Enhancing Quadruped Locomotion Stability with Reaction Wheel Systems and Model Predictive Control

Chi-Yen Lee

CMU-RI-TR-22-57

July 28, 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Prof. Zachary Manchester, *chair*
Prof. Aaron Johnson
Shuo Yang
Brian Jackson

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract

The development of quadruped robots offers a mobility solution that allows robot agents to navigate complicated terrains, making them extremely versatile robots in a variety of environments. Today, there are a number of research challenges facing quadruped development. First, the current state-of-the-art quadruped suffers from under-actuation during the majority of its use cases such as trotting and running. Second, the non-linear contact dynamics of legged systems make them complicated control systems to model.

In the first part of this thesis, we address the issue of under-actuation by introducing a prototype reaction wheel system that gives quadruped robots enhanced attitude stabilization ability. The dynamics of reaction wheel actuation systems allow for a straightforward linearization of the robot dynamics in comparison to other inertial stabilization appendages such as tails. We model the system as a single gyrostat and simplify the dynamics to pose the problem as a linear discrete-time trajectory optimization problem that can be solved as a quadratic program. The linear MPC is implemented on hardware at 1000hz while reasoning about the speed and torque limits of the reaction wheel systems.

The second part of the thesis explores implementing a novel Contact Implicit Model Predictive Control (CI-MPC) system on hardware. We introduce a control stack that does not rely on explicit contact mode specification, and we demonstrated that the CI-MPC is capable of solving Linear Complementary Problem (LCP) contact dynamics on hardware in real-time.

The final part of the thesis proposes a method that can reliably identify inertial parameters for quadruped systems that be used for model-based control algorithms such as the two mentioned above. We introduce a two-step calibration routine to identify the planar center of mass (CoM) position and the effective centroidal dynamics parameters of any quadruped using only joint sensors and an inertial measurement unit (IMU). Our proposed calibration routine consists of two steps: A bisection search method is used to locate the position of the planar CoM, and a sinusoidal excitation method is used to extract moments of inertia about each body axis. We verify the inertial parameter identification method in simulation, and we implemented the center of mass finding algorithm in both simulation and hardware. The results of hardware CoM finding

experiments verified in a balancing controller that requires 5mm CoM position accuracy.

Acknowledgments

I would like to start by thanking my research advisor Professor Zac Manchester for giving me the opportunity to work on these projects. His guidance and insights help me grow tremendously as a researcher and an engineer, and his unwavering support and belief in me make these projects possible.

Furthermore, I would like to thank members of the Robotic Exploration Lab. The environment for research and idea discussion is invaluable to my research progress. I would like to particularly thank Shuo Yang for being my second mentor and closest collaborator, and for inviting me to work on the quadruped control platform that he developed. These projects would not have been possible without his help and support.

Finally, I would like to thank all my friends and family for supporting my dream to work in the field of robotics. Being able to come to the US and work in RI is a dream come true for me, and I cannot be where I am today without the help I received from everyone along the way.

Funding

This work was partially supported by Google.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

The development of legged robots offers a mobility solution for autonomous agents to navigate in difficult terrains that is typically impossible for traditional wheeled robots to traverse. The mobility of the legged robots will allow them to traverse both natural and artificial landscapes, making these robots extremely versatile in a variety of environments. The primary focus of this research is on quadruped robots, specifically quadruped control and improving quadruped stability during locomotion.

There are currently a number of control challenges in quadruped control research. First, the current state-of-the-art quadruped robots suffer from under-actuation during a majority of the use cases such as walking and running [20]. For the past decades, the design for high-performance quadruped systems has in large converged. The current design of a rigid torso with four 3 degrees of freedom (DOF) legs with a point end-effector causes the quadrupeds to become underactuated whenever it has only two or fewer feet in contact with the environment. Research in biomechanics has shown that animals rely on the conservation of angular momentum to perform inertial stabilization [15, 16]. Several past research has demonstrated the importance of inertial stabilization in high mobility maneuvers [4, 7]. In addition to animals, spacecraft systems can also perform attitude control using momentum control devices [8]. It is clear that we can draw inspiration from these areas to innovate on the current quadruped design.

In addition to under-actuation, the contact-rich nature of quadruped systems also makes them complicated systems to model. Quadruped robots rely on constantly

making and breaking contact with the environment in order to navigate themselves. Over the years, there has been a number of strategies for controlling system with contacts, including hybrid-zero dynamics [1, 22], neural network policies [9, 10], and model-predictive control [24]. However, there is little work on general-purpose control techniques that can reason about contact events without requiring gait-generation heuristics, a platform-specific model, or extensive parameter tuning.

1.1 Thesis Contribution

In this thesis, we aim to use model predictive control (MPC) to enhance quadruped mobility during locomotion [19]. Specifically, we approach this problem from three perspectives: using novel hardware innovation to improve on the current quadruped design, experimentation with new algorithms that improve contact reasoning, and using state estimation techniques to improve model accuracy for MPC. First, we address the problem of under-actuation by introducing a reaction wheel systems module onto quadruped. The reaction wheel system grants quadruped inertial stabilization ability without ground contacts. The dynamics of the reaction wheel systems also allow straightforward linearization, and we leveraged the linearized dynamics to formulate a linear convex MPC problem. In the second part of the thesis, we switch direction from the hardware innovation and explore applying a method that improves contact reasoning on quadruped through a Contact Implicit Model Predictive Control (CI-MPC) algorithm [5]. We experimented with a new quadruped control architecture and demonstrated its feasibility with hardware experiments. Finally, MPC implementation such as the two we mentioned above often relies on having a good model of the system. For the final project, we present a method that can reliably estimate quadrupeds' inertial properties. Specifically, we introduce a simple calibration routine that relies only on an inertial measurement unit (IMU) and joint sensors to estimate the planar center of mass location and moment of inertia for a quadruped.

1.2 Thesis Structure

The rest of the thesis will be organized by projects. Each chapter will introduce the technical details of the projects mentioned in Section 1.1 and present our findings and conclusion. Specifically, Chapter 2 will cover the work on stabilizing quadruped locomotion with reaction wheel systems. Chapter 3 will talk about the work on applying CI-MPC on quadruped. Chapter 4 will cover the work on quadruped center of mass position and moment of inertia estimation.

Chapter 2

Reaction Wheel Assisted Locomotion for Legged Robots

In this chapter, we cover the work on enhancing quadruped locmotion stability through the addition of reaction wheel systems on the current state-of-the-art quadruped design. The rest of the chapter will be organized as followed: Section 2.2 will cover the necessary background material including quaternion algebra, simplified quadruped dynamics, the gyrostat model, and the current control architecture; Section 2.3 will cover the details on the hardware design for the reaction wheel add-on module; Section 2.4 will cover our proposed model for optimal control; Section 2.5 will cover the way we pose the control problem as a discrete trajectory optimization problem; Section 2.6 covers the swing leg control; and finally Section 2.7 will cover the result of this reaction wheel system.

2.1 Motivation

The core design of quadrupedal robots has in large converged over the past decades. Most high-performance contemporary quadrupedal robots share many fundamental design similarities that include a rigid torso, four 3 degrees-of-freedom (DOF) legs, and a round end-effector (point foot) at the end of each leg. While simple and practical, this configuration is a highly underactuated control system during locomotion [11]. During the two-foot standing phase, the robots lose rotation control authority around

the line of support with only two points of contact. Large body orientation errors can only be eliminated by foot mode switching [3, 11], and a quadruped robot becomes especially vulnerable to impact and disturbance during the two feet standing phase.

On the other hand, terrestrial animals with legs use many strategies to perform inertial stabilization during maneuvers such as walking, running, and jumping. Humans heavily regulate their whole body's angular momentum during locomotion through limb movements [16]. Cheetahs have been observed using their tails during high-speed chases and turning maneuvers [15, 23]. Falling cats are able to adjust their attitude during falls due to their highly flexible spines [14]. To improve quadruped robot stabilization ability, it is clear that we need to augment the current state-of-the-art quadruped design.

We take inspiration from the aerospace industry to enhance quadruped inertial stabilization ability. Reaction wheel systems are widely used in satellites to perform pointing and attitude control [8]. Our goal is to fundamentally enhance the stability of quadrupeds through novel hardware design that includes reaction wheels to make the robot fully actuated during locomotion. Without significantly modifying the standard 12 DOF quadruped robot design, we add a proof-of-concept payload module on the back of the robot that provides additional torque control using two reaction wheels. The 5-kg module as shown in Figure 2.2 is compact, reusable, and designed with high control bandwidth. We use Model Predictive Control (MPC) [6] to control both the robot orientation and wheel speed so the robot orientation stays controllable during a two-leg stance phase. The method, which we call *reaction-wheel MPC*, is validated in both simulation and hardware. The major contributions of this work are:

- Design and construction of a two-axis reaction wheel prototype for quadrupeds.
- A convex MPC algorithm that leverages the reaction wheels to improve disturbance rejection.
- Demonstration of enhanced stability after incorporating reaction wheels into current quadruped design.

2.2 Background

2.2.1 Unit Quaternions

In this document, we follow the Hamilton convention for quaternion — a quaternion q is described by a scalar and vector part stacked on top of each other, $q = [q_s, q_v^T]^T$. Quaternion multiplication is defined as:

$$q_1 \otimes q_2 = [q_1]_L \; q_2 = [q_2]_R \; q_1. \tag{2.1}$$

where $[q_1]_L$ and $[q_1]_L$ are orhtonormal matrices defined as

$$[q]_L = \begin{bmatrix} q_s & -q_v^T \\ q_v & q_s \, I_3 + [q_v]^\times \end{bmatrix}, \tag{2.2}$$

$$[q]_R = \begin{bmatrix} q_s & -q_v^T \\ q_v & q_s \, I_3 - [q_v]^\times \end{bmatrix}. \tag{2.3}$$

$[*]^\times$ is the skew symmetric operator for a vector. To describe quaternion kinematics, it is helpful to promote a local 3-parameter rotation representation into a quaternion with zero scalar component. This can be done with the linear transform H, described as follows:

$$\begin{bmatrix} 0 \\ \omega \end{bmatrix} = H \, \omega = \begin{bmatrix} 0 \\ I_3 \end{bmatrix} \; \omega. \tag{2.4}$$

The time derivative of a unit quaternion that describes the attitude of a body is related to the body's angular velocity through the quaternion kinematic equation:

$$\dot{q} = \frac{1}{2} \; [q]_L \; H \, \omega. \tag{2.5}$$

2.2.2 Simplified Quadruped Dynamics

In many implementations of MPC for legged locomotion, the dynamics of the robot are often simplified to a single rigid body with four reaction forces from the ground [3, 6]. This model ignores the leg masses with the assumption that they are light enough to be negligible relative to the mass of the torso. Given a mass m, body frame

moment of inertia ${}^{B}I$, the governing equation of a single rigid body model quadruped is

$$\dot{x} = \begin{bmatrix} \dot{r} \\ \dot{q} \\ {}^{N}\ddot{r} \\ {}^{B}\dot{\omega} \end{bmatrix} = \begin{bmatrix} {}^{N}v \\ \frac{1}{2}[q]_{L} H {}^{B}\omega \\ \frac{1}{m} {}^{N}F - g \\ ({}^{B}I)^{-1}({}^{B}\tau_{f} - {}^{B}\omega \times {}^{B} I\omega) \end{bmatrix}, \tag{2.6}$$

where the state of the system includes the center of mass position r, quaternion representation of the body attitude q, inertial frame translational velocity ${}^{N}v$, and body frame angular velocity ${}^{B}\omega$. The input of the system contains an inertial frame force input ${}^{N}F$ and a body frame torque input due to ground reaction force ${}^{B}\tau_{f}$. For a single rigid body model quadruped, the input forces and torques can be mapped from the ground reaction f_i force from foot position p_i for each foot with index i through the following relationship:

$$u = \begin{bmatrix} {}^{N}F \\ {}^{B}\tau_{f} \end{bmatrix} = \begin{bmatrix} I_3 & \dots & I_3 \\ R^{T}[p_1]^{\times} & \dots & R^{T}[p_n]^{\times} \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}. \tag{2.7}$$

