2}_{xx_t} x_t\|^2\}$ and $\phi_{obju}(u_t) \propto \exp\{-\frac{1}{2}\|Q^{1/2}_{uu_t} u_t\|^2\}$, while the constraint factors are $\phi_{dyn}(x_{t+1}, x_t, u_t) \propto \exp\{-\frac{1}{2}\|x_{t+1} - F_{x_t} x_t - F_{u_t} u_t\|^2_{\Sigma_c}\}$ where the covariance $\Sigma_c = 0$ creates infinite terms in W. When factor graphs have factors with zero covariance, the least squares problem turns into a *constrained* least squares problem which we can solve using e.g. modified Gram-Schmidt [50]. If linear terms are desired in the cost function in (7.2a) (e.g. track a non-zero setpoint), we can always express the objective factor in a Gaussian form as $\phi_{objx}(x_t) \propto \exp\{-\frac{1}{2}\|Q^{1/2}_{xx_t}(x_t - x_{ref})\|^2\}$, where x_{ref} is some tracking target.

The VE algorithm is a method to solve (7.1) while exploiting the sparsity of A by solving for one variable at a time. For a variable $\theta_i \in X$, we can identify its *separator* S_i: the set of other variables sharing factors with θ_i. Then we extract sub-matrices A_i, W_i, and sub-vector b_i from the rows of A, W, and b such that $[A_i|b_i]$ contains all factors connected to θ_i. We collect the rows in $[A_i|b_i]$ with finite weights to define objective factor $\phi_i(\theta_i, S_i)$ and rows with infinite weights to define constraint factor $\psi_i(\theta_i, S_i)$. Then we "eliminate" variable θ_i following 3 steps[1]:

[1]In the probabilistic form, steps 2 and 3 would come from factoring $\phi_i(\theta_i, S_i)\psi_i(\theta_i, S_i) \propto p(\theta_i|S_i)p(S_i)$. For Gaussian distributions, $\theta_i^*(S_i) = E[p(\theta_i|S_i)]$ and $\phi_i^*(S_i)\psi_i^*(S_i) = p(S_i)$.

Step 1. Identify all the factors adjacent to θ_i to get $[A_i|b_i]$. Split $[A_i|b_i]$ into $\phi_i(\theta_i, S_i)$ and $\psi_i(\theta_i, S_i)$.

Step 2. Solve the (constrained) least squares problem:

$$\theta_i^*(S_i) = \arg\min_{\theta_i} \ \phi_i(\theta_i, S_i) \ \text{ s.t. } \ \psi_i(\theta_i, S_i) = 0$$

using modified Gram-Schmidt or other constrained optimization methods [20, Ch.10]. $\theta_i^*(S_i)$ denotes that θ_i^* is a function of the variables in S_i.

Step 3. Substitute $\theta_i \leftarrow \theta_i^*$ by replacing the factors $\phi_i(\theta_i, S_i)$ and $\psi_i(\theta_i, S_i)$ with $\phi_i^*(S_i) \coloneqq \phi_i(\theta_i^*, S_i)$ and $\psi_i^*(S_i) \coloneqq \psi_i(\theta_i^*, S_i)$, respectively, in $[A|b]$.

We follow an *elimination order* [78] to eliminate one variable $\theta_i \in X$ at a time. After all variables are eliminated, the factor matrix A is effectively converted into an upper-triangular matrix R allowing X to be solved by matrix back-substitution. Therefore, one interpretation of the VE algorithm is performing sparse QR factorization on A [36].

To apply VE to the LQR factor graph in Fig. 7-2, we choose the ordering $x_N, u_{N-1}, x_{N-1}, \ldots, x_0$ and execute Steps 1-3 to eliminate each variable. This order is chosen to generate feedback policies where the controls are functions of the present states. When eliminating a state x_i for the special case of LQR, the constrained least-squares problem in Step 2 is trivially solved as $x_i^*(u_{i-1}, x_{i-1}) = F_u u_{i-1} + F_x x_{i-1}$. Additionally, $\psi_{x_i}^*$ will be empty since $\psi_{x_i}(x_i^*, u_{i-1}, x_{i-1})$ is satisfied for any choice of u_{i-1} and x_{i-1}. Fig. 7-3a shows the factor graphs, corresponding optimization problems, and sub-matrices $[W_i][A_i|b_i]$ before and after eliminating x_2.

The optimal feedback control policy emerges when eliminating a control u_i. The combined constraint factor ψ_{u_i} is empty (since $\psi_{x_{i+1}}^*$ is empty), so Step 2 reduces to an unconstrained minimization problem. To solve it using QR factorization, split the objective $||A_i[u;x]||_2^2 = ||R_iu + T_ix||_2^2 + ||E_ix||_2^2$ using the QR factorization $A_i = Q\begin{bmatrix} R_i & T_i \\ 0 & E_i \end{bmatrix}$ noting that Q is orthogonal and thus doesn't change the norm. Then, $u_i^*(x_i) = -K_ix_i$ where $K_i \coloneqq R_i^{-1}T_i$ efficiently optimizes the first term and $\phi_{u_i}^*(x_i) = ||E_ix||_2^2$ is the new factor on x. The elimination is shown in Fig. 7-3b.

Furthermore, the *cost_to_go* (or "value function" [9]), which commonly appears in DP-based LQR literature, is visually evident in the (right) factor graph from Fig. 7-3b as the sum of the two unary factors on x_1:

$$cost\_to\_go_1(x_1) = x_1^TQx_1 + x_1^TE_1^2x_1.$$

Continuing to eliminate the rest of the variables reveals the general formula of the *cost_to_go* after applying block-QR elimination to solve for K_i and E_i:

$$cost\_to\_go_i(x_i) = x_i^T(Q_{xx_t} + F_{x_t}^TV_{i+1}F_{x_t} - K_i^TF_{u_i}^TV_{i+1}F_{x_t})x_i$$

where V_{i+1} comes from $\phi_{u_i}^*(x_i) + x_i^TQ_{xxt}x_i = x_i^TV_ix_i$.

7.4 Problem And Method

In this section we first formulate the standard LQR and EC-LQR problems following the notation used in [81]. Then we solve a standard LQR problem as a factor graph and review relevant concepts related to factor graphs. Next, we solve EC-LQR with local constraints using factor graphs and compare our algorithm to the one proposed by [81], the most recent DP-based approach. Finally, we show how our method handles EC-LQR with cross-time-step constraints.

7.4.1 Problem Formulation

For a robotic system with state $x_t \in \mathbb{R}^n$ and control input $u_t \in \mathbb{R}^m$, we define a state trajectory as $\mathbf{x} = [x_0, x_1, \ldots, x_T]$ and control input trajectory as $\mathbf{u} = [u_0, u_1, \ldots, u_{T-1}]$ where T is the trajectory length. The optimal control input trajectory $\mathbf{u}^*$ and its corresponding state trajectory $\mathbf{x}^*$ are the solution to the constrained linear least squares problem:

$$\min_{\mathbf{u}} x_T^T Q_{xx_T} x_T + \sum_{t=0}^{T-1} (x_t^T Q_{xx_t} x_t + u_t^T Q_{uu_t} u_t) \tag{7.2a}$$

$$\text{s.t.} \quad x_{t+1} = F_{x_t} x_t + F_{u_t} u_t \tag{7.2b}$$

$$G_{x_t} x_t + G_{u_t} u_t + g_{l_t} = 0, \quad t \in \mathcal{C} \tag{7.2c}$$

$$G_{x_T} x_T + g_{l_T} = 0 \tag{7.2d}$$

$$\sum_{i \in C_{kx}} S_{xki} x_i + \sum_{j \in C_{ku}} S_{ukj} u_j + s_k = 0 \tag{7.2e}$$

where Q_{xx_T}, Q_{xx_t}, and Q_{uu_t} are positive definite matrices defining the cost function; F_{x_t} and F_{u_t} define the system dynamics at time t; constraints (7.2c) and (7.2d) are local auxiliary constraints; and constraint (7.2e) is a new formulation for cross-time-step constraints. In (7.2c) and (7.2d), $G_{x_t} \in \mathbb{R}^{l_t \times n}$, $G_{u_t} \in \mathbb{R}^{l_t \times m}$, and $g_{l_t} \in \mathbb{R}^{l_t}$ form local constraints with constraint dimension l_t; $\mathcal{C}$ is the set of time steps where a local constraint, such as initial state constraint, applies; and G_{x_T} and g_{l_T} form a local constraint with dimension l_T on the final step. In the cross-time-step constraint (7.2e), $S_{xki} \in \mathbb{R}^{c_t \times n}$, $S_{ukj} \in \mathbb{R}^{c_t \times m}$, and $s_k \in \mathbb{R}^{c_t}$ form constraints on a set of states x_i and controls u_j where k is the index of the cross-time-step constraint. In this chapter we focus on representing quadratic cost in the factor graph, but linear terms in the cost function can be incorporated too as shown in the next section.

7.4.2 EC-LQR with Local Constraints

The factor graph representation of EC-LQR with only local constraints (7.2c) and (7.2d) in Problem 7.2 is the same as the factor graph in Figure 7-1 but without the red square marked "cross-time-step constraint". We still use the same elimination order: x_2, u_1, x_1, u_0, x_0 to execute VE.

Eliminating a state

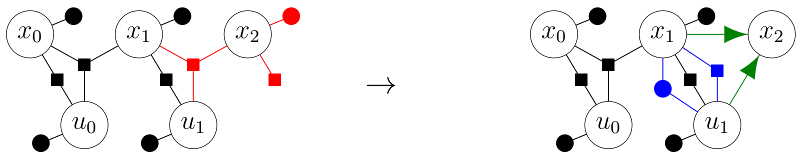
The process for eliminating a state involves one more constraint when generating $\psi^*_{x_i}(S_{x_i})$, but solving for x_i remains the same as in standard LQR case. Figure 7-4a shows the process of eliminating x_2.