The ground reaction forces are further subjected to friction cone constraints to prevent slippage. Often the friction cone constraint is approximate as a pyramid and enables the constraints to be expressed as a set of linear inequality constraints

$$\begin{aligned} -\mu f_z \leq & f_x \leq \mu f_z \\ -\mu f_z \leq & f_y \leq \mu f_z \end{aligned} \tag{2.8}$$

2.2.3 Gyrostat dynamics

The dynamics of a rigid body with a reaction wheel can be modeled as a gyrostat. A gyrostat is a system of coupled rigid bodies whose relative motions do not change the total inertia tensor of the system, and the fundamental governing equation that describes this model can be written as,

$${}^{B}I\dot{\omega} + {}^{B}\omega \times ({}^{B}I {}^{B}\omega + \rho) = \tau_{p}, \tag{2.9}$$

where ρ is the total angular momentum stored in the reaction wheels, and τ_ρ is the torque input into the wheels.

2.2.4 MIT Quadruped Control Architecture

The base control architecture for the reaction wheel quadruped is based on the work done by Di Carlo et al [6]. We built upon an open source implementation [1] that interfaces directly with the Unitree A1 quadruped. Fig. 2.1 provides a detailed illustration of the different components of the control block. Overall, this control architecture separately solves the swing and stance foot control problem. A stance foot i is defined by a foot that's currently in contact with the ground and returns a ground reaction force f_i. A swing foot i is defined as a foot that is currently in a swing motion and tacking a new desired location in order to drive the robot forward. Each pair of diagonal feet on the robot periodically switch between stance and swing mode as defined by a phase ϕ_i that is calculated by a gait counter. For swing foot, the controller is an end-effector PID feedback tracker that is described in 2.6. The stance foot ground reaction force and reaction wheel torques are calculated by the MPC. The MPC takes in the reference state x_d and $\dot{x}_d$, the phase of each feet, ϕ_i, position p_i, and velocity $\dot{p}_i$ for each foot i and output ground reaction force f_i for the foot that are in stance phase. The main contribution of this paper focuses on the MPC formulation and the modeling of the quadruped reaction wheel systems.

2.3 Hardware Design

The two-axis reaction wheel add-on module is a prototype stabilization system designed by Benjamin Bokser [2] and built in Robotic Exploration Lab (RExLab) specifically for the Unitree A1 Quadruped. It contains two reaction wheels for the roll and pitch axis, each with a 7.6cm and 8.5cm radius. The reaction wheels are driven by two brushless motors, each with a continuous max current draw of 60A, a maximum spin speed of 3800 RPM, and maximum torque output of 5 Nm. The system is held together by a chassis made out of polycarbonate plates and 3D-printed

[1] https://github.com/ShuoYangRobotics/A1-QP-MPC-Controller
[2] https://www.benbokser.com/reaction-wheel-assisted-quadruped-locomotion.html

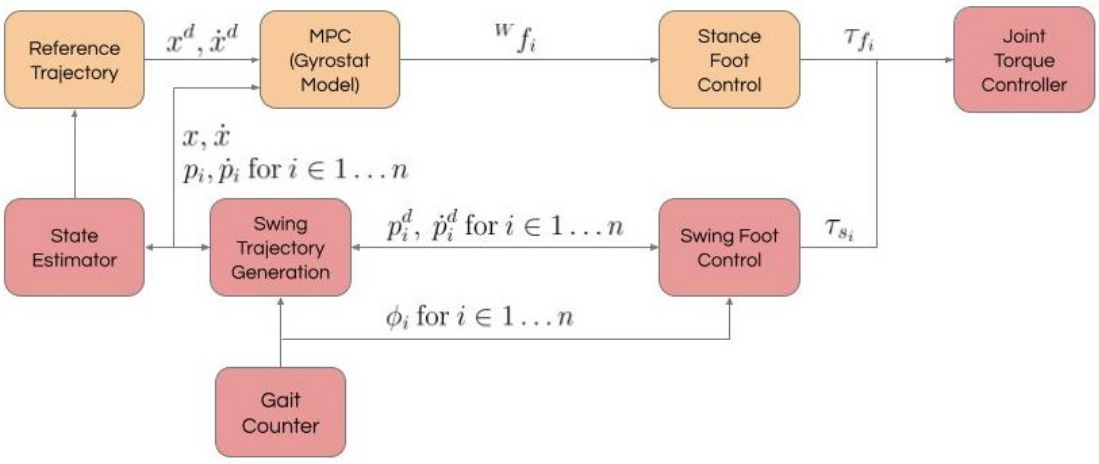

Figure 2.1: Control diagram that illustrates the quadruped control stack.

parts. The overall system has a dimension of 100 × 210 × 300 mm with a total weight of 4.6 kg that includes a 2200 mAh battery Lithium Polymer battery pack.

This module is built as a prototype for demonstrating the effect of increasing extra degrees of actuation on quadruped through reaction wheels. Variables such as efficiency, weight, and dimension are not optimized. While the system adds significant weight in addition to the weight of the robot itself, we will demonstrate that the addition of reaction wheels is significant enough to improve the overall stability of the system despite changes in the inertial parameters.

2.4 Model And Stability Analysis

We now present a simple yet effective representation of a quadruped equipped with reaction wheels by combining the single rigid body quadruped model with the gyrostat model.

2.4.1 Gyrostat Quadruped Dynamics

Our reaction wheel module adds two reaction wheels that provide body-frame torque controls about the roll and pitch axes. To incorporate the reaction wheels into the

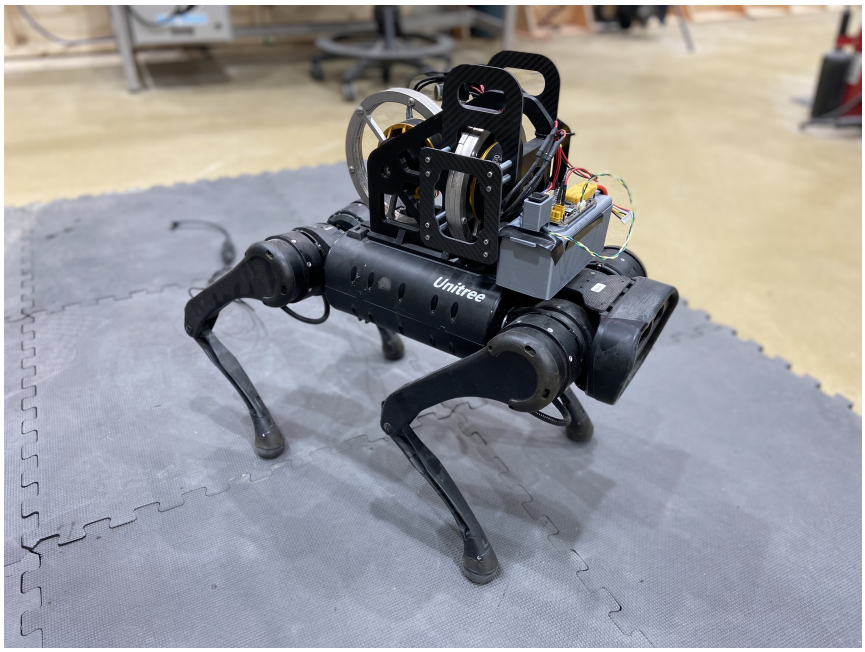

Figure 2.2: The Unitree A1 Quadruped is mounted with our custom-made reaction wheel module.

control system, we combined the single rigid body quadruped model with the gyrostat model by integrating Eq. 2.6 and Eq. 2.11:

$$\dot{x} = \begin{bmatrix} \dot{r} \\ \dot{q} \\ {}^{N}\ddot{r} \\ {}^{B}\dot{\omega} \\ \dot{\rho} \end{bmatrix} = \begin{bmatrix} {}^{N}v \\ \frac{1}{2}[q]_{L} H {}^{B}\omega \\ \frac{1}{m} {}^{N}F - g \\ ({}^{B}I)^{-1}({}^{B}\tau_f + \tau_\rho - {}^{B}\omega \times ({}^{B}I\omega + \rho)) \\ \tau_\rho \end{bmatrix}, \tag{2.10}$$

where in addition to the original centroidal model state, we introduce an addition state vector ρ and control vector τ_ρ that represents the angular momentum and torque inputs for the reaction wheels on each axis. The control input vector u can now be written as:

$$u = \begin{bmatrix} {}^{N}F \\ {}^{B}\tau_f \\ \tau_\rho \end{bmatrix} = \begin{bmatrix} I_3 & \dots & I_3 & 0_{32} \\ R^T[p_1]_\times & \dots & R^T[p_n]^\times & 0_{32} \\ 0_{23} & \dots & 0_{23} & I_2 \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_n \\ \tau_\rho \end{bmatrix}. \tag{2.11}$$

2.4.2 Controllability Under Trotting

During the trotting phase, the robot repositions itself by moving two of its legs to a new location while keeping the other two on the ground to stabilize its attitude and position. The stability of the robot under this condition can be analyzed with the controllability matrix about its linearized dynamics around an equilibrium point

$$\begin{aligned} A_k &= \left.\frac{\partial f}{\partial x}\right|_{x_0,u_0} \\ B_k &= \left.\frac{\partial f}{\partial u}\right|_{x_0,u_0} \\ \delta_{x_{k+1}} &= A_k\delta_{x_k} + B_k\delta_{u_k} \end{aligned} \quad , \tag{2.12}$$

where A_k and B_k are the linearized discrete transition and control matrix for the dynamics function f about an equilibrium point x_0 and u_0. The controllability matrix for discrete time system is given as:

$$\mathcal{C} = [B_k \;\; A_kB \;\; \dots A_k^{n-1}B] \tag{2.13}$$

where n is the dimension of the state. If $\mathcal{C}$ has full row rank, that means each of the n states is reachable by the control input u. For the single rigid body model with two ground reaction force input vectors, we found that its controllability matrix $\mathcal{C}$ has full row rank. However, the system is underactuated with the control matrix B_k losing rank for the attitude along the support vector. For the same model with a reaction wheel added on, we found that the control matrix B_k has a full rank for attitude actuation. The control matrix is still not full rank as the attitude control is now also coupled directly with the momentum of the reaction wheels. However, we assume the robot operates around a stable attitude during locomotion. This means the reaction wheel momentum can be kept low during the nominal trotting condition and activate when the robot encounters disturbances. We formulate this idea as a discrete-time trajectory optimization problem.

2.5 Convex MPC Formulation

We now formulate the reaction wheel quadruped control problem as a discrete-time trajectory optimization problem.

2.5.1 Linearized Dynamics

We extend the work done by Di Carlo et al, in which they made several key assumptions to enable the formulation of a linear convex MPC [6]. We leverage the same small-angle approximation around stable walking conditions to linearize pitch and roll dynamics. In addition, we assume that the angular velocity of the body is small enough to leave out the Coriolis term in the rotational dynamics. Using the same idea and the assumption that the reaction wheel velocities are also kept low during stable trotting, we eliminate the same Coriolis term in the gyrostat dynamics, reducing the attitude dynamics from Eq. 2.10 to

$$ {}^{B}\dot{\omega} = ({}^{B}I)^{-1}({}^{B}\tau_f + \tau_\rho) \tag{2.14} $$

For online implementation, we use Euler angles Θ instead of a quaternion as attitude parameterization, and we express angular velocity and translational velocity in world coordinates. Giving us the following linearized dynamics:

$$ \begin{aligned} \frac{d}{dt}\begin{bmatrix} r \\ \Theta \\ {}^{W}\dot{r} \\ {}^{W}\omega \\ \rho \end{bmatrix} &= \begin{bmatrix} 0_3 & 0_3 & 0_3 & 1_3 & 0_3 \\ 0_3 & 0_3 & R_z^T(\psi) & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \end{bmatrix} \begin{bmatrix} r \\ \Theta \\ {}^{W}\dot{r} \\ {}^{W}\omega \\ \rho \end{bmatrix} + \\ &\begin{bmatrix} 0_3 & \dots & & 0_3 \\ 0_3 & \dots & & 0_3 \\ \frac{1_3}{m} & \dots & \frac{1_3}{m} & 0_3 \\ {}^{W}I^{-1}[p_1]_\times & \dots & {}^{W}I^{-1}[p_n]_\times & {}^{W}I^{-1}R_z^T(\psi) \\ 0_3 & \dots & & 1_3 \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_n \\ \tau_\rho \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ g \end{bmatrix}, \end{aligned} \tag{2.15} $$

where Θ is the robot orientation parameterized by euler angles and $R_z(\psi)$ represents the rotation matrix that is linearized around the yaw angle ψ. The equation can then be written down in a convenient linear-time-varying form,

$$\dot{x}(t) = A(\psi)x(t) + B(r_1, \ldots, r_n, \psi)u(t), \tag{2.16}$$

2.5.2 Linear Discrete Trajectory Optimization

The problem is now linearized with the continuous time transition and control matrices A and B. We convert those matrices into discrete time A_d and B_d matrices. This control problem is then posed as a classic discrete-time linear trajectory optimization problem as follows:

$$\min_{x,u} \quad \sum_{0}^{k-1} \|x^d_{i+1} - x_{i+1}\|_{Q_i} + \|u_i\|_{R_i} \tag{2.17a}$$

$$\text{subject to} \quad x_{i+1} = A_{di}x_i + B_{di}u_i, i = 0 \ldots k-1 \tag{2.17b}$$

$$\underline{c}_i \leq C_i u_i \leq \bar{c}_i, i = \ldots k-1 \tag{2.17c}$$

$$Du_i = 0, i = 0 \ldots k-1, \tag{2.17d}$$

where x_i, u_i, Q_i, and R_i 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 Equation 2.17c are used to enforce linearized friction cone constraints for each ground reaction force vector. The equality constraints D_i in Equation 2.17d are used to constrain foot forces to be zero when a foot is in the swing phase. Finally, since we are working with linearized dynamics, the optimization in Equation (2.17) can be written down as a quadratic program (QP). We also regularize the speed of the reaction wheels inside the dynamics penalty term and constrain the reaction wheel torques with the affine constraints in the QP. The solution of the above MPC problem returns the ground reaction forces for each of the feet in contact with the ground. We convert the ground reaction forces into joint torques as follows,

$$\tau_i = J_i^T R^T f_i, \tag{2.18}$$

where τ_i, J_i, f_i, R are the joint torques, forward kinematic jacobian, solved ground reaction forces for leg i, and world frame to body frame rotation matrix, respectively. During actual implementation, we also include a feedback term on the angular momentum of the roll reaction wheel to compensate for the error on center of mass position

$$\phi_d = \bar{\phi}_d + K_\phi \rho_x, \tag{2.19}$$

where $\bar{\phi}_d$ is the nominal roll angle that is usually set to 0, K_ϕ is the feedback gain for the roll reaction wheel momentum, ρ_x is the roll reaction wheel momentum, and ϕ_d is the final desired roll angle for the robot. We found that this feedback term is essential to avoiding reaction wheel saturation due to wrong center of mass location.

2.6 Swing Leg Control

The foot placement on the xy plane is calculated using the following equation from Di Carlo et al's formulation [6], which took inspiration from the work done by Raibert [18].

$$p_i^d = p_i^{hip} + {}^W v \frac{\Delta t}{2} \tag{2.20}$$

where Δt is the time the foot will spend on the ground, p_i^{hip} is the hip location projected on the ground, and ${}^W v$ is the velocity of the CoM in the world frame. The z direction reference location is calculated via interpolation Bezier curve and a predetermined reference gait height. With the reference trajectory, the torque for each joint to perform tracking is computed via

$$\tau_i = J_i^T (K_p ({}^B p_i^d - {}^B p_i) + K_d ({}^B v_i^d - {}^B v_i)) \tag{2.21}$$

where J_i is the foot Jacobian, K_p and K_d are diagonal positive proportional and derivative gain matrices, ${}^B p_i$ and ${}^B v_i$ are the position and velocity of the i-th foot in body frame, and ${}^B p_i^d$,${}^B v_i^d$ are the reference position and velocity of the corresponding swing leg trajectories.

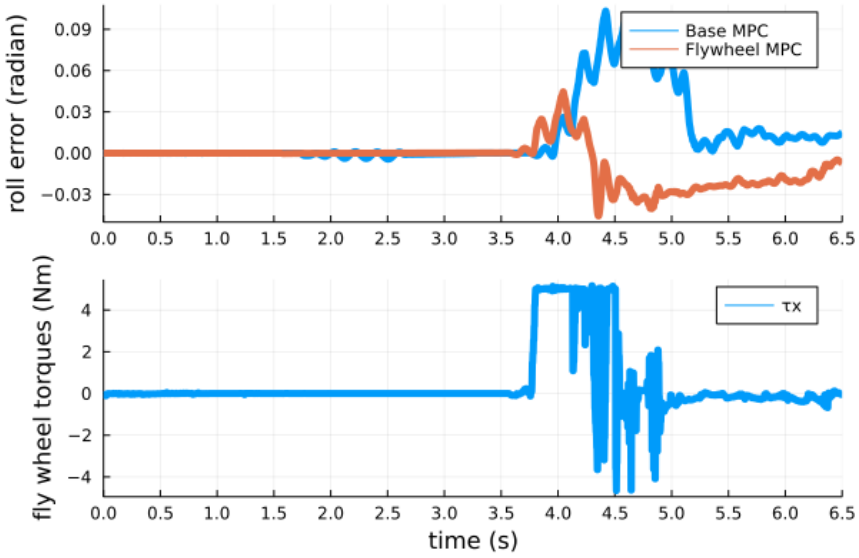

Figure 2.3: Robot roll error responses to a 350N impulse on the body frame y-axis at $t = 3.6$ seconds. The top graph illustrated the roll error trajectory with respect to time for the base MPC controller and the reaction wheel-assisted controller. The bottom graph plots the torque exerted by the x-axis reaction wheel during the experiment.

2.7 Simulation Results

We tested the reaction wheel MPC in a controlled Gazebo environment on a modified Unitree A1 model mounted with our reaction wheel module. We performed two kinds of experiments on the robot: a disturbance test on the robot body during the trotting phase of locomotion, and an aerial reorientation test where we drop the robot from a specified height at a known attitude offset.

2.7.1 Locomotion Disturbance

In the disturbance rejection tests, we supplied a 300N, 350N, and 400N impulse on the y-axis of the robot body. Figure 2.6 shows the roll error response of the robot during one of the impact experiments in Gazebo. The experiments demonstrated an enhanced ability to recover from sudden impact. Orientation errors are reduced up to 40 percent in these tests. During the 450N impulse experiments, the reaction wheel enhanced controller consistently recovers from the impact while the base MPC fails.

2.7.2 Aerial Re-orientation

In addition to the locomotion test, we also tested the aerial reorientation capability of our reaction wheel add-on module. By locking the joints of the robot and solely relying on the torques from the reaction wheels, we dropped the robot from 0.5m and provide it with angular offsets of 0.6 radians in the pitch axis for the experiment shown in Figure 2.4a, 2.4b, and 2.4c. The reaction wheels were able to steer the robot and correct its orientation in midair before touchdown. The experiment verifies the reaction wheels are able to quickly correct large orientation errors.

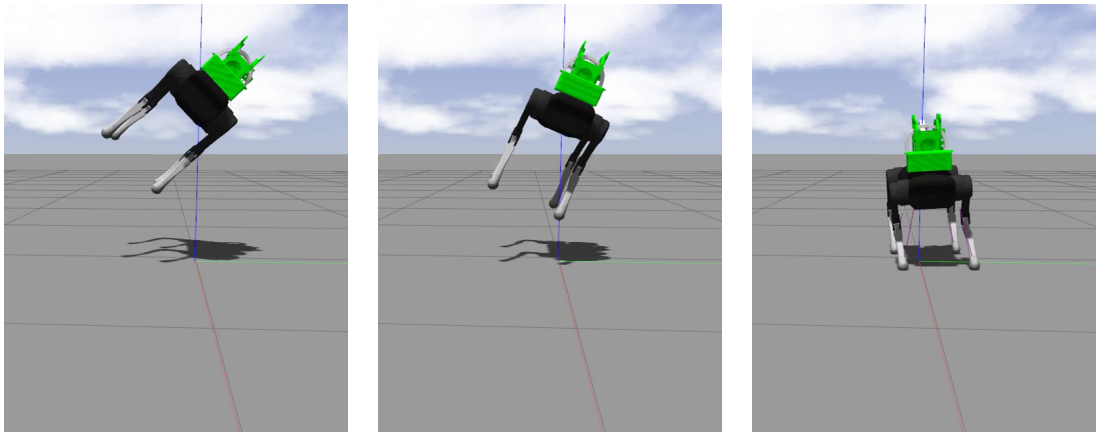

Figure 2.4: A drop test sequence where the robot reorients itself with the torques from the x axis reaction wheel.

2.7.3 Beam Walking

Finally, we tested the robot's ability to walk under a small support polygon. Similar to the two-leg stance scenario, the robot's control matrix drops rank and becomes underactuated when all of its feet are aligned in a straight line. In that instance, the robot does not have direct control over the rotation around the support vector. In this test, we put the system to test by making the robot tracks a straight line with an extremely narrow foot gait. This is the exact same stance required for stable beam walking, and the robot is able to successfully stabilize itself make make the traverse with the assistance of the reaction wheel.