Eliminating a control

The process for eliminating a control is a constrained minimization with some constraints on u_i derived from $\psi^*_{x_{i+1}}(u_i, x_i)$ and/or $G_{x_i}x_i + G_{u_i}u_i + g_{l_i} = 0$. The elimination procedure is shown in Figure 7-4b. From the result of eliminating u_1 as shown on the right in Figure 7-4b, we observe that

- the optimal control policy $u^*_1(x_1) = -K_1x_1 + k_1$ falls out,
- $\phi^*_{u_1}(x_1) = ||P_1^{1/2}x_1 - p_1||^2$ corresponds to the *cost_to_go*$(x_1) = x_1^TV_1x_1 - v_1x_1$ from [81] where $V_1 = P_1 + Q_{xx_1}$ and $v_1 = 2p_1^TP_1$, and
- $\psi^*_{u_1} = H_1x_1 - h_1 = 0$ corresponds to the *constraint_to_go*$(x_1) = H_1x_1 - h_1 = 0$ from [81]

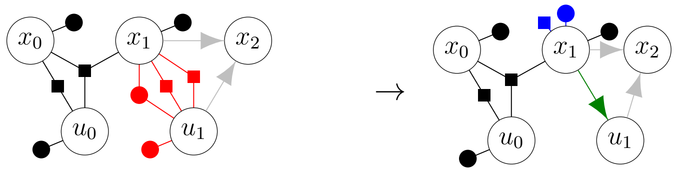
We continue with VE to eliminate the remaining variables similarly. After each u_i is eliminated, we can obtain an optimal control policy, *constraint_to_go*, and



$$\begin{aligned} x_2^*(x_1,u_1) = \arg\min_{x_2}\ & x_2^T Q_{xx_T} x_2 \\ \text{s.t. } & G_{x_2}x_2 - g_{l_2} = 0 \\ & x_2 - F_{u_1}u_1 - F_{x_1}x_1 = 0 \end{aligned} \quad \rightarrow \quad \begin{aligned} x_2^*(u_1,x_1) &= F_{x_1}x_1 + F_{u_1}u_1 \\ \phi_{x_2}^*(u_1,x_1) &= ||F_{x_1}x_1 + F_{u_1}u_1||^2_{Q_{xx_T}} \\ \psi_{x_2}^*(u_1,x_1) &= G_{x_2}F_{u_1}u_1 + G_{x_2}F_{x_1}x_1 - g_{l_2} = 0 \end{aligned}$$

$$\begin{array}{c} W_{x_2} \\ \begin{bmatrix} I \\ \infty \\ \infty \end{bmatrix} \end{array} \begin{array}{ccc|c} x_2 & u_1 & x_1 & \\ Q^{\frac{1}{2}}_{xx_T} & & & 0 \\ G_{x_2} & & & g_{l_2} \\ I & -F_{u_1} & -F_{x_1} & 0 \end{array} \rightarrow \begin{array}{c} W'_{x_2} \\ \begin{bmatrix} \infty \\ I \\ \infty \end{bmatrix} \end{array} \begin{array}{ccc|c} x_2 & u_1 & x_1 & \\ I & -F_{u_1} & -F_{x_1} & 0 \\ 0 & Q^{\frac{1}{2}}_{xx_T}F_{u_1} & Q^{\frac{1}{2}}_{xx_T}F_{x_1} & 0 \\ 0 & G_{x_2}F_{u_1} & G_{x_2}F_{x_1} & g_{l_2} \end{array}$$

(a) Eliminate x_2



$$\begin{aligned} u_1^*(x_1) = \arg\min_{u_1}\ & \phi_{x_2}^*(x_1,u_1) + u_1^T R u_1 \\ \text{s.t. } & G_{x_2}F_{u_1}u_1 - G_{x_2}F_{x_1}x_1 - g_{l_2} = 0 \\ & G_{u_1}u_1 - G_{x_1}x_1 - g_{l_1} = 0 \end{aligned} \quad \rightarrow \quad \begin{aligned} u_1^*(x_1) &= -K_1x_1 + k_1 \\ \phi_{u_1}^*(x_1) &= ||P_1^{\frac{1}{2}}x_1 - p_1||^2 \\ \psi_{u_1}^*(x_1) &= H_1x_1 - h_1 = 0 \end{aligned}$$

$$\begin{array}{c} W_{u_1} \\ \begin{bmatrix} I \\ \infty \\ I \\ \infty \end{bmatrix} \end{array} \begin{array}{cc|c} u_1 & x_1 & \\ Q^{\frac{1}{2}}_{xx_T}F_{u_1} & Q^{\frac{1}{2}}_{xx_T}F_{x_1} & 0 \\ G_{x_2}F_{u_1} & G_{x_2}F_{x_1} & g_{l_2} \\ Q^{\frac{1}{2}}_{uu_1} & & 0 \\ G_{u_1} & G_{x_1} & g_{l_1} \end{array} \rightarrow \begin{array}{c} W'_{u_1} \\ \begin{bmatrix} R_1 \\ \infty \\ I \end{bmatrix} \end{array} \begin{array}{cc|c} u_1 & x_1 & \\ I & K_1 & k_1 \\ 0 & H_1 & h_1 \\ 0 & P_1^{\frac{1}{2}} & p_1 \end{array}$$

(b) Eliminate u_1

Figure 7-4: Two elimination steps for EC-LQR with local constraints. This figure has the same layout as Figure 7-3.

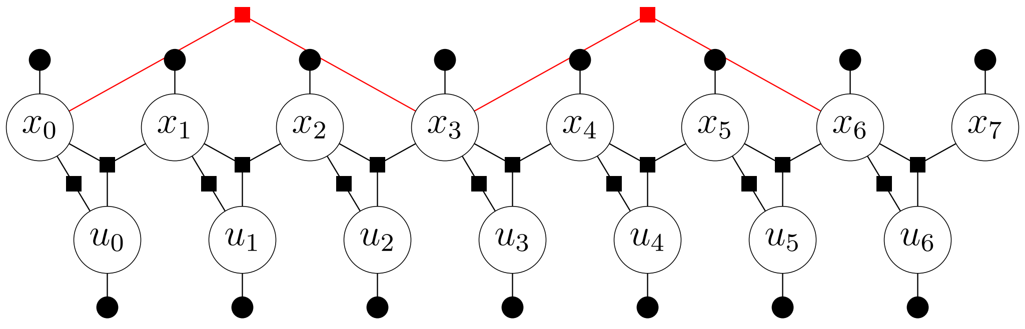
$cost\_to\_go$ – all of which being functions of x_i. When the problem is linear and all matrices are invertible or full column rank, the optimal solution is unique. We will demonstrate our method finding the unique optimal solution in Section 7.5.

7.4.3 Computational Complexity Analysis

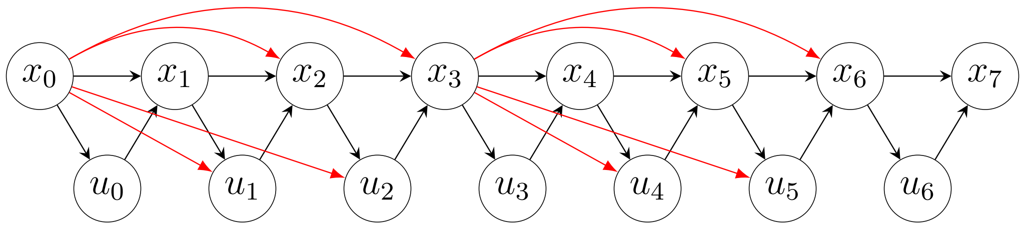
Because Step 1 collects only the factors connected to the variable we seek to eliminate, VE is very efficient and the complexity of eliminating a single variable is independent of the trajectory length. When eliminating one variable, we factorize a matrix, A_i, whose rows consist of all the factors connected to the variable and whose columns correspond to the variable and its separator. Thus, the maximum dimension of A_i in EC-LQR problem with just local constraints is $3n \times (2n+m)$ when eliminating a state or $(2n+m) \times (n+m)$ when eliminating a control. In the worst case, the QR factorization on this matrix has complexity $O(2(3n)(2n+m)^2) = O(24n^3 + 24n^2m + 6nm^2)$ when eliminating a state or $O(2(2n+m)^2(n+m)) = O(8n^3 + 16n^2m + 10nm^2 + 2m^3)$ when eliminating a control. To obtain the solution from the sparse QR factorization result of A, we apply back substitution whose computation complexity is $O(n^2 + m^2)$, so the overall computation complexity of solving the trajectory with length T is $O(T \cdot (\kappa_1 n^3 + \kappa_2 n^2 m + \kappa_3 n m^2 + \kappa_4 m^3))$, which is the same as the state of the art DP approach [81].

7.4.4 EC-LQR with Cross-time-step Constraints

The factor graph's ability to add factors on any set of variables allows us to add more general auxiliary constraints and objectives than [81], such as cross-time-step constraints. Note that cross-time-step *objectives* could also be handled the same way if desired. The VE algorithm for solving EC-LQR with cross-time-step constraints



(a) Factor graph



(b) The Bayes Net after Variable Elimination

Figure 7-5: Example cross-time-step constraint in a factor graph. The bottom figure is a Bayes net showing variable dependencies after VE.

(or even objectives) remains exactly the same as in Section 7.4.2. For example, in Fig. 7-5, the cross-time-step constraint is $Sx_{n_c+p} + Sx_{n_c} + s = 0$. When eliminating x_{n_c+p}, its separator will contain x_{n_c+p-1}, u_{n_c+p-1} and x_{n_c}. After elimination of x_{n_c+p}, the new *constraint_to_go* factor will be connected to not only x_{n_c+p-1} and u_{n_c+p-1}, but also x_{n_c}. Subsequent elimination steps will generate similar factors. As a result, after all variables are eliminated, the final feedback controllers for control inputs between x_{n_c+p} and x_{n_c} are functions of two states instead of just the current state. Fig. 7-5b illustrates the result in the form of a Bayes Net [36] where arrows represent the variable dependencies.