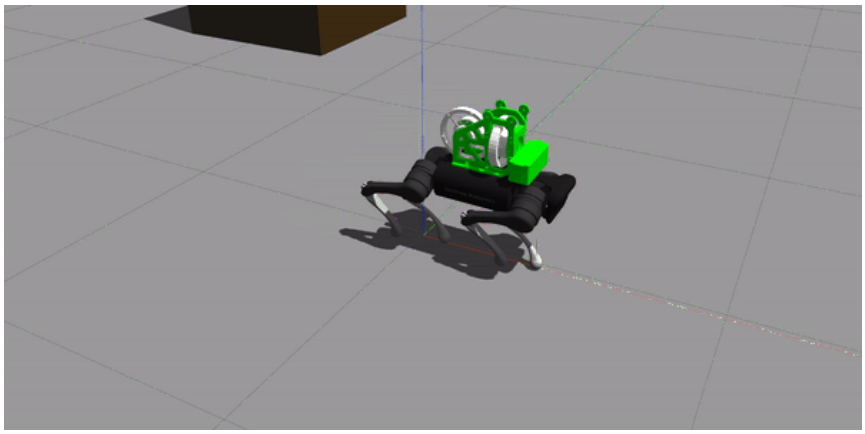

Figure 2.5: The quadruped reaction wheel system traversing a straight line along the x-axis with a narrow gait.

2.7.4 Hardware Demo

We implemented the proposed controller on a Unitree A1 robot with our custom-made payload module, as shown in Figure 2.2. The baseline controller [3] is as described in [6]. The MPC employed a look-ahead horizon of 0.5 seconds divided into 20 time steps, and the average solve time is 0.001 seconds for an AMD Ryzen Threadripper CPU. The reaction wheels provided body torque control in the roll and pitch rotation axes, each with a maximum output torque of 5Nm and a maximum spin speed of 1900 RPM. We qualitatively verified that the controller runs as expected and the robot was able to recover from arbitrary impulse disturbances. Figure 2.8 shows a sequence of the robot maintaining attitude while recovering from an impulse disturbance. In addition to the disturbance test, we tested the robot's ability to perform beam walking on hardware. The robot is able to traverse a 3 feet segments on a 6 cm beam with the assistance of the reaction wheel system prototype. Figure 2.9 shows the experiment setup for the beam walking experiment. Data from the motion capture system is used to obtain accurate location estimate of the robot's pose, and we use the leg kinematics and robot's onboard IMU to smooth out the position estimate.

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

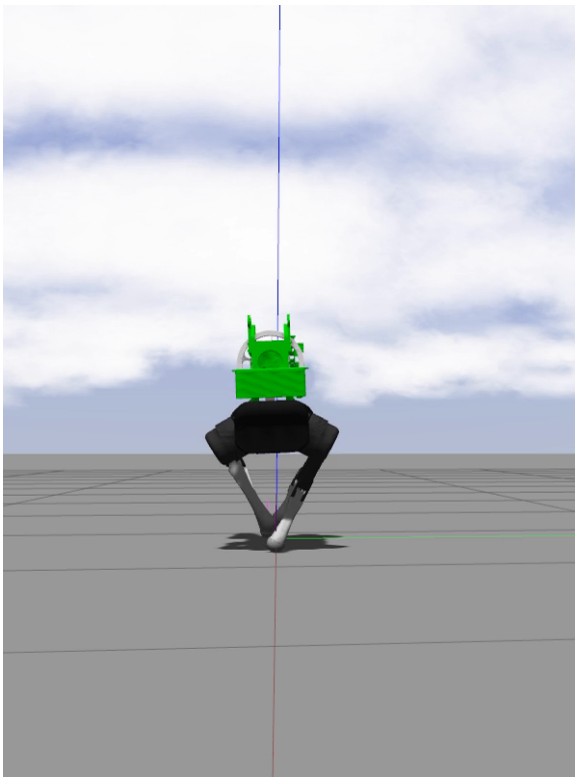

Figure 2.6: Front view of the quadruped reaction wheel system traversing a straight line along the x-axis with a narrow gait.

2.8 Conclusion

In this project, we demonstrated the feasibility of using reaction wheels to assist the attitude control of a quadruped during locomotion. We showed that by using a linearized formulation of the gyrostat dynamics, the control problem can be simplified into a clean linear convex trajectory optimization problem. Tests in simulation demonstrate the ability for the robot to reorient itself in mid-air without ground support, and it also demonstrates improved stabilization ability for quadruped under disturbances. On hardware, we demonstrate that the narrow gait walking, a motion that is extremely difficult to achieve without reaction wheel assistance, can be achieved with the reaction wheel systems and our convex MPC formulation. This work demonstrates the potential of integrating reaction wheels into modern quadruped design. For future design iterations, we recommend adding the reaction wheel systems

Figure 2.7: Hardware impulse test where we provide an impulse force on the robot during locomotion with a kick. Figure 2.7a shows the robot in stable trotting phase when the impulse is applied. Figure 2.7b shows the robot losing balance on the footholds while maintaining a stable attitude as it eventually recovers from the impulse in Figure 2.7c.

.

closer to the center of mass for the quadrupeds. This would allow the the robot to keep its center of mass lower to the ground for stability while lowering the overall moment of inertia for the whole system.

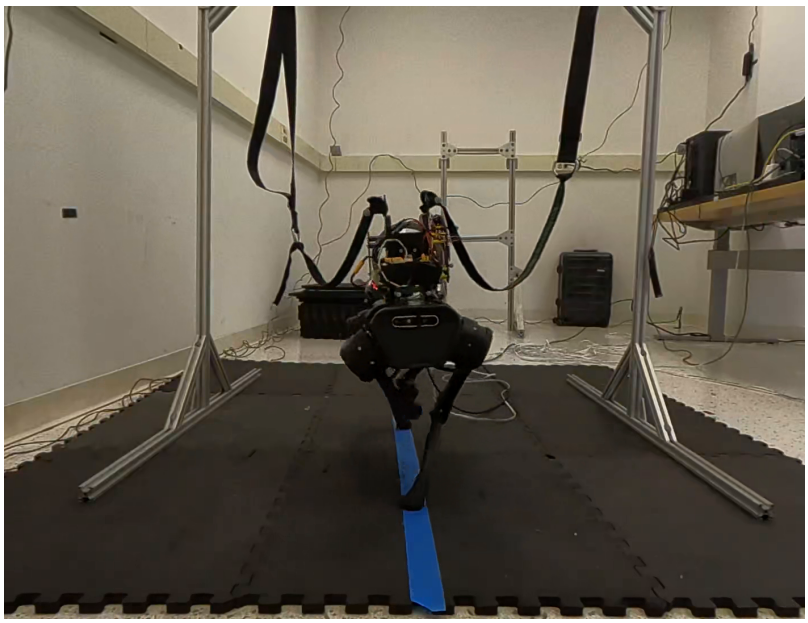

Figure 2.8: The robot is programmed to track a point that is located on the marked blue tape on the ground.

.

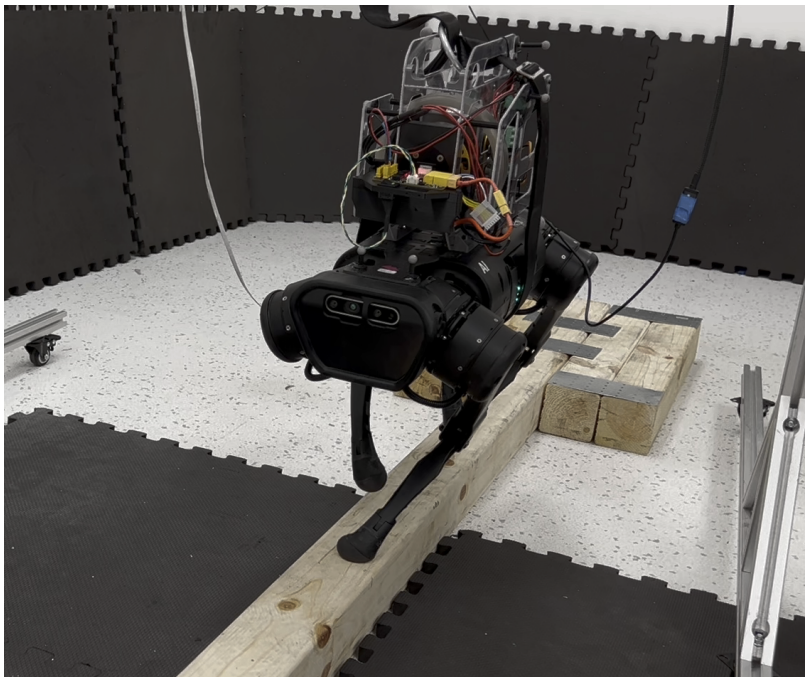

Figure 2.9: Hardware implementation of the beam walking demonstration. The robot is able to traverse a 3 feet segments of a 6 cm wide wooden beam with the assistance of the reaction wheel system.

.

Chapter 3

Applying Fast Linear Contact-Implicit Model-Predictive Control for Quadruped

In this chapter, we will cover the work done on implementing Fast Linear Contact-Implicit Model Predictive Control (CI-MPC) for quadrupeds. Section 3.2 will cover the fundamentals of CI-MPC and differentiable dynamics. Section 3.4 covers the simplified quadruped model we use to enable online tracking. Section 3.3 covers the control architecture for running CI-MPC on the quadruped. Section 3.5 covers the online foot tracking controller. Section 3.6.1 covers some of the sample trajectory and experiments we perform on actual hardware. Please note that the symbols used in this chapter are independent of the previous chapter.

3.1 Motivation

Controlling a system that needs to make and break contact with the environment is a difficult challenge in robotics. There have been many approaches to this problem, including neural network policies, hybrid zero dynamics, and hybrid model predictive control. Up until recently, there have been very few solutions that can reason about *general* contact dynamics. Le Cleac'h and Howell proposed a differentiable contact physics simulator that produces smooth gradients from contact events. They

formulate the contact problem as a complementarity problem that simultaneously satisfies contact and friction cone constraints. Then they use the path following method to gradually converge from a "soft" to "hard" contact solution by decreasing the central path parameter. Once at the solution point, the implicit function theorem is used to obtain dynamics gradients.

Using the gradient and strategic linearization of the knot points of a trajectory, Le Cleac'h and Howell utilize a combination of numerical optimization techniques and exploitation of the problem structure to come up with a general Model Predictive Control algorithm that can reason about contact changes. In this document, we apply the model predictive algorithm, called Linear Contact Implicit Model Predictive Control (LCI-MPC), for the first time on hardware on a quadruped robot. The major contributions of this work are:

- A modified single rigid body model specifically for real-time LCI-MPC implementation.
- Demonstration of complex contact-rich behaviors on hardware for a quadruped using LCI-MPC.

3.2 Background

At a high level, CI-MPC uses a contact-dynamics formulation that is efficiently evaluated and differentiated with a custom path following solver. First, the dynamics are formulated as a parameterized complementarity problem that jointly solves the impact and friction problem. Using a path-following method, the dynamics are solved by successively reducing the complementary slackness to avoid numerical issues inherent to non-smooth and discontinuous impact and friction dynamics. Then a Linear Complementarity Problem (LCP) is formed by selectively linearizing the formulation about a reference trajectory. Next, with the linearized dynamics, the CI-MPC employs a bilevel-optimization scheme that evaluates the contact dynamics and their gradient for an upper-level trajectory optimization problem by solving lower-level optimization problems. The implicit-function theorem is used to differentiate through the LCP to obtain gradients to the CI-MPC policy. Finally, a custom linear solver for the LCP that leverages offline pre-computation and partial factorization enables

real-time implementation of the CI-MPC algorithm. For brevity, this document covers background on the differentiable dynamics while leaving the details of the linearized contact-implicit dynamics solver for further reading in [5].