We further show our method maintains linear complexity with the length of the trajectory. Notice in Fig. 7-5b that each cross-time-step constraint spanning from time step t_a to t_b adds additional dependencies of variables x_k, u_k ($t_a < k \leq t_b$) on variables associated with the cross-time-step constraint. Therefore, as long as the maximum number of variables associated with a cross-time-step constraint is bounded by d, and the maximum number of cross-time-step constraints spanning over any time step is bounded by q, the number of variables involved in any elimination step (which contribute to the κ constants) is bounded by $3 + (d - 1) \times q$ thereby bounding the complexity of each elimination operation.

7.5 Experiments

We run simulation experiments to demonstrate the capability of the proposed method[2]. We implement our method using the Georgia Tech Smoothing And Mapping (GT-SAM) toolbox [35]. We compare our approach with three baseline methods. Baseline method 1 is [133], Baseline method 2 is [81], and Baseline method 3 is using Mat-

[2]Source code is available on Github

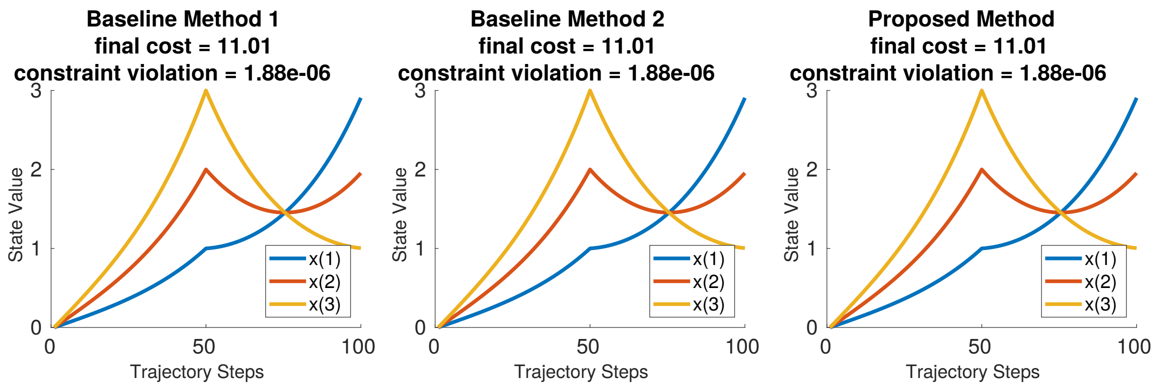


Figure 7-6: Optimal trajectory, cost, and constraint violation comparison of three methods for Problem 7.3. For each method we plot the three dimensions of the state x. All methods produce the same result.

lab's `quadprog` quadratic programming solver (which does not produce an optimal control *policy*). We first present comparison experiments for EC-LQR on random systems. We then show our approach handling cross-time-step constraints on an example system motivated by a single leg hopping robot.

7.5.1 Cost, Constraint Violation & Controller Comparison

The first experiment is to find the optimal trajectory for a simple system with $x_i \in \mathbb{R}^3$ and $u_i \in \mathbb{R}^3$ that is subject to state constraints. The EC-LQR problem is given by:

$$\begin{aligned} \min_{\mathbf{u}} \ & (x_T - x_N)^T Q_{xx_T}(x_T - x_N) + \sum_{t=0}^{T-1}(x_t^T Q_{xx_t} x_t + u_t^T Q_{uu_t} u_t) \\ \text{s.t.} \ \ & x_{t+1} = F_x x_t + F_u u_t, \quad x_0 = [0 \ \ 0 \ \ 0]^T, \\ & x_N = [3 \ \ 2 \ \ 1]^T, \quad x_{T/2} = [1 \ \ 2 \ \ 3]^T \end{aligned} \tag{7.3a}$$

where $dt = 0.01$, $F_x = I_{3\times3} + I_{3\times3} \cdot dt$, $F_u = I_{3\times3} \cdot dt$, $T = 100$, $Q_{xx_t} = 0.01 \cdot I_{3\times3}$, $Q_{uu_t} = 0.001 \cdot I_{3\times3}$, and $Q_{xx_T} = 500 \cdot I_{3\times3}$. In this case $\mathcal{C} = \{0, T/2\}$.

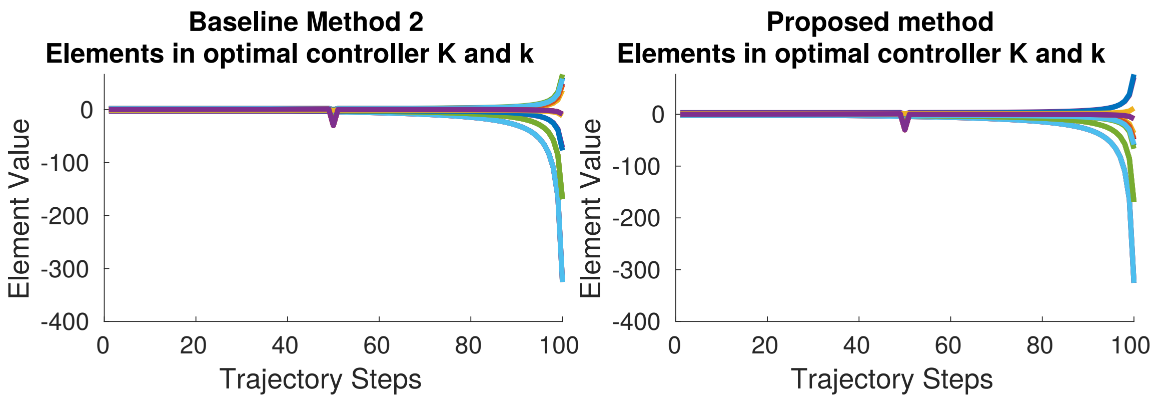


Figure 7-7: The plots of feedback control gain matrices from Baseline Method 2 and ours (we omit Baseline 1 because its result is identical to Baseline 2). Each curve represents one element in K_t or k_t.

Fig. 7-6 compares the optimal state trajectories using three methods. Baseline 3 is omitted for space reasons, but all three baselines and our method arrive at the exact same solution, with 0 constraint violation and identical total cost, as expected since the optimal solution is unique.

To show our method can also handle state and control local constraints, we replace the last state-only constraint (7.3a) to be a constraint that contains both the state and the control as $x_{N/2} + u_{N/2} + [1 \quad 2 \quad 3]^T = 0$. We solve this problem to get the optimal controllers $u_t = -K_t x_t + k_t$. K_t and k_t are identical among Baseline 1, Baseline 2 and ours. Fig. 7-7 omits Baseline 1 for space reasons. Baseline 3 does not produce a controller.

7.5.2 Run Time Comparison

We focus on comparing our method and Baseline 2 since Baseline 2 is the only baseline that has linear complexity and generates a feedback policy. Both methods are implemented in C++ and tested on a computer with an Intel i7-8809G 3.10GHz

CPU. We generate random problems with given sizes and compare average run times over 10 trials. With $l_t = m - 1$ dimensional local constraints at every time step, we first fix $n = m = 3$ and vary trajectory length T:

T	100	200	300	400	500	600
[81] (ms)	**0.88**	**1.06**	**1.67**	**2.01**	**2.35**	**2.81**
Ours (ms)	2.32	3.17	4.30	4.68	5.86	6.86

then we fix $T = 100$ and increase n and m together:

n, m	10	20	30	40	50	60
[81] (ms)	**3.74**	14.5	44.1	83.5	152.3	247.7
Ours (ms)	3.81	**11.8**	**27.1**	**51.2**	**99.0**	**170.2**

The experiments show that for both methods, run time grows linearly with increasing trajectory length as expected. Our method performs better for larger state and control dimensions. We believe this behavior is attributable to QR factorization being faster than SVD (used in Baseline 2), which overcomes the graph overhead for large m.

7.5.3 Cross-time-step Constraints

To illustrate an example of how cross-time-step constraints can be used to generate useful trajectories, we use a double integrator system ($x_i = [\text{position}; \text{velocity}], u =$ acceleration) with periodic "step placements". Consider the x-coordinate of a hopping robot's foot which initially starts in contact with the ground and makes contact with the ground again every 20 time steps. Each contact, it must advance forward by 0.6 units and match the ground velocity (which may be non-zero e.g. on a moving walkway). The problem is given by:

$$\min_{\mathbf{u}} x_T^T Q_{xx_T} x_T + \sum_{t=0}^{T-1} (x_t^T Q_{xx_t} x_n + u_t^T Q_{uu_t} u_t) \tag{7.4a}$$

$$\text{s.t.} \quad x_{t+1} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ dt \end{bmatrix} u_t, \quad x_0 = [0 \;\; 0]^T, \tag{7.4b}$$

$$x_{n_c+20} - x_{n_c} = \begin{bmatrix} 0.6 & 0 \end{bmatrix}^T, \quad n_c = 0, 20, 40, 60, 80 \tag{7.4c}$$

The cross-time-step constraints (7.4c) enforce that contacts must occur at a fixed position relative to and with the same velocities as the previous contacts $p = 20$ time steps prior. These create constraint factors between two state variables $p = 20$ time steps apart, as in Fig. 7-5 ($p = 3$ in Fig. 7-5).

Fig. 7-8 shows the solutions to Problem 7.4 using Baseline 2 [81], Baseline 3 (QP), and our method, as well as the results when using the same controllers with a perturbed initial state $x_0 = [0 \;\; 1.8]^T$ (i.e. walking on a moving walkway with velocity 1.8). We omit Baseline 1 from the Figure for space reasons since it performs identically to Baseline 2. We apply some modifications to allow for comparison since Baselines 1 and 2 cannot natively handle cross-time-step constraints and Baseline 3 cannot generate an optimal policy, but even so, the adjusted baselines do not generate optimal trajectories from perturbed initial state, as shown in Fig. 7-8 (bottom). For Baseline 2, we convert the cross-time-step constraints to same-time-step constraints $x_{n_c} = [0.03n_c \; 0]^T$ for $n_c = 0, 20, \ldots$ resulting in incorrect constraints after perturbing the initial state. An alternative would be to introduce 10 additional state dimensions (two for each cross-time-step constraint) analagous to Lagrange multipliers, but we argue that such an approach is not sustainable for online operation and many cross-time-step constraints. For Baseline 3, we re-use the control sequence from Problem

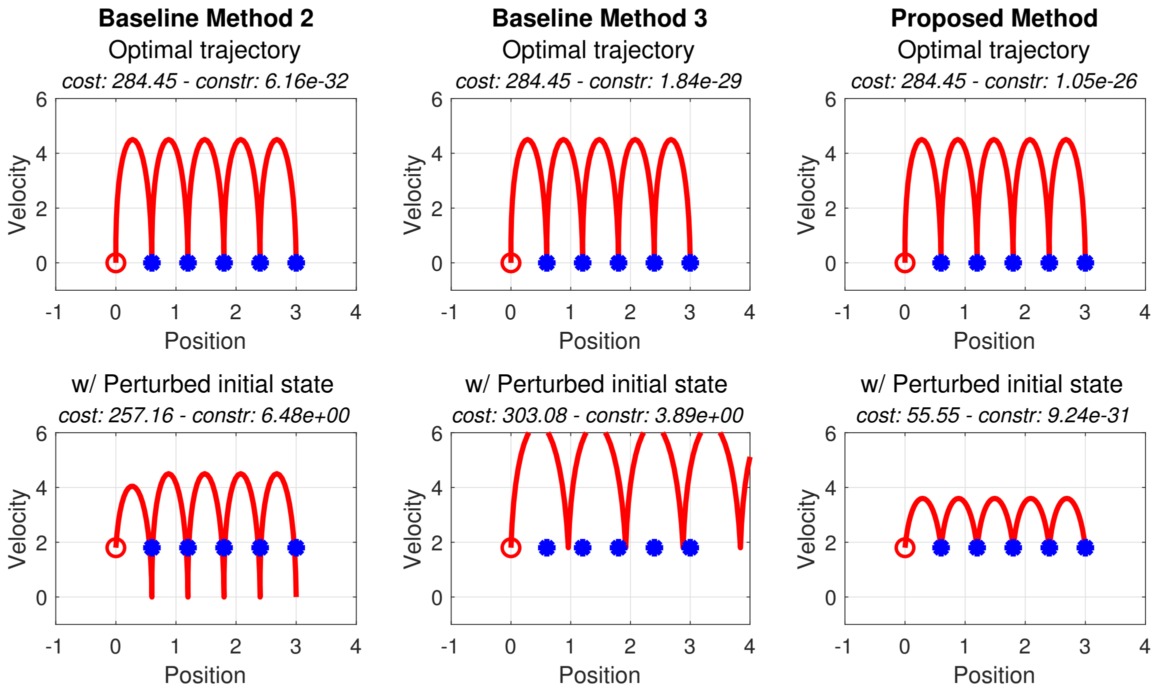


Figure 7-8: The state trajectories solving Problem 7.4 using Baseline method 2 (left), Baseline method 3 (middle), and our proposed method (right) with control sequence/policies applied to the original problem (top) and after perturbing the initial state (bottom). All methods generate the same trajectory to the initial problem, but only ours gives a policy which generates the optimal trajectory for the perturbed problem. "Cost" and "Constr" denote the total objective cost and constraint violation, respectively.

7.4 for the perturbed case. Our method's control law produces a state trajectory that is optimal and without constraint violation even with a perturbed initial state an shown in Fig. 7-8 (bottom right).

7.6 Future Work

Just as LQR is a building block for Differential Dynamic Programming (DDP) [88, 98], linear factor graphs could also be a building block for more general nonlinear optimal control problems. In this direction, the following practical developments should be investigated: incorporating inequality constraints e.g. using barrier or penalty functions [49]; extending to nonlinear systems using nonlinear factor graphs [36]; addressing over-constrained "constraints" in VE via prioritization of constraints; leveraging incremental solving using Bayes Trees [69] to do efficient replanning; and combining estimation and optimal control into the same factor graph to better close the perception-control loop.

7.7 Conclusions

In this chapter, we proposed solving equality constrained linear quadratic regular problems using factor graphs. We showed that factor graphs can represent linear quadratic optimal control problems with auxiliary constraints by capturing the relationships amongst variables in the form of factors. Variable elimination, an algorithm that exploits matrix sparsity to optimize factor graphs, is used to efficiently solve for the optimal trajectory and feedback control policy. We demonstrated that our approach can handle more general constraints than traditional DP approaches while also matching or exceeding state-of-the-art performance with traditional constraints. We believe our method has great potential to be used in a number of complex robotics systems which require solving more general constrained optimal control problems.

7.8 Appendix

We provide a probabilistic view of Figure 7-2 to show its connection to MPC and VILO discussed in previous chapters. If we treat $X = [x_2; u_1; x_1; u_0; x_0]$ as a stack of random variables, before connecting variables with edges it has a prior distribution

$p(X)$. We then define three types of likelihood functions [38]:

$$l_{objx}(x_t; \mathcal{Q}_t) \propto P(\mathcal{Q}_t|x_t) = \exp\Big\{ -\frac{1}{2}\|x_t\|^2_{Q_{xx_t}} \Big\}, \tag{7.5}$$

$$l_{obju}(u_t; \mathcal{R}_t) \propto P(\mathcal{R}_t|u_t) = \exp\Big\{ -\frac{1}{2}\|u_t\|^2_{Q_{uu_t}} \Big\}, \tag{7.6}$$

$$\begin{aligned} l_{dyn}(x_{t+1}, x_t, u_t; \mathcal{F}_t) &\propto P(\mathcal{F}_t|x_{t+1}, x_t, u_t) \\ &= \exp\Big\{ -\frac{1}{2}\|x_{t+1} - F_{x_t}x_t + F_{u_t}u_t\|^2_{\Sigma_c} \Big\}, \end{aligned} \tag{7.7}$$

where $\|\cdot\|_\Sigma$ is the Mahalanobis norm, and Σ is the covariance of the distribution. Notice $\Sigma_c \to 0$. As the definition suggests, a likelihood function measures the probability of elements in the random variable X on condition that a "factor event" such as $\mathcal{Q}$ be 0. It's not hard to see the correspondences between edges in Figure 7-2 and likelihood functions. Define all factor events as $Z = [\mathcal{F}_1, \mathcal{F}_2, \mathcal{Q}_0, \mathcal{Q}_1, \mathcal{Q}_2, \mathcal{R}_0, \mathcal{R}_1]$, the joint probability distribution of X and Z is essentially

$$\begin{aligned} p(X, Z) =& p(Z|X)p(X) \\ =& p(x_0, u_0, x_1, u_1, x_2, \mathcal{F}_1, \mathcal{F}_2, \mathcal{Q}_0, \mathcal{Q}_1, \mathcal{Q}_2, \mathcal{R}_0, \mathcal{R}_1) \\ =& p(\mathcal{Q}_0|x_0)p(\mathcal{Q}_1|x_1)p(\mathcal{F}_1|x_0, u_0, x_1)p(\mathcal{F}_2|x_1, u_1, x_2) \\ & p(\mathcal{R}_0|u_0)p(\mathcal{R}_1|u_1)p(x_0, u_0, x_1, u_1, x_2) \end{aligned} \tag{7.8}$$

The last expansion is done using the Bayes Theorem. Apply the Bayes Theorem again, and assume the prior distribution of $p(X)$ is uniform, we can conclude that the *maximum a posteriori* (MAP) distribution of X follows [36]

$$X^{MAP} = \arg\max_X \; p(Z|X) = \arg\max_X \; l(X; Z) = \arg\max_X \; \phi(X)$$

where $l(X;Z)$ is the product of likelihood functions obtained by rewriting conditional probability terms in Eq. (7.8) to their counterparts in (7.5), (7.6), or (7.7). $\phi(X)$ is a simplified notation defined as $l(X;Z) = \phi(X) = \prod \phi_i(X_i)$. ϕ_i is still the likelihood function we defined earlier, and X_i is a set of elements of X that relates to the function ϕ_i. We call a ϕ_i as a factor.

The key procedure links the probabilistic view of the factor graph and the LQR is the VE. When $X = X^{MAP}$, $\phi(X^{MAP}) \propto p(X^{MAP})$. Let $X = [\theta_0 \ldots \theta_n]$, we can factorize $p(X)$ as

$$p(X) = p(\theta_0|S_0)p(\theta_1|S_1)\ldots p(\theta_n) = \prod_j p(\theta_j|S_j). \tag{7.9}$$

The S_j is called *separator* [12], which is defined as the set of variables that θ_j is conditioned after the factorization. This factorization process is the VE. Its final form depends on the *elimination order*, which is the order how elements in X got picked during VE. More importantly, in the case of X contains states and control inputs, when control u_i in X is picked, **the factorized term $p(u_i|S_i)$ is the optimal feedback controller**.