3.2.1 LCP Contact Dynamics

CI-MPC uses a velocity-based time-stepping scheme that optimizes a Linear Complementarity Problem (LCP) in order to find the system's next configuration q_{t+1}, and velocity v_{t+1}. The LCP includes a subproblem for impact and friction. The impact subproblem has the following structure:

$$M(q_t + hv_t)(v_{t+1} - v_t) = \tag{3.1}$$
$$\quad J(q_t + hv_t)^T \lambda_t + B(q_t + hv_t)u_t - hC(q_t, v_t),$$
$$q_{t+1} = q_t + hv_{t+1}, \tag{3.2}$$
$$\gamma_t^T \phi(q_{t+1}) = 0, \tag{3.3}$$
$$\gamma_t,\ \phi(q_{t+1}) \geq 0, \tag{3.4}$$

where M is the mass matrix, C is the dynamic bias that includes Coriolis and gravitational terms, J is the contact Jacobian that maps contact forces from local surface to the generalized coordinates, B is the input jacobian that maps control inputs into generalized coordinates, h is the discretization time step, $\lambda_t = (\gamma_t^{(1)}, \beta_t^{(1)}, \ldots, \gamma_t^{(c)}, \beta_t^{(c)})$ represents the contact forces in the local frame, with normal force $\gamma_t^{(i)} \in \mathbf{R}$ and friction forces $\beta_t^{(i)} \in \mathbf{R}^{2(d-1)}$; and signed-distance function, $\phi : \mathbf{R}^n \to \mathbf{R}^c$ that returns distance between contact points on the mechanism with the environment; $\circ$ is an element-wise (Hadamard) vector product. The manipulator equation 3.1 uses a semi-implicit Euler scheme for discretization, and the complementarity constraints ensure that contacts are only applied when the contact points are in contact with the ground.

The Coulomb friction is modeled for each contact point using the maximum

dissipation principle along with a linearized friction cone

$$\eta - \begin{bmatrix} v \\ -v \end{bmatrix} - \psi \mathbf{1} = 0 \tag{3.5}$$

$$\psi \cdot (\mu\gamma - \mathbf{1}^T\beta) = 0 \tag{3.6}$$

$$\beta^T\eta = 0 \tag{3.7}$$

$$\beta, \eta \geq 0, \tag{3.8}$$

where $v \in \mathbf{R}^{d-1}$ is the tangential velocity at a contact point, $\mu \in \mathbf{R}_+$ is the coefficient of friction, $\psi \in \mathbf{R}$ is the dual variable (Lagrange multiplier) associated with a linearized friction-cone constraint, and $\eta \in \mathbf{R}^{2(d-1)}$ is the dual variable associated with the nonnegative friction-force constraint. To evaluate the contact dynamics, CI-MPC solves the coupled impact and friction problems together as a joint feasibility problem that simultaneously satisfies (3.2-3.4) and (3.5-3.8),

$$\text{find} \quad q_{t+1}, \lambda_t, \psi_t, \eta_t^{(1)}, \ldots, \eta_t^{(c)}, s_\phi, s_\psi \tag{3.9}$$

$$\begin{aligned} \text{s.t.} \quad & \Big(M(q_{t-1})(q_t - q_{t-1}) \\ & \quad - M(q_t)(q_{t+1} - q_t)\Big)/h \\ & \quad - hC(q_t, (q_{t+1} - q_t)/h) \\ & \quad + J(q_{t+1})^T\lambda_t + B(q_{t+1})u_t = 0, \end{aligned} \tag{3.10}$$

$$s_\phi - \phi(q_{t+1}) = 0, \tag{3.11}$$

$$s_\psi^{(i)} - (\mu^{(i)}\gamma_t^{(i)} - \mathbf{1}^T\beta_t^{(i)}) = 0, \forall i, \tag{3.12}$$

$$\begin{aligned} & \eta_t^{(i)} - P^{(i)}(q_{t+1})(q_{t+1} - q_t)/h \\ & \quad - \psi_t^{(i)}\mathbf{1} = 0, \forall i, \end{aligned} \tag{3.13}$$

$$\gamma_t \circ s_\phi = \rho\mathbf{1}, \tag{3.14}$$

$$\psi_t \circ s_\psi = \rho\mathbf{1}, \tag{3.15}$$

$$\beta_t^{(i)} \circ \eta_t^{(i)} = \rho\mathbf{1}, \forall i, \tag{3.16}$$

$$\gamma_t, s_\phi, \psi_t, s_\psi \geq 0, \tag{3.17}$$

$$\beta_t^{(i)}, \eta_t^{(i)} \geq 0, \forall i. \tag{3.18}$$

Additional slack variables s_ϕ, $s_\psi \in \mathbf{R}^c$ are used as a standard technique with path-following methods to simplify the log-barrier term.

3.2.2 Path Following Method

Path-following methods can efficiently and reliably solve optimization problems with inequality constraints [12]. A problem,

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x;\theta) \\ \text{subject to} \quad & g(x;\theta) = 0, \\ & x \geq 0, \end{aligned} \tag{3.19}$$

with decision variables $x \in \mathbf{R}^n$, problem data $\theta \in \mathbf{R}^p$, objective $f : \mathbf{R}^n \times \mathbf{R}^p \to \mathbf{R}$, and equality constraints $g : \mathbf{R}^n \times \mathbf{R}^p \to \mathbf{R}^m$, can be rewritten with a logarithmic barrier in order to handle the inequality constraints as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x;\theta) - \rho \sum_{i=1}^{n} \log(x^{(i)}) \\ \text{subject to} \quad & g(x;\theta) = 0, \end{aligned} \tag{3.20}$$

and then solved by finding solutions to a sequence of barrier subproblems as the central-path parameter $\rho \to 0$. The optimality conditions for (3.20) are:

$$\nabla_x f(x;\theta) + \nabla_x g(x;\theta)^T y - z = 0, \tag{3.21}$$

$$g(x;\theta) = 0, \tag{3.22}$$

$$x \circ z = \rho \mathbf{1}, \tag{3.23}$$

$$x, z \geq 0, \tag{3.24}$$

where $y \in \mathbf{R}^m, z \in \mathbf{R}^n$ are dual variables associated with the equality and inequality constraints, respectively, $\circ$ is an element-wise (Hadamard) vector product, and $\mathbf{1}$ is a vector of ones.

The equality constraints (3.21-3.23) form a residual vector or solution map, $r : \mathbf{R}^{n+m+n} \times \mathbf{R}^p \times \mathbf{R}_+ \to \mathbf{R}^{n+m+n}$, that takes $w = (x, y, z) \in \mathbf{R}^{n+m+n}$ and the problem data as inputs. These problem data are fixed parameters during optimization, e.g., bounds on decision variables or weights in the objective. Newton or quasi-Newton

methods are used to find search directions that reduce the norm of the residual and a backtracking line search is employed to ensure that the inequality constraints (3.24) are strictly satisfied for candidate points at each iteration. Once the optimality conditions (3.21-3.24) are solved to a desired tolerance, the central-path parameter is decreased and the new subproblem is warm-started with the current solution and then solved. This procedure is repeated until the central-path parameter, also referred to as complementary slackness, is below the desired tolerance.

3.2.3 Implicit Function Theorem

An implicit function, $r : \mathbf{R}^K \times \mathbf{R}^p \to \mathbf{R}^k$, is defined by

$$r(w^*; \theta) = 0, \tag{3.25}$$

for solutions $w^* \in \mathbf{R}^k$ and problem data $\theta \in \mathbf{R}^p$. At a stationary point, $w*(\theta)$, the sensitivity of the solution with respect to the problem data can be computed with the implicit-function theorm:

$$\frac{\partial r}{\partial w}\delta w + \frac{\partial r}{\partial \theta}\delta\theta = 0, \tag{3.26}$$

and then solve for δw:

$$\frac{\partial w^*}{\partial \theta} = -(\frac{\partial r}{\partial w})^{-1}\frac{\partial r}{\partial \theta}, \tag{3.27}$$

This approach is used to differentiate the solution from the path-following method to compute the gradients of the contact dynamics for CI-MPC.

3.2.4 Linearized Contact-Implicit Dynamics

The LCI-MPC policy aims to track a reference trajectory comprising configurations $\bar{Q} = (\bar{q}_0, \ldots, \bar{q}_T)$, control inputs $\bar{U} = (\bar{u}_1, \ldots, \bar{u}_{T-1})$, and the contact forces $\bar{\Lambda} = (\bar{\lambda}_1, \ldots, \bar{\lambda}_{T-1})$.

Similar to linear MPC, we reduce the online computational burden by utilizing simplified dynamics. Specifically, we formulate linearized time-varying contact-implicit dynamics which comprise the manipulator dynamics, signed-distance function, and maximum dissipation principle terms linearized about the reference trajectory, while

the key contact dynamics modeled with nonlinear complementarity and inequality constraints are retained. At each time step along the trajectory, the dynamics are solved as the following feasibility problem

$$\begin{aligned} \text{find} \quad & w \\ \text{subject to} \quad & C(w - \bar{w}) + D(\theta - \bar{\theta}) = 0 \\ & \gamma \circ s_\phi = \rho \mathbf{1}, \\ & \psi \circ s_\psi = \rho \mathbf{1} \\ & \beta^{(i)} \circ \eta^{(i)} = \rho \mathbf{1}, \forall i \\ & \gamma, s_\phi, \psi, s_\psi \geq 0 \\ & \beta^{(i)}, \eta^{(i)} \geq 0, \forall i. \end{aligned} \tag{3.28}$$

Here, $\bar{w}$ and $\bar{\theta}$ are reference decision variables and problem data, respectively, for the path-following method. C and D are matrices that define an underdetermined linear system of equations and are pre-computed offline. The linear contact-implicit dynamics,

$$q_{t+1} = s_t(q_{t-1}, q_t, u_t), \tag{3.29}$$

$s_t : \mathbf{R}^n \times \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}^n$, solve (3.28) and return the configuration at the next time step. The contact forces at the current time step can also be returned.

3.2.5 Trajectory Optimization and Linear Dynamics Solver

CI-MPC aims to solve the following trajectory optimization problem,

$$\begin{aligned} \underset{x_{1:T}, u_{1:T-1}}{\text{minimize}} \quad & g_T(x_T) + \sum_{t=1}^{T-1} g_t(x_t, u_t) \\ \text{subject to} \quad & x_{t+1} = f_t(x_t, u_t), \quad t = 1, \ldots, T-1, \\ & (x_1 \text{ given}). \end{aligned} \tag{3.30}$$

As mentioned in the previous section, the dynamics function $f_t(x_t, u_t)$ is replaced with simplified linearized dynamics. The dynamics are comprised of linearized dynamics at the trajectory knot point while keeping the non-linear complementary constraints. The dynamics integration is hence formulated as an LCP while the top-level cost function optimization is solved with an outer Gauss-Newton optimization method,

using the gradient obtained from the implicit function theorem during the LCP solver iteration. When this problem is solved online, CI-MPC utilizes various techniques to speed up the solve time through exploitations of the problem sparsity. The details of the implementation are in the publication written by Le Clea'ch and Howell [5].