To better present the detailed steps of VE, we convert the probabilistic view to a matrix view using the fact that our likelihood functions are Gaussian distributions. From Definition (7.5), (7.6), and (7.7), we can see all likelihood functions or factors $\phi_i(X_i)$ can be written as

$$\phi_i(X_i) \propto \exp\Big\{ -\frac{1}{2}\|A_iX_i - b_i\|^2_{\Sigma_i}\Big\}$$

Then

$$\begin{aligned} X^{MAP} &= \arg\max_X \ \phi(X) = \arg\min_X \ -2\log(\prod \phi_i(X_i)) \\ &= \arg\min_X \ \sum_i \|A_iX_i - b_i\|^2_{\Sigma_i} = \arg\min_X \ \|AX-b\|^2_\Sigma \end{aligned} \tag{7.10}$$

where A, b and Σ stack all A_i, b_i, and Σ_i together respectively. Then the solution of X^{MAP} is a linear least square problem. A caveat is the covariance matrix in (7.7) is 0 because the system dynamics constraints must be strictly obeyed. So some terms in Σ are zero. If we expand the Mahalanobis norm in (7.10) and define weight matrix $W = \Sigma^{-1/2}$, we have $(AX-b)^TW(AX-b)$ with some infinite weights. This weighted linear least square system can be solved by the modified Gram-Schmidt algorithm [50].

Using the matrix view, VE has a direct connection to sparse matrix factorization [36]. If we factorize matrix A in (7.10) as $A = QR$ using the modified Gram-Schmidt, where Q is orthonormal and R is upper-triangular. Then

$$\min_X \|AX-b\|^2_\Sigma = \min_X Q^T\|AX-b\|^2_\Sigma = \min_X \|RX-d\|^2_\Sigma + \|e\|^2_\Sigma$$

where $d = Q^Tb$. The last term $\|e\|^2_\Sigma$ appears as a residual if R have rows with all 0s on the bottom part. Now each block row of $\|RX-d\|^2_\Sigma$ corresponds to one term in (7.9) because R is upper-triangular. We shall emphasize that the factorization of A can be done by doing the modified Gram-Schmidt on just one column at a time. Since ith column of A corresponds to variable θ_i in X, we "eliminate" it. Moreover, A is sparse in the optimal control setting, so the elimination can be done very efficiently. This observation leads to the following VE steps to eliminate θ_i:

Step 1. Identify all the factors directly connected to θ_i (all block rows in A with nonzero elements on block column i). We also extract rows' weights from the weight matrix W. Block rows with infinite weights are constraint factor $\psi_i(\theta_i, S_i)$, while block rows with finite weight are objective factor $\phi_i(\theta_i, S_i)$. Now all these block rows formulate $\|A_i[\theta_i; S_i] - b_i\|^2_{\Sigma_i}$ We can normalize finite weights to be 1 by multiplying $\Sigma_i^{-1/2}$ on A_i and b_i.

Step 2. Factorize $[A_i|b_i]$ to $[R_i|d_i]$ using the modified Gram-Schmidt on its first column. In the case of quadratic costs and linear system dynamics constraints, we can treat it as a constrained least square problem and solve manually, as we show in Figure 7-3.

Step 3. Output the first block row of $[R_i|d_i]$ as $p(\theta_i|S_i)$. Add the rest blocks of $[R_i|d_i]$ to replace corresponding blocks in $[A_i|b_i]$.

We repeat this process until all variables have been eliminated. More details on the VE algorithm are in [36].

Chapter 8

Conclusion & Future Work

Here is the revised version of your paragraph with improved grammar and conciseness, while retaining the original meaning and incorporating LaTeX tags:

In this thesis, we systematically explored model predictive control, the Kalman filter, sliding window estimation, and their interconnections, with applications to legged locomotion. The reaction wheel MPC enabled a novel legged robot platform to demonstrate superior stability and new locomotion behaviors. Online kinematic calibration studies examined how locomotion-related error sources can affect Kalman filter-based odometry accuracy and how to mitigate them. The Kalman filter is closely connected to sliding window state estimation because they fundamentally share the same numerical optimization problem structure. Based on sliding window estimation, we propose a novel visual-inertial-leg odometry algorithm using a factor graph that can achieve low-drift, long-term state estimation. We then focus on introducing more sensors to the legged robot odometry sensor suite to further improve estimation accuracy and robustness, particularly by utilizing more compact sensors such as IMUs on the robot's feet. We also develop the first multi-IMU visual-inertial odometry algorithm capable of achieving low-drift, long-term state estimation during long-distance rough terrain locomotion. Moreover, we experimented with how control-related parameters could affect estimation system performance, providing guidance for legged robot controller tuning.

Following these contributions, we solve the equality-constrained LQR problem using the factor graph, highlighting an important connection between control and state estimation by demonstrating that the equality-constrained LQR problem can be solved using the same numerical optimization algorithm as sliding window state estimation.

Our design philosophy, which emphasizes mimicking human and animal shapes without constraining ourselves to them, could guide many future robotics system designs. We use the most compact and effective solutions to address fundamental limitations and performance issues, even if the solutions do not resemble animal shapes. The usage of reaction wheels and additional IMUs are good examples of this philosophy.

The methods proposed in this thesis all come with not only mathematical formulations but also open-source C++ or Matlab implementations, hosted on Github with permanent addresses. Considerable software engineering work is done to make sure they can be used easily by the research community. The Cerberus and Cerberus 2.0 also has Docker and ROS interfaces that allow them to be integrated into actual robot hardware directly.

Finally, we would like to discuss some future work directions that can be built upon the work in this thesis.

8.1 Legged Control

As we discussed in Chapter 1.1, MPC and RL have become the two dominant methods in legged robot control. In both cases, the controller hierarchy includes an MPC/RL-based local planner that translates high-level user commands into robot task space commands, followed by a low-level controller that converts task space commands into joint space commands. The success of this hierarchy on quadruped robots is partly due to the simple structure of the robot dynamics and the exceptional torque control performance of direct drive motors [157]. If the robot motor dynamics are complicated, transferring an MPC/RL algorithm from simulation to hardware is much more difficult [62]. For legged robots and humanoid robots with even more complex dynamics, it is not yet clear if such a hierarchy will work very well. It is likely that tuning the low-level controller will require many classical control theory techniques to counter disturbances from the external environment and the robot structure. Additionally, since the low-level control must run at 500Hz-2000Hz, the frequency requirement makes it challenging for the low-level controller to be replaced by function approximators, which typically struggle to achieve such fast inference frequencies.

8.2 Legged Estimation

The estimation techniques discussed in this thesis provide a comprehensive introduction to model-based methods. Several sensor suite formulations are studied, which can be readily applied to all types of legged robots. As we pointed out, improving algorithms is usually not as effective as adding more sensors to the system. Since multi-IMU PO performs much better than single IMU PO, a very interesting question naturally arises: what if we add one IMU per rigid link on the robot? From an engineering perspective, this is not hard to achieve. On complicated legged systems such as full-size humanoid robots, this could result in 30 or more IMUs being used. According to estimation theory, the estimation accuracy will improve until the estimation covariance equals the sensor-level signal covariance divided by the number of sensors. However, approaching this limit while maintaining a low estimation computation cost may be difficult. Nonetheless, it is worth exploring, and a general multi-link-multi-sensor estimation framework may be developed.

8.3 Legged System Hardware Development

In recent years, as researchers continue to expand the capabilities of legged robots, our attention has inevitably been drawn to humanoid robots. Although we do not constrain ourselves to human and animal shapes, increasing evidences suggest that moving towards a human shape could benefit robot performance. While the reaction wheel is a compact solution that can improve a legged robot's controllability, it is still not very cost-effective: the wheel apparatus can only improve stability and nothing else. A better solution could be adding arms [8], which have already been shown to help humanoid robots balance quite well, or let the robot stands up [28]. Alternatively, researchers have also explored flat feet designs [151] for legged robots to mitigate the controllability issues caused by point feet. Interestingly, this makes four legs seem redundant, and humanoid robots may prove to be more cost-effective. Nevertheless, the study of legged robots is a stepping stone toward developing more powerful humanoids.

8.4 Legged System Software Development

Here is the revised version of your paragraph with improved grammar and conciseness, while retaining the original meaning and incorporating LaTeX tags:

From a system perspective, we believe there are two areas that need more attention when developing a complete legged system.

The first is developing solvers and problem construction infrastructure that simplify constructing both control and estimation problems. In a practical legged robot system, the MPC runs in one process using OSQP [138] to solve the control problem, while its estimator runs in another process using a Gauss-Newton gradient descent solver. Each process includes functions that can be reused to speed up computation, such as forward kinematics, Jacobians, and contact signal generation. If both processes can use the same solver and a shared set of kinematic functions that cache computation results, they will be more efficient.

The second is considering control parameter and estimation parameter selections together in a more systematic way. In Chapter 6 we experimentally showed that certain control parameters affect estimation accuracy. A further step could be explicitly modeling these parameters in both estimation and control problems to understand how their variations affect the characteristics of both problems.

8.5 Data Driven Methods

Cerberus 2.0 considers most error sources in legged locomotion that can be captured by explicit models; however, there are still unmodeled dynamics that are difficult to capture with hand-designed models. Therefore, data-driven methods should be pursued to further improve performance. For example, on the control side, data-driven methods can be used to learn the model of the robot dynamics, the cost function of the MPC, and a terrain representation that can be converted to MPC constraints. On the estimation side, all sensor measurement models can be learned from data if the ground truth state is available.

Although I believe that data-driven methods will be the future of legged locomotion control and estimation, numerical optimization-based methods should still play an important role in many cases. Even with learned robot dynamics or sensor models, control or estimation problem structures are usually well captured by explicit numerical optimization programs. This representation is powerful enough to adapt to changes in robot dynamics, the number and types of sensors, and task details, making it easy to develop controllers and estimators for various robotic systems and tasks. Additionally, numerical optimization allows users to analyze the solution's sensitivity to problem parameters by examining the gradient or Hessian matrices at the solution. This enables the formulation of problems with increased robustness through proper reformulation.