3.3 Quadruped Control Architecture for CI-MPC

Due to the differentiable contact dynamics, CI-MPC comes with the ability to adapt contact sequences online. It is even capable of reason about and generating new contact sequences for systems under disturbances. For this reason, the quadruped control architecture for CI-MPC is considerably simpler in comparison to the control systems illustrated in Fig. 2.1. Recall that in the classic MIT hybrid MPC formulation, the dynamics are broken down into pre-specified sequences of contacts, and the contact points are generated through Raibert heuristics and tracked with an end-effector PID feedback controller. As shown in Fig. 3.2, the CI-MPC control stack get rids of the entire pipeline related to generating contact sequences and replaced it with a single controller capable of calculating contact forces and feet placement in real-time. Furthermore, the control diagram referenced in Fig. 2.1 is only built around trotting locomotion. Whereas the new CI-MPC control architecture is set up to track any kind of dynamically feasible reference template trajectory.

3.4 Simplified Quadruped Model

We will now introduce the simplified quadruped model for the online implementation of CI-MPC. Most successful implementation of online MPC relies on a simplified version of the system model. These models offered reduced state and control dimensions while capturing the important dynamic characteristics of the actual system. In order to solve the trajectory optimization online for a quadruped robot, we utilize a modified version of the single rigid body dynamics model from Eq. 2.6. In order to model the feet' interaction with the ground and allow CI-MPC to optimize for swing leg trajectories, we introduced four *virtual point* masses m_f that represents the feet of the robot. This modified the single rigid body dynamics equation Eq. 2.6 to

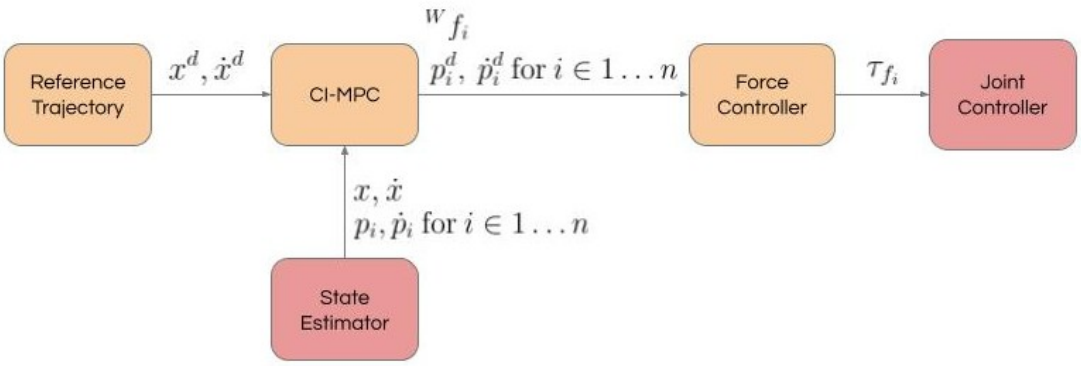

Figure 3.1: Control diagram that illustrates the quadruped control stack.

$$\dot{x} = \begin{bmatrix} \dot{r} \\ \dot{q} \\ {}^N\ddot{r} \\ {}^B\dot{\omega} \\ \dot{p}_1 \\ \ddot{p}_1 \\ \vdots \\ \dot{p}_n \\ \ddot{p}_n \end{bmatrix} = \begin{bmatrix} {}^N v \\ \frac{1}{2}[q]_L H {}^B\omega \\ \frac{1}{m} {}^N J_l^T \lambda - g \\ ({}^B I)^{-1}(J_a^T \lambda - {}^B\omega \times {}^B I\omega) \\ {}^N v_i \\ \frac{1}{m_f} F_i \\ \vdots \\ {}^N v_n \\ \frac{1}{m_f} F_n \end{bmatrix}, \tag{3.31}$$

where F_i is the force applied at each of the feet, and J_l^T and J_a^T are the attitude and translation component of the contact Jacobian matrices that map the ground reaction forces λ to the body. Finally, the sign distance functions are defined by the distance of each of the virtual point masses with the contact surfaces.

3.5 Force Tracking Controller

The output from the CI-MPC is a vector of forces applied to each of the virtual point masses. In the model, these virtual forces move the point masses around in

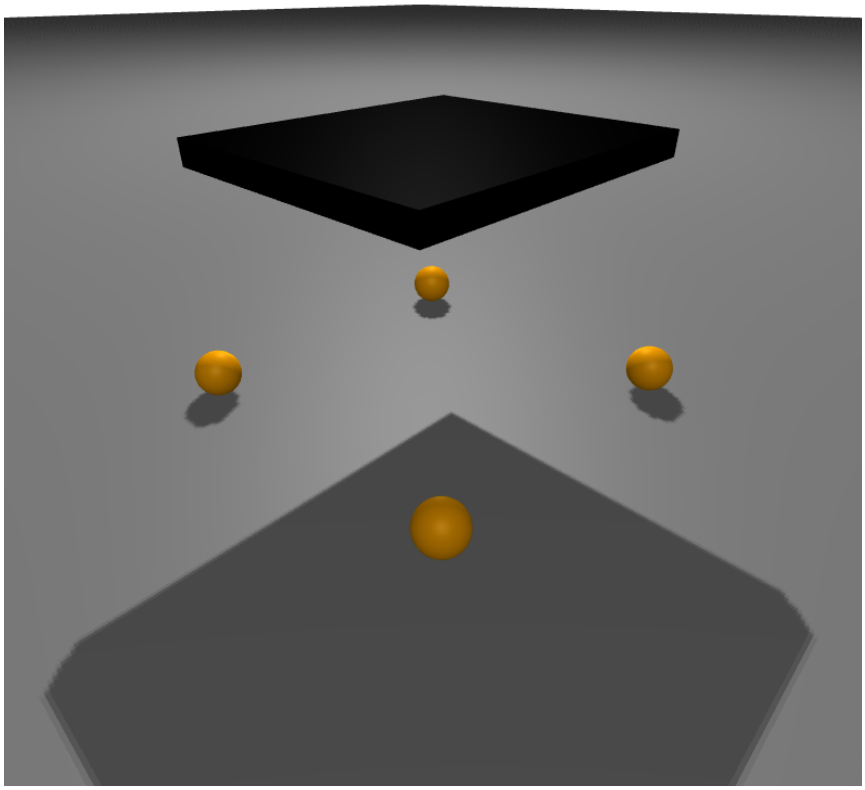

Figure 3.2: Visualization of the simplified quadruped model that is used online in CI-MPC. The four virtual point masses are represented by four yellow spheres, and the torso of the robot body is represented by a black rectangle.

the environment. When a contact event occurs between any of the spheres and the environment, a wrench is applied at the center of mass of the model from the normal and frictional forces. On hardware implementation, however, it does not make sense to directly track the force outputs with the joint torque controller when the point mass is not in contact with the environment. This is because the virtual point mass has no physical meaning for the actual robot. We could potentially tune the mass of the feet such that it captures the dynamics of the robot legs on hardware, but it would require a long tuning process. In practice, we utilize a combination of force and position tracking for a full-body quadruped. We obtain the desired feet position and velocity p^d and $\dot{p}^d$ from the first two steps of the newton solved trajectory from

the CI-MPC in Eq. 3.30. Then along with the solution F_i from Eq. 3.31, the torque to force controller for foot i becomes

$$\tau_i = J_i^T(K_p(p_i^d - p_i) + K_d(\dot{p}_i^d - \dot{p}_i)) + J_i^T R^T F_i \tag{3.32}$$

where J is the foot jacobian for foot i, K_p and K_d are the proportional and velocity gain for the position tracker, and F_i is the force to be exerted on the point mass. This formulation works with the assumption that the virtual point mass m_f is small, and therefore the force F_i is small when the point masses are moving freely away from the contact surface. At contact, the F_i is the equivalent of the ground reaction forces required to balance the robot. This tracking algorithm removes the necessity to distinguish between stance and swing foot logic.

3.6 Hardware Implementations and Experiments

3.6.1 Reference Trajectory Generation

The CI-MPC works by linearizing a set of knot points about a dynamically feasible trajectory. In order to generate a dynamically feasible trajectory, we used the formulation developed by Manchester et al to solve the trajectory as a nonlinear trajectory optimization problem with dynamics constraints similar to Eq. 3.9 [13]. A non-feasible trajectory is first generated by hand. The solver is initialized with the reference trajectory. The initial normal forces, friction forces, and complementary slackness variables are initialized to zero while the cost function heavily penalizes any deviation from the reference trajectory. Particularly important knot point parameters, for example, the foot clearance height of the trotting trajectory, are set as explicit constraints. The final trajectory is solved offline with IPOPT and the norm of the complementary slackness is checked to ensure hard contact constraints are met. The solution is then given to CI-MPC for linearization.

3.6.2 Julia Interface for Embedded System

One particular engineering challenge for hardware implementation comes from the integration between Julia and the embedded C++ control system. The dynamics

equation generation [1] and CI-MPC solver [2] are all implemented and heavily optimized in Julia. A total rewrite of the core LCP solver in C++ would add weeks into development time. We opt for integrating Julia directly with the existing control stack using the official Julia guide. Due to the fact that Julia uses a Just-In-Time (JIT) compilation scheme and the large code base of the LCP solver, each initialization takes minutes to compile at every restart of the program. We used PackageCompilter.jl [3] to precompile the entire sysimage that includes LCP solver. At runtime, the C++ program calls an instance of the Julia read–eval–print-loop (REPL) that is running a precompiled sysimage. This allows us to implement rapid change and iteration to the CI-MPC algorithm. We wrote a C++/Julia package [4] that handles Julia to C++ and type conversion and interface between our embedded control stack and the core Julia CI-MPC solver.

3.6.3 Hardware Implementation Considerations

We experimented with three different policies for hardware testing, and we closed the control loop on an Unitree A1 robot. We implemented the controller on a computer running AMD Ryzen Threadripper and achieved an average solve time of 2 ms. However, we noticed that the solution time could increase to 10 - 20 ms depending on the deviation from the reference trajectory. This severely limits the available look-ahead horizon during implementation, and the system becomes unstable whenever the robot deviates too much from the reference trajectories. We were able to keep the solve time low and track the aforementioned reference trajectories by limiting the look-ahead horizon to two steps. Due to the issue with solve time, we we have to use reference trajectories with at least 0.05s discretization time step to ensure that the solution is good enough for hardware implementation.

[1] https://github.com/thowell/RoboDojo.jl
[2] https://github.com/dojo-sim/ContactImplicitMPC.jl
[3] https://github.com/JuliaLang/PackageCompiler.jl
[4] https://github.com/RoboticExplorationLab/EmbeddedLciMpc.jl

3.6.4 Experiment 1: Trotting Policy

The first policy we tested is a standard trotting policy. The reference trajectory has a time step of 0.05 seconds and a gait cycle of 0.8 seconds. Each pair of diagonal feet (front right and rear left, front left and rear right) alternate going into the swing phase. Fig. 3.3 shows the z profile of the swing foot and Fig. 3.4 shows the solved reference normal forces for each of the feet during the 0.8-second gait. During online implementation, we use a circular buffer to update the reference points to keep the robot in a continuous trotting motion. No cost is placed on the position tracking cost, and the robot changes its desired location through changes in the reference velocity target. The robot demonstrates reliable trotting behavior using the above reference trajectories, and a video demonstration of the trotting policy is available.