Therefore, an ideal combination would be, for example, in the estimation case, to still use the factor graph as the backbone of the estimator. Instead of manually coding the factor functions and their Jacobians, we can use learning-based function approximators to represent them. This approach allows the overall problem to be understood through numerical optimization, while the function approximators compensate for model mismatches.

In conclusion, as legged robot research reaches a mature stage where multiple real-world applications are possible, various algorithmic and hardware innovations can further improve legged robot robustness. We have discussed established standard approaches for legged control and estimation, and then demonstrated how adding seemingly incremental changes to the system can result in significant improvements in robot performance. We believe this methodology applies not only to legged robots but also to all similar robotic systems that need to leverage multiple sensors and interact with the external environment. This thesis serves as a stepping stone toward a future where humans and robots live harmoniously.

Bibliography

[1] DE Adams. Introduction to inertial navigation. *The Journal of Navigation*, 9(3):249–259, 1956.

[2] Joao S Albuquerque and Lorenz T Biegler. Decomposition algorithms for on-line estimation with nonlinear models. *Computers & chemical engineering*, 19(10):1031–1039, 1995.

[3] Simon L Altmann. *Rotations, quaternions, and double groups*. Courier Corporation, 2005.

[4] Karl J Astrom. *Introduction to stochastic control theory*. Elsevier, 1971.

[5] Jared B Bancroft and Gérard Lachapelle. Data fusion algorithms for multiple inertial measurement units. *Sensors*, 11(7):6771–6798, 2011.

[6] Alex Barclay, Philip E Gill, and J Ben Rosen. SQP methods and their application to numerical optimal control. In *Variational calculus, optimal control and applications*, pages 207–222. Springer, 1998.

[7] Timothy D Barfoot. *State estimation for robotics*. Cambridge University Press, 2017.

[8] C Dario Bellicoso, Koen Krämer, Markus Stäuble, Dhionis Sako, Fabian Jenelten, Marko Bjelonic, and Marco Hutter. Alma-articulated locomotion and manipulation for a torque-controllable robot. In *2019 International conference on robotics and automation (ICRA)*, pages 8477–8483. IEEE, 2019.

[9] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

[10] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[11] Marko Bjelonic, Ruben Grandia, Oliver Harley, Cla Galliard, Samuel Zimmermann, and Marco Hutter. Whole-body mpc and online gait sequence generation for wheeled-legged robots. In *2021 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 8388–8395. IEEE, 2021.

[12] Jean RS Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pages 1–29. Springer, 1993.

[13] Gerardo Bledt and Sangbae Kim. Implementing regularized predictive control for simultaneous real-time footstep and ground reaction force optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6316–6323, 2019.

[14] Gerardo Bledt, Matthew J Powell, Benjamin Katz, Jared Di Carlo, Patrick M Wensing, and Sangbae Kim. Mit cheetah 3: Design and control of a robust, dynamic quadruped robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2245–2252. IEEE, 2018.

[15] Fabian Blöchliger, Michael Blösch, Péter Fankhauser, Marco Hutter, and Roland Siegwart. Foot-eye calibration of legged robot kinematics. In *Advances in Cooperative Robotics*, pages 420–427. World Scientific, 2017.

[16] Michael Bloesch. *State estimation for legged robots-kinematics, inertial sensing, and computer vision*. PhD thesis, ETH Zurich, 2017.

[17] Michael Bloesch, Christian Gehring, Péter Fankhauser, Marco Hutter, Mark A Hoepflinger, and Roland Siegwart. State estimation for legged robots on unstable and slippery terrain. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6058–6064. IEEE, 2013.

[18] Michael Bloesch, Marco Hutter, Christian Gehring, Mark A Hoepflinger, and Roland Siegwart. Kinematic batch calibration for legged robots. In *2013 IEEE International Conference on Robotics and Automation*, pages 2542–2547. IEEE, 2013.

[19] Michael Bloesch, Marco Hutter, Mark A Hoepflinger, Stefan Leutenegger, Christian Gehring, C David Remy, and Roland Siegwart. State estimation for legged robots - consistent fusion of leg kinematics and imu. *Robotics*, 17:17–24, 2013.

[20] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[21] Travis Brown and James Schmiedeler. Energetic effects of reaction wheel actuation on underactuated biped robot walking. pages 2576–2581, 05 2014.

[22] Travis L. Brown and James P. Schmiedeler. Reaction wheel actuation for improving planar biped walking efficiency. *IEEE Transactions on Robotics*, 32(5):1290–1297, 2016.

[23] Marco Camurri, Maurice Fallon, Stéphane Bazeille, Andreea Radulescu, Victor Barasuol, Darwin G Caldwell, and Claudio Semini. Probabilistic contact estimation and impact detection for state estimation of quadruped robots. *IEEE Robotics and Automation Letters*, 2(2):1023–1030, 2017.

[24] Marco Camurri, Milad Ramezani, Simona Nobili, and Maurice Fallon. Pronto: A multi-sensor state estimator for legged robots in real-world scenarios. *Frontiers in Robotics and AI*, 7:68, 2020.

[25] Paul Chauchat, Axel Barrau, and Silvere Bonnabel. Factor graph-based smoothing without matrix inversion for highly precise localization. *IEEE Transactions on Control Systems Technology*, 29(3):1219–1232, 2020.

[26] Chi-Tsong Chen. *Linear System Theory and Design.* Oxford University Press, 1999.

[27] Gerry Chen and Yetong Zhang. LQR control using factor graphs. `https://gtsam.org/2019/11/07/lqr-control.html`. Accessed: 2020-09-13.

[28] Xuxin Cheng, Ashish Kumar, and Deepak Pathak. Legs as manipulator: Pushing quadrupedal agility beyond locomotion. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5106–5112. IEEE, 2023.

[29] Matthew Chignoli and Patrick M. Wensing. Variational-based optimal control of underactuated balancing for dynamic quadrupeds. *IEEE Access*, 8:49785–49797, 2020.

[30] Simon Le Cleac'h, Taylor Howell, Mac Schwager, and Zachary Manchester. Fast contact-implicit model-predictive control, 2021.

[31] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

[32] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[33] Alexander Cunningham, Manohar Paluri, and Frank Dellaert. Ddf-sam: Fully distributed slam using constrained factor graphs. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3025–3030. IEEE, 2010.

[34] Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 295–302. IEEE, 2014.

[35] Frank Dellaert. Factor graphs and GTSAM: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.

[36] Frank Dellaert, Michael Kaess, et al. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 6(1-2):1–139, 2017.

[37] Jared Di Carlo, Patrick M. Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Dynamic locomotion in the MIT cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE.

[38] Jing Dong, Mustafa Mukadam, Frank Dellaert, and Byron Boots. Motion planning as probabilistic inference using gaussian processes and factor graphs. In *Robotics: Science and Systems*, volume 12, page 4, 2016.

[39] Boston Dynamics. Spot. `https://www.bostondynamics.com/spot`, 2021. [Online; accessed 10-Sep-2021].

[40] Kevin Eckenhoff, Patrick Geneva, and Guoquan Huang. Mimc-vins: A versatile and resilient multi-imu multi-camera visual-inertial navigation system. *IEEE Transactions on Robotics*, 37(5):1360–1380, 2021.

[41] Fabio Elnecave Xavier, Guillaume Burger, Marine Petriaux, Jean-Emmanuel Deschaud, and Francois Goulette. Multi-IMU Proprioceptive State Estimator for Humanoid Robots. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2023)*, Detroit, United States, October 2023. IEEE.

[42] Farbod Farshidian et al. OCS2: An open source library for optimal control of switched systems. [Online]. Available: `https://github.com/leggedrobotics/ocs2`.

[43] Farbod Farshidian, Edo Jelavic, Asutosh Satapathy, Markus Giftthaler, and Jonas Buchli. Real-time motion planning of legged robots: A model predictive control approach. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 577–584. IEEE, 2017.

[44] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. Georgia Institute of Technology, 2015.

[45] Mason A. Peck Frederick A. Leve, Brian J. Hamilton. *Spacecraft Momentum Control Systems.* Springer, 2015.

[46] Mohanarajah Gajamohan, Michael Merz, Igor Thommen, and Raffaello D'Andrea. The cubli: A cube that can jump up and balance. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3722–3727. IEEE.

[47] Fabian Girrbach, Manon Kok, Raymond Zandbergen, Tijmen Hageman, and Moritz Diehl. Adaptive compensation of measurement delays in multi-sensor fusion for inertial motion tracking using moving horizon estimation. In *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*, pages 1–7. IEEE, 2020.

[48] Carlos Gonzalez, Victor Barasuol, Marco Frigerio, Roy Featherstone, Darwin G. Caldwell, and Claudio Semini. Line walking and balancing for legged robots with point feet. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3649–3656, 2020.

[49] Ruben Grandia, Farbod Farshidian, René Ranftl, and Marco Hutter. Feedback mpc for torque-controlled legged robots. *arXiv preprint arXiv:1905.06144*, 2019.

[50] Mårten Gulliksson. On the modified Gram-Schmidt algorithm for weighted and constrained linear least squares problems. *BIT Numerical Mathematics*, 35(4):453–468, 1995.

[51] Brian Hall. *Lie groups, Lie algebras, and representations: an elementary introduction*, volume 222. Springer, 2015.

[52] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[53] Ross Hartley, Maani Ghaffari, Ryan M Eustice, and Jessy W Grizzle. Contact-aided invariant extended kalman filtering for robot state estimation. *The International Journal of Robotics Research*, 39(4):402–430, 2020.

[54] Ross Hartley, Maani Ghaffari Jadidi, Lu Gan, Jiunn-Kai Huang, Jessy W Grizzle, and Ryan M Eustice. Hybrid contact preintegration for visual-inertial-contact state estimation using factor graphs. In *International Conference on Intelligent Robots and Systems*, pages 3783–3790, 2018.

[55] Ross Hartley, Maani Ghaffari Jadidi, Jessy W Grizzle, and Ryan M Eustice. Contact-aided invariant extended kalman filtering for legged robot state estimation. *arXiv preprint arXiv:1805.10410*, 2018.

[56] Ross Hartley, Josh Mangelson, Lu Gan, Maani Ghaffari Jadidi, Jeffrey M Walls, Ryan M Eustice, and Jessy W Grizzle. Legged robot state-estimation through combined forward kinematic and preintegrated contact factors. In *International Conference on Robotics and Automation*, pages 1–8. IEEE, 2018.

[57] Guoquan P Huang, Anastasios I Mourikis, and Stergios I Roumeliotis. Observability-based rules for designing consistent ekf slam estimators. *The International Journal of Robotics Research*, 29(5):502–528, 2010.

[58] Jeffrey Humpherys, Preston Redd, and Jeremy West. A fresh look at the kalman filter. *SIAM review*, 54(4):801–823, 2012.

[59] Marco Hutter, Christian Gehring, Dominic Jud, Andreas Lauber, C Dario Bellicoso, Vassilios Tsounis, Jemin Hwangbo, Karen Bodie, Peter Fankhauser, Michael Bloesch, et al. Anymal-a highly mobile and dynamic quadrupedal robot. In *International Conference on Intelligent Robots and Systems*, pages 38–44. IEEE, 2016.

[60] Marco Hutter, Hannes Sommer, Christian Gehring, Mark Hoepflinger, Michael Bloesch, and Roland Siegwart. Quadrupedal locomotion using hierarchical operational space control. *The International Journal of Robotics Research*, 33(8):1047–1062, 2014.

[61] Jemin Hwangbo, Carmine Dario Bellicoso, Péter Fankhauser, and Marco Hutter. Probabilistic foot contact estimation by fusing information from dynamics

and differential/forward kinematics. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3872–3878. IEEE, 2016.

[62] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[63] Vadim Indelman, Stephen Williams, Michael Kaess, and Frank Dellaert. Factor graph based incremental smoothing in inertial navigation systems. In *2012 15th International Conference on Information Fusion*, pages 2154–2161. IEEE, 2012.

[64] Intel. Intel Realsense D435. `https://www.intelrealsense.com/depth-camera-d435/`, 2022. [Online; accessed 10-Sep-2022].

[65] Brian E Jackson, Kevin Tracy, and Zachary Manchester. Planning with attitude. *IEEE Robotics and Automation Letters*, 6(3):5658–5664, 2021.

[66] Adnane Jadid, Linda Rudolph, Frieder Pankratz, and Gudrun Klinker. Utilizing multiple calibrated imus for enhanced mixed reality tracking. In *2019 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 384–386. IEEE, 2019.

[67] J Craig John et al. Introduction to robotics: mechanics and control. *Reading: Addison-Wesley*, 1989.

[68] Aaron M. Johnson, Thomas Libby, Evan Chang-Siu, Masayoshi Tomizuka, Robert J. Full, and D. E. Koditschek. *TAIL ASSISTED DYNAMIC SELF RIGHTING*, pages 611–620. WORLD SCIENTIFIC.

[69] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

[70] Shuuji Kajita, Hirohisa Hirukawa, Kensuke Harada, Kazuhito Yokoi, Shuuji Kajita, Hirohisa Hirukawa, Kensuke Harada, and Kazuhito Yokoi. Biped walking. *Introduction to humanoid robotics*, pages 105–158, 2014.

[71] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[72] Hilbert J Kappen, Vicenç Gómez, and Manfred Opper. Optimal control as a graphical model inference problem. 2009.

[73] Jonathan Kelly and Gaurav S Sukhatme. Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration. *The International Journal of Robotics Research*, 30(1):56–79, 2011.

[74] Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

[75] Joon-Ha Kim, Seungwoo Hong, Gwanghyeon Ji, Seunghun Jeon, Jemin Hwangbo, Jun-Ho Oh, and Hae-Won Park. Legged robot state estimation with dynamic contact event information. *IEEE Robotics and Automation Letters*, 6(4):6733–6740, 2021.

[76] Yeeun Kim, Byeongho Yu, Eungchang Mason Lee, Joon-ha Kim, Hae-won Park, and Hyun Myung. Step: State estimator for legged robots using a preintegrated foot velocity factor. *IEEE Robotics and Automation Letters*, 7(2):4456–4463, 2022.

[77] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154, Sendai, Japan, 2004. IEEE.

[78] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[79] Hendrik Kolvenbach, Elias Hampp, Patrick Barton, Radek Zenkl, and Marco Hutter. Towards jumping locomotion for quadruped robots on the moon. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5459–5466, 2019.

[80] Arthur J Krener and Kayo Ide. Measures of unobservability. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 6401–6406. IEEE, 2009.

[81] Forrest Laine and Claire Tomlin. Efficient computation of feedback control for equality-constrained LQR. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6748–6754. IEEE, 2019.

[82] Thomas Dall Larsen, Nils A Andersen, Ole Ravn, and Niels Kjølstad Poulsen. Incorporation of time delayed measurements in a discrete-time kalman filter, 1998.

[83] Quentin Leboutet, J Rogelio Guadarrama-Olvera, Florian Bergner, and Gordon Cheng. Second-order kinematics for floating-base robots using the redundant acceleration feedback of an artificial sensory skin. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4687–4694. IEEE, 2020.

[84] Ern J Lefferts, F Landis Markley, and Malcolm D Shuster. Kalman filtering for spacecraft attitude estimation. *Journal of Guidance, control, and Dynamics*, 5(5):417–429, 1982.

[85] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[86] Mingyang Li and Anastasios I Mourikis. High-precision, consistent ekf-based visual-inertial odometry. *The International Journal of Robotics Research*, 32(6):690–711, 2013.

[87] Mingyang Li and Anastasios I Mourikis. Online temporal calibration for camera–imu systems: Theory and algorithms. *The International Journal of Robotics Research*, 33(7):947–964, 2014.

[88] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems.

[89] Thomas Libby, Aaron M. Johnson, Evan Chang-Siu, Robert J. Full, and Daniel E. Koditschek. Comparative design, scaling, and control of appendages for inertial reorientation. 32(6):1380–1398.

[90] Pei-Chun Lin, Haldun Komsuoglu, and Daniel E Koditschek. A leg configuration measurement system for full-body pose estimates in a hexapod robot. *IEEE Transactions on Robotics*, 21(3):411–422, 2005.

[91] Kevin M Lynch and Frank C Park. *Modern robotics.* Cambridge University Press, 2017.

[92] Jeremy Ma, Max Bajracharya, Sara Susca, Larry Matthies, and Matt Malchano. Real-time pose estimation of a dynamic quadruped in gps-denied

environments for 24-hour operation. *The International Journal of Robotics Research*, 35(6):631–653, 2016.

[93] Venkatesh Madyastha, Vishal Ravindra, Srinath Mallikarjunan, and Anup Goyal. Extended kalman filter vs. error state kalman filter for aircraft attitude estimation. In *AIAA Guidance, Navigation, and Control Conference*, page 6615, 2011.

[94] Jan R Magnus and Heinz Neudecker. *Matrix differential calculus with applications in statistics and econometrics.* JohnWiley&Sons, 2019.

[95] Robert Mahony, Tarek Hamel, and Jean-Michel Pflimlin. Nonlinear complementary filters on the special orthogonal group. *IEEE Transactions on automatic control*, 53(5):1203–1218, 2008.

[96] Zachary Manchester, Neel Doshi, Robert J Wood, and Scott Kuindersma. Contact-implicit trajectory optimization using variational integrators. *The International Journal of Robotics Research*, 38(12-13):1463–1476, 2019.

[97] Peter S Maybeck. *Stochastic Models, Estimation, and Control*, volume 1. Academic Press, 1982.

[98] David Q Mayne. Differential dynamic programming–a unified approach to the optimization of dynamic systems. In *Control and Dynamic Systems*, volume 10, pages 179–254. Elsevier, 1973.

[99] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[100] Richard Montgomery. Gauge theory of the falling cat. *Fields Inst. Commun.*, 1, 07 1993.

[101] John B Moore. Discrete-time fixed-lag smoothing algorithms. *Automatica*, 9(2):163–173, 1973.

[102] Anastasios I Mourikis, Stergios I Roumeliotis, et al. A multi-state constraint kalman filter for vision-aided inertial navigation. In *ICRA*, volume 2, page 6, 2007.

[103] Richard M Murray. *A Mathematical Introduction to Robotic Manipulation.* CRC Press, 2017.

[104] Kenneth R Muske, James B Rawlings, and Jay H Lee. Receding horizon recursive state estimation. In *1993 American Control Conference*, pages 900–904. IEEE, 1993.

[105] Michael Neunert, Markus Stäuble, Markus Giftthaler, Carmine D Bellicoso, Jan Carius, Christian Gehring, Marco Hutter, and Jonas Buchli. Whole-body nonlinear model predictive control through contacts for quadrupeds. *IEEE Robotics and Automation Letters*, 3(3):1458–1465, 2018.

[106] Wyatt S Newman, Craig E Birkhimer, Robert J Horning, and Ann T Wilkey. Calibration of a motoman p8 robot based on laser tracking. In *Proceedings 2000 IEEE International Conference on Robotics and Automation.*, volume 4, pages 3597–3602. IEEE, 2000.

[107] Kevin Nickels, Eric Huber, and Matthew DiCicco. Hand-eye calibratilon using active vision. In *2007 IEEE Aerospace Conference*, pages 1–9, 2007.

[108] Jorge Nocedal and Stephen Wright. *Numerical optimization.* Springer Science & Business Media, 2006.