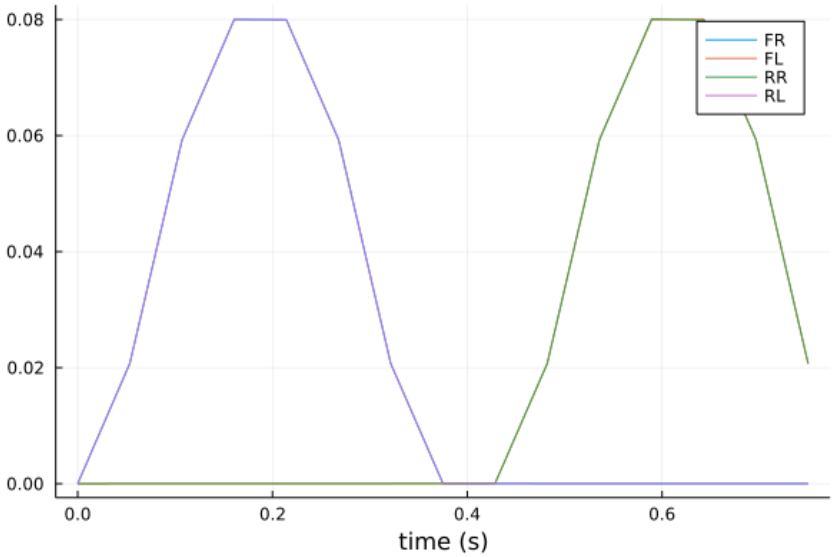

Figure 3.3: Foot height of the generated trajectory.

3.6.5 Experiment 2: Box Climbing Policy

The second policy we tested on hardware is a more complicated policy that interacts with non-flat terrain. The reference trajectory is designed to guide the robot to place its front feet on the box and lean its center of mass onto the box. The box, or more precisely a general elevated platform, is modeled as a *tanh* function for its contact

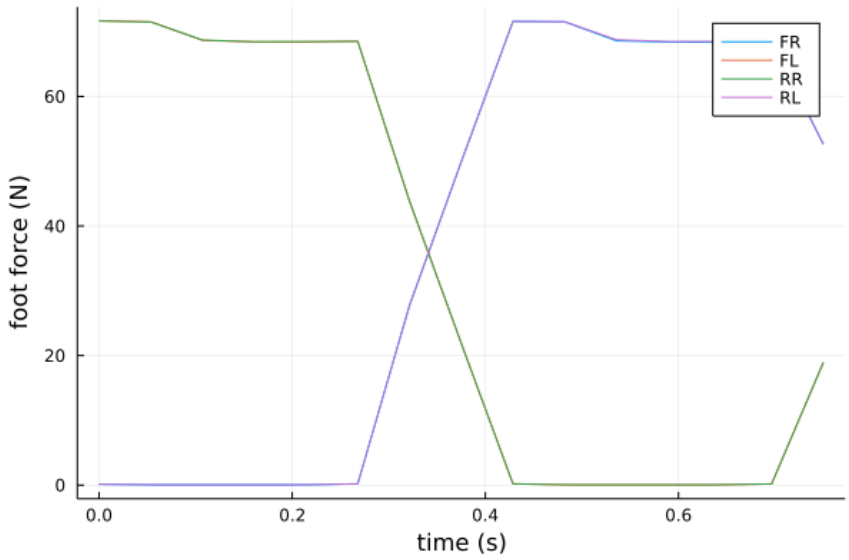

Figure 3.4: Reference normal force for the trotting policy.

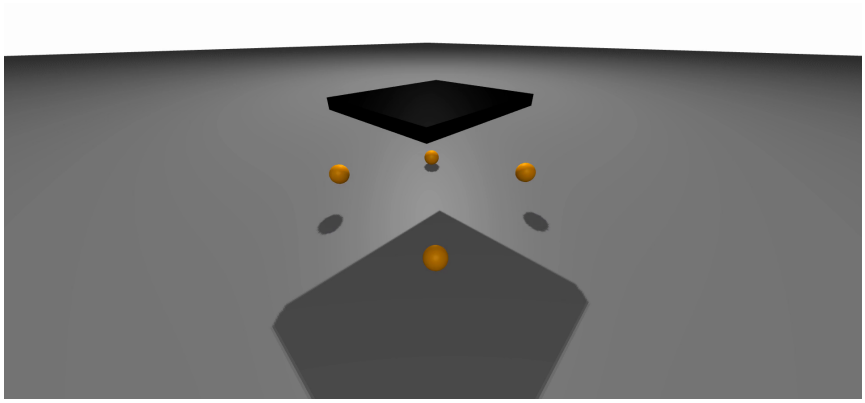

Figure 3.5: Visualization of the trotting policy on the simplified quadruped model.

surface, and the signed distance function for each virtual point is modified accordingly. Fig. 3.7 shows a visualization of the centroidal model and the reference trajectory, and Fig. 3.8 shows the hardware demonstration of the same trajectory. The next iteration of this trajectory is to guide the robot to completely maneuver itself over a larger obstacle.

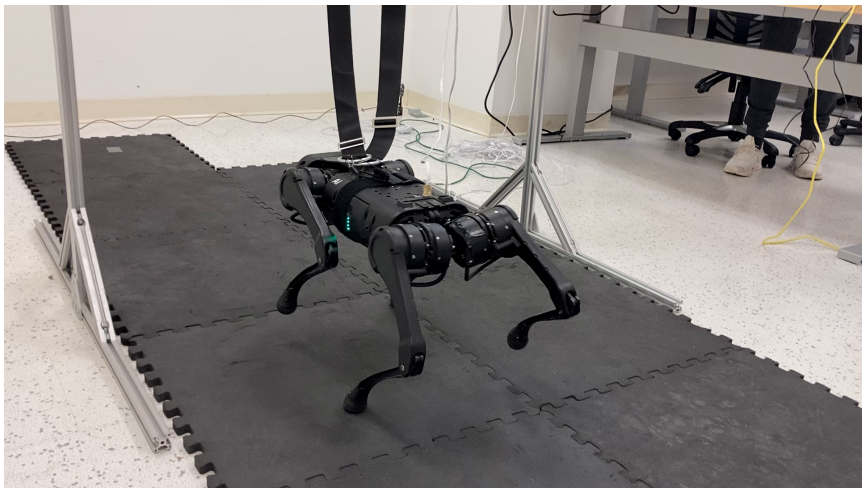

Figure 3.6: Hardware demonstration of the trotting policy.

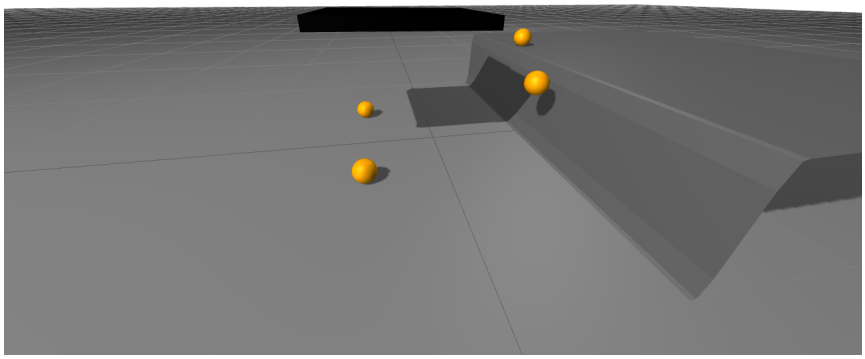

Figure 3.7: Visualization of the box climbing reference trajectory for the simplified quadruped model.

3.6.6 Experiment 3: Wall Leaning Policy

The final policy we tested on hardware is a policy that involves the robot shifting its support from a horizontal surface to a vertical surface. The end goal is to guide the robot such that it can stand up and lean against the wall. We were not able to finish implementing this policy, as our state estimator at the time of this writing do not have the capability to accurately estimate the wall position and consistently pinpoint its location. We were able to finish implementing the first half of the policy that can guide the robot's foot against the wall and shift its center of mass. Fig. 3.9 shows a

Figure 3.8: Hardware demonstration of the quadruped tracking of the box climbing trajectory.

visualization of the trajectory terminal state in visualization for the simplified model, and Fig. 3.10 shows the result achieved on hardware.

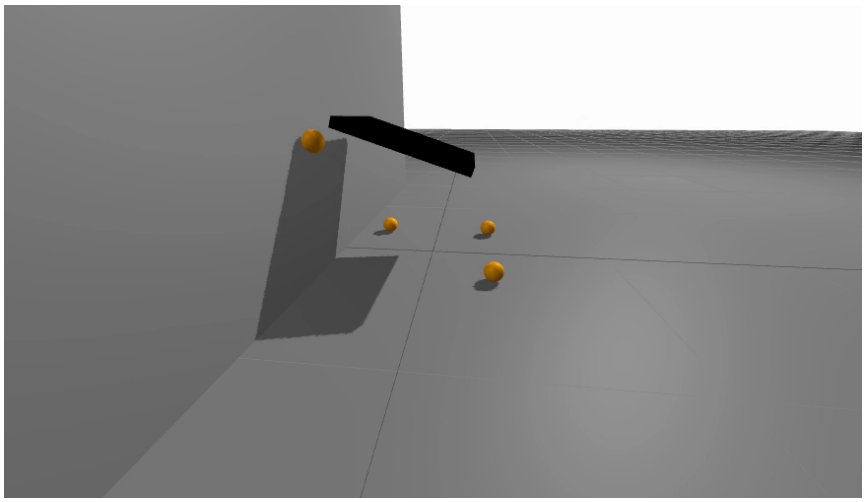

Figure 3.9: Visualization of the wall-leaning reference trajectory for the simplified quadruped model.

3.7 Conclusions

In this project, we introduced a robust pipeline for implementing CI-MPC on hardware. This is also the first instance of CI-MPC running on a hardware platform, and it

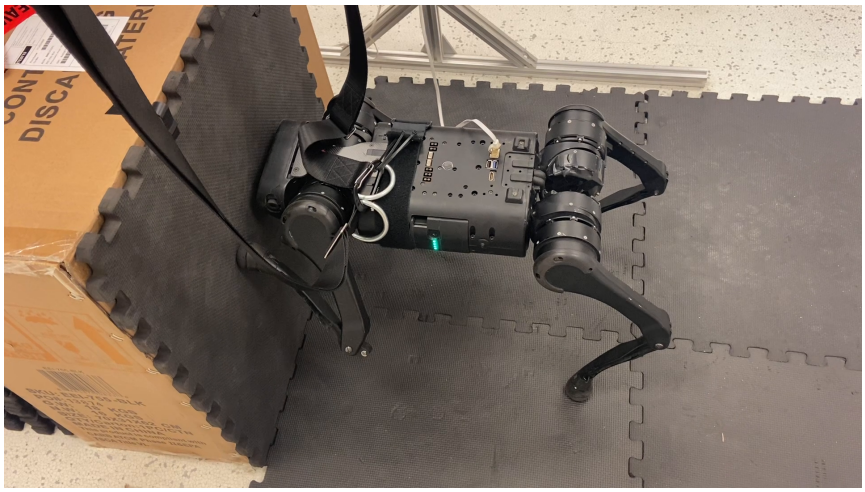

Figure 3.10: Hardware demonstration of the wall-leaning reference trajectory.

showed that LCP contact dynamics can be solved reliably in real-time on hardware for closed-loop optimal control. We demonstrated three example trajectories on hardware. From a challenging trotting policy to policy to more complicated policies that include interaction with non-flat terrains. There are several improvements that can improve the current implementation of CI-MPC. First, we could further speed up the implementation by writing the solver in C++. This would allow us to run the CI-MPC with a longer horizon and a finer discretization timestep. Second, we need to fix some solver issues plaguing the current solver. The solver struggles to converge on MPC with a longer horizon, and a small discretization time step also increases solver sensitivity to disturbances. Finally, we need to figure out a way to introduce perception and update contact surfaces in CI-MPC.