[109] Joseph Norby and Aaron M Johnson. Fast global motion planning for dynamic legged robots. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3829–3836. IEEE.

[110] Joseph Norby, Jun Yang Li, Cameron Selby, Amir Patel, and Aaron M. Johnson. Enabling dynamic behaviors with aerodynamic drag in lightweight tails. 37(4):1144–1153. Conference Name: IEEE Transactions on Robotics.

[111] Joseph Norby, Jun Yang Li, Cameron Selby, Amir Patel, and Aaron M Johnson. Enabling dynamic behaviors with aerodynamic drag in lightweight tails. *IEEE Transactions on Robotics*, 37(4):1144–1153, 2021.

[112] OptiTrack. OptiTrack. `https://optitrack.com/`, 2021. [Online; accessed 10-Sep-2021].

[113] Jongwon Park, Jinyi Lee, Jinwoo Lee, Kyung-Soo Kim, and Soohyun Kim. Raptor: Fast bipedal running and active tail stabilization. In *2014 11th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 215–215, 2014.

[114] Amir Patel and M. Braae. Rapid turning at high-speed: Inspirations from the cheetah's tail. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5506–5511. ISSN: 2153-0866.

[115] Marko Popovic, Andreas Hofmann, and Hugh Herr. Angular Momentum Regulation during Human Walking: Biomechanics and Control. volume 3, pages 2405–2411, January 2004.

[116] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[117] Michael Posa, Scott Kuindersma, and Russ Tedrake. Optimization and stabilization of trajectories for constrained dynamical systems. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1366–1373. IEEE, 2016.

[118] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[119] Tong Qin and Shaojie Shen. Online temporal calibration for monocular visual-inertial systems. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3662–3669, 2018.

[120] Marc H. Raibert and Ernest R. Tello. Legged robots that balance. *IEEE Expert*, 1(4):89–89, 1986.

[121] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494. IEEE, 2009.

[122] Andrzej Reinke, Marco Camurri, and Claudio Semini. A factor graph approach to multi-camera extrinsic calibration on legged robots. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 391–394. IEEE, 2019.

[123] Michal Reinstein and Matej Hoffmann. Dead reckoning in a dynamic quadruped robot: Inertial navigation system aided by a legged odometer. In *International Conference on Robotics and Automation*, pages 617–624, 2011.

[124] Gerald P Roston and Eric P Krotkov. *Dead Reckoning Navigation For Walking Robots.* Department of Computer Science, Carnegie-Mellon University, 1991.

[125] Nicholas Rotella, Michael Bloesch, Ludovic Righetti, and Stefan Schaal. State estimation for a humanoid robot. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 952–958. IEEE, 2014.

[126] ZVIS Roth, B Mooring, and Bahram Ravani. An overview of robot calibration. *IEEE Journal on Robotics and Automation*, 3(5):377–385, 1987.

[127] Robert T Savely. Apollo experience report: onboard navigational and alignment software. 1972.

[128] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92, 2011.

[129] Sangok Seok, Albert Wang, Meng Yee Chuah, David Otten, Jeffrey Lang, and Sangbae Kim. Design principles for highly efficient quadrupeds and implementation on the mit cheetah robot. In *International Conference on Robotics and Automation*, pages 3307–3312, 2013.

[130] Shaojie Shen, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar. Vision-based state estimation and trajectory control towards high-speed flight with a quadrotor. In *Robotics: Science and Systems*, volume 1, page 32. Citeseer, 2013.

[131] Ken Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, 1985.

[132] Gabe Sibley, Larry Matthies, and Gaurav Sukhatme. Sliding window filter with application to planetary landing. *Journal of Field Robotics*, 27(5):587–608, 2010.

[133] Athanasios Sideris and Luis A Rodriguez. A Riccati approach for constrained linear quadratic optimal control. *International Journal of Control*, 84(2):370–380, 2011.

[134] Sarjoun Skaff, Alfred A Rizzi, Howie Choset, and Matthew Tesch. Context identification for efficient multiple-model state estimation of systems with cyclical intermittent dynamics. *IEEE Transactions on Robotics*, 27(1):14–28, 2010.

[135] Joan Sola. Quaternion kinematics for the error-state kalman filter. *arXiv preprint arXiv:1711.02508*, 2017.

[136] Marius Stan. Euler, newton, and foundations for mechanics. 2017.

[137] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.

[138] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[139] Ke Sun, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J Taylor, and Vijay Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972, 2018.

[140] Yu Sun, Wyatt L Ubellacker, Wen-Loong Ma, Xiang Zhang, Changhao Wang, Noel V Csomay-Shanklin, Masayoshi Tomizuka, Koushil Sreenath, and Aaron D Ames. Online learning of unknown dynamics for model-based controllers in legged locomotion. *IEEE Robotics and Automation Letters*, 6(4):8442–8449, 2021.

[141] Duy-Nguyen Ta, Marin Kobilarov, and Frank Dellaert. A factor graph approach to estimation and model predictive control on unmanned aerial vehicles. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 181–188. IEEE, 2014.

[142] Yuval Tassa, Tom Erez, and William Smart. Receding horizon differential dynamic programming. *Advances in neural information processing systems*, 20, 2007.

[143] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. IEEE, 2014.

[144] James Taylor. The cramer-rao estimation error lower bound computation for deterministic nonlinear systems. *IEEE Transactions on Automatic Control*, 24(2):343–344, 1979.

[145] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT press, Cambridge, Mass., 2005.

[146] Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th annual international conference on machine learning*, pages 1049–1056, 2009.

[147] Marco Tranzatto, Takahiro Miki, Mihir Dharmadhikari, Lukas Bernreiter, Mihir Kulkarni, Frank Mascarich, Olov Andersson, Shehryar Khattak, Marco Hutter, Roland Siegwart, et al. Cerberus in the darpa subterranean challenge. *Science Robotics*, 7(66):eabp9742.

[148] Unitree. A1. `https://www.unitree.com/products/a1/`, 2021. [Online; accessed 10-Sep-2021].

[149] Vladyslav Usenko, Jakob Engel, Jörg Stückler, and Daniel Cremers. Direct visual-inertial odometry with stereo cameras. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1885–1892. IEEE, 2016.

[150] Roberto G Valenti, Ivan Dryanovski, and Jizhong Xiao. Keeping a good attitude: A quaternion-based orientation filter for imus and margs. *Sensors*, 15(8):19302–19330, 2015.

[151] Giorgio Valsecchi, Ruben Grandia, and Marco Hutter. Quadrupedal locomotion on uneven terrain with sensorized feet. *IEEE Robotics and Automation Letters*, 5(2):1548–1555, 2020.

[152] Matthieu Vigne, Antonio El Khoury, Florent Di Meglio, and Nicolas Petit. State estimation for a legged robot with multiple flexibilities using imu s: A kinematic approach. *IEEE Robotics and Automation Letters*, 5(1):195–202, 2019.

[153] Matthieu Vigne, Antonio El Khoury, Marine Pétriaux, Florent Meglio, and Nicolas Petit. Movie: a velocity-aided imu attitude estimator for observing and controlling multiple deformations on legged robots. *IEEE Robotics and Automation Letters (RA-L)*, 2022.

[154] Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IEEE Transactions on control systems technology*, 18(2):267–278, 2009.

[155] Joe Watson, Hany Abdulsamad, and Jan Peters. Stochastic optimal control as approximate input inference. In *Conference on Robot Learning*, pages 697–716, 2020.

[156] Jan Wendel and Gert F Trommer. Tightly coupled gps/ins integration for missile applications. *Aerospace Science and Technology*, 8(7):627–634, 2004.

[157] Patrick M Wensing, Albert Wang, Sangok Seok, David Otten, Jeffrey Lang, and Sangbae Kim. Proprioceptive actuator design in the mit cheetah: Impact mitigation and high-bandwidth physical interaction for dynamic legged robots. *Ieee transactions on robotics*, 33(3):509–522, 2017.

[158] David Wisth, Marco Camurri, and Maurice Fallon. Robust legged robot state estimation using factor graph optimization. *IEEE Robotics and Automation Letters*, 4(4):4507–4514, 2019.

[159] David Wisth, Marco Camurri, and Maurice Fallon. Preintegrated velocity bias estimation to overcome contact nonlinearities in legged robot odometry. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 392–398. IEEE, 2020.

[160] David Wisth, Marco Camurri, and Maurice Fallon. Vilens: Visual, inertial, lidar, and leg odometry for all-terrain legged robots. *arXiv preprint arXiv:2107.07243*, 2021.

[161] David Wisth, Marco Camurri, and Maurice Fallon. Vilens: Visual, inertial, lidar, and leg odometry for all-terrain legged robots. *IEEE Transactions on Robotics*, 2022.

[162] X Xinjilefu, Siyuan Feng, and Christopher G Atkeson. A distributed mems gyro network for joint velocity estimation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1879–1884. IEEE, 2016.

[163] Shuo Yang, Howie Choset, and Zachary Manchester. Online kinematic calibration for legged robots. *IEEE Robotics and Automation Letters*, 7(3):8178–8185, 2022.

[164] Shuo Yang, Zixin Zhang, Benjamin Bokser, and Zachary Manchester. Multi-imu proprioceptive odometry for legged robots. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023.

[165] Shuo Yang, Zixin Zhang, Zhengyu Fu, and Zachary Manchester. Cerberus: Low-drift visual-inertial-leg odometry for agile locomotion. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

[166] Yanhao Yang, Joseph Norby, Justin K. Yim, and Aaron M. Johnson. Improving tail compatibility through sequential distributed model predictive control. In *RSS Workshop on Software Tools for Real-Time Optimal Control*, July 2021.

[167] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, 2014.

[168] Yanhao Zhang, Teng Zhang, and Shoudong Huang. Comparison of ekf based slam and optimization based slam algorithms. In *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 1308–1313. IEEE, 2018.