Chapter 4

A Quadruped Inertial Parameter Estimation Method with Bisection Search and Sinusoidal Excitations

4.1 Motivation

For model-based control algorithms like the two introduced in this document, the accuracy of the system's model directly impacts the performance of the controller. Past and recent works on robot system identification focused primarily on identifying the full inertial properties of each individual link [2, 21]. However, many of the best-performing state-of-the-art controllers require only a simplified centroidal model of the robot [6]. The primary contribution of our work is to introduce a drastically simplified method for extracting any quadruped's centroidal inertial parameters. We introduce a simple two-step calibration routine to identify the planar center of mass (CoM) and the effective centroidal dynamics parameters using only joint sensors and an inertial measurement unit (IMU). Our proposed calibration routine consists of two steps:

- A bisection search method to locate planar CoM.
- A sinusoidal excitation method to extract moments of inertia for each body axis.

The ideas behind these routines are simple enough to be applied to nearly any quadrupedal system with a set of joint sensors and an IMU. Our algorithms require no specific controller or complicated physical setup. We demonstrate the methods in both hardware and simulation. A video demonstrating the experiments is also available online[1].

4.2 End-effector Bisection Search

The main idea behind the bisection search method is to leverage the tilting direction of the robot when standing on two feet as an indicator of the center of mass error relative to a support line. When a quadruped lifts two of its feet off the ground during a stance, the support polygon formed by the four feet becomes a support line between the remaining diagonal feet, as shown in Figure 4.1. The robot should remain balanced for a brief moment in an unstable equilibrium point if the CoM is directly on the support vector [17]. Otherwise, the robot will tilt in the direction of the CoM offset as illustrated in Figure 4.2a and 4.2b. Using this knowledge, we shift the diagonal support line using a bisection search in the robot's body frame until it moves below the CoM location. The details of the search method are summarized in Algorithm 1, where we take in diagonal pairs of nominal foot position vectors r_1, r_2 and r_3, r_4, an upper and lower search bound x_u and x_l, and a t_{thres} that represents the time it takes for the robot to tilt into a static pose. Line 2-5 offset the support vector by x_m, the midpoint of x_l and x_u, and use inverse kinematics to calculate a desired joint configuration q_d. Line 6-8 then move the robot into the desired static stance with a joint position controller, and Line 9-12 then lift two of the diagonal leg and let the robot lean toward the direction of the CoM offsets. Depending on the direction of tilt, as determined by the pitch error, we update x_u or x_l to narrow the search direction until the two numbers come within a certain threshold (which was set to 0.005 m in Algorithm 1). We perform the same process for both pairs of diagonal feet (front right feet and rear left feet v.s. front left feet and rear right feet), and we identify two lines that we know the planar CoM must be located on. The center of mass location can be obtained by finding the intersection of the two lines.

[1] https://youtu.be/oWS1gqfT1m0

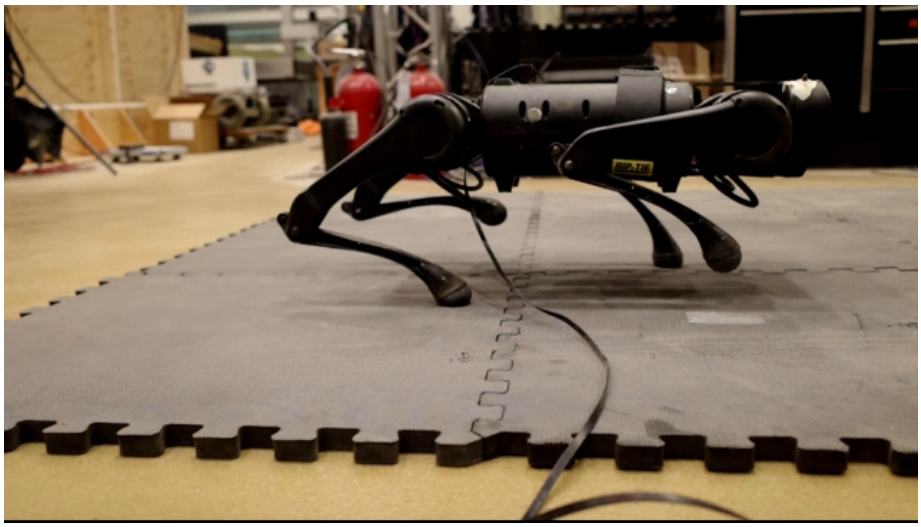

Figure 4.1: An image of an Unitree A1 quadruped balancing with its CoM directly above the support vector line formed by its front left and rear right feet.

Algorithm 1 FindSupportVector(r_1, r_2, r_3, r_4 x_u, x_l, t_{thres})

Require: $x_u > x_l$

while $x_u - x_l > 0.005$ **do**
  $x_m \leftarrow (x_u + x_l)/2$
  $\hat{r}_1.x \leftarrow r_1.x + x_m$
  $\hat{r}_2.x \leftarrow r_2.x + x_m$
  $q_d = \text{invKin}(\hat{r}_1, \hat{r}_2)$
  repeat
    jointPositionControl(q, q_d)
  until $q_d \approx q$
  $q_n = \text{liftDiagonalFeet(q)}$
  repeat
    jointPositionControl(q, q_n)
  until $t > t_{thres}$
  if $pitcherror > 0$ **then**
    $x_l = x_m$
  else
    $x_u = x_m$
  end if
end while
return $\hat{r}_1$, $\hat{r}_2$

Algorithm 2 FindCOM(r_1, r_2, r_3, r_4 x_h, x_l, t_{thres})

1: Initialize x_l, x_h
2: Initialize r_1, r_2, r_3, r_4
3: $\hat{r}_1, \hat{r}_2$ = FindSupportVector(r_1, r_2, r_3, r_4, x_h, x_l, t_{thres})
4: $\hat{r}_3, \hat{r}_4$ = FindSupportVector(r_3, r_4, r_1, r_1, x_h, x_l, t_{thres})
5: x_c, y_c = FindIntersection($[\hat{r}_1, \hat{r}_2]$, $[\hat{r}_3, \hat{r}_4]$)

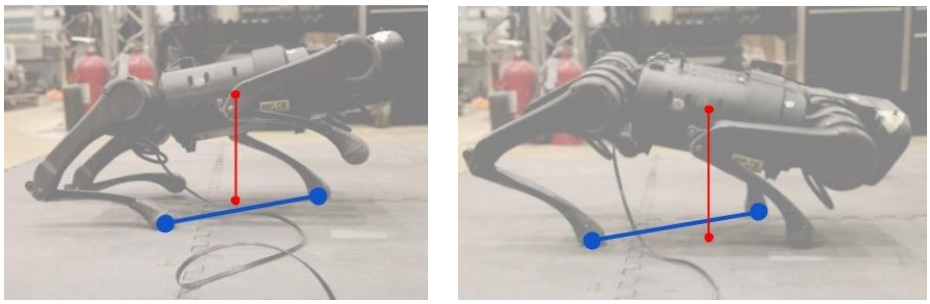

Figure 4.2: Depending on the center of mass (red) location relative to the support vector line (blue), the robot will either tilt forward like in 4.2b or backward like in 4.2a

.

4.3 Trunk Inertial Parameter Estimation With Sinusoidal Excitations

Instead of trying to estimate the inertial parameters for the full-body model, we focus on fitting the inertial parameters for a simplified centroidal model. The governing equation of a centroidal model quadruped can be written as,

$$\begin{bmatrix} \ddot{p} \\ \frac{d}{dt}(I\omega) \end{bmatrix} = \sum_{i=0}^{n} \begin{bmatrix} \frac{f_i}{m} \\ \hat{r_i} \times f_i \end{bmatrix} - \begin{bmatrix} g \\ 0 \end{bmatrix}, \tag{4.1}$$

where f_i is the ground reaction force for foot i, p is the CoM position, I is the moment of inertia, ω is the angular velocity, r_i is the foot position relative to the CoM, and g is gravity. Assuming that the angular velocity is small and the off-diagonal terms on the inertia tensor are negligible, we simplify the rotational dynamics in each axis as

$$I_j\dot{\omega}_i + C_j\omega_i = \tau_j, \tag{4.2}$$

where I_j, ω_j, C_j, and τ_j are the robot's moment, angular velocity, damping constant, and total torque for a rotation axis j. I_j and C_j can be identified by forming a regressor matrix with a dataset of ω_j, $\dot{\omega}_j$ and τ_j. To obtain the angular acceleration $\dot{\omega}_j$, we avoid doing numerical differentiation by providing a sinusoidal input to the system. We extract the dominant frequency $\mathcal{F}_j$, amplitude a_j, and phase shift ϕ_j via Fast Fourier Transform, and we take the derivative of the wave function analytically to get $\dot{\omega}_j$, giving us the expression

$$\omega_j(t) = a_j \sin(2\pi\mathcal{F}_j t + \phi_j) \tag{4.3}$$

$$\dot{\omega}_j(t) = a_j 2\pi\mathcal{F}_j \cos(2\pi\mathcal{F}_j t + \phi_j). \tag{4.4}$$

With an analytical expression of the angular velocity and angular acceleration, we form the regressor matrix by sampling ω_j and $\dot{\omega}_j$ at a number of discrete time steps: $0 \dots t_f$. With the regressor matrix, we arrive at the following linear least-squares problem

$$\begin{bmatrix} \dot{\omega}_j(0) & \omega_j(0) \\ \dot{\omega}_j(t_1) & \omega_j(t_1) \\ \vdots & \vdots \\ \dot{\omega}_j(t_n) & \omega_j(t_n) \end{bmatrix} \begin{bmatrix} I_j \\ C_j \end{bmatrix} = \begin{bmatrix} \tau_j(0) \\ \tau_j(t_1) \\ \vdots \\ \tau_j(t_f) \end{bmatrix}. \tag{4.5}$$

We apply this sinusoidal excitation process for each of the three rotation axes, and we perform separate the least squared optimization for each axis to extract the moment of inertia and damping constant.

4.4 Results

We implemented the center of mass search on both hardware and simulation. As of the time of writing, we have only performed the moment of inertia identification in simulation.

4.4.1 Simulation Results

We ran both calibration routines in Gazebo with the full Unitree A1 quadruped model. Table 4.1 shows the simulated and estimated moment of inertia and CoM. To obtain the effective moment of inertia values, we use the parallel axis theorem and calculate the total moment of inertia of the robot's trunk, hips, and thigh links. Similarly, we calculate the CoM position by calculating the weighted sum of the CoM of each link in the robot's body frame. Monte Carlo simulation of the calibration routine is able to consistently identify the CoM position within 5mm of the ground truth position. The error is also directly proportional to the joint and end-effector error. Figure 4.3 shows the norm of the CoM position error during a bisection search, with the search error converging to under 5mm in roughly six iterations.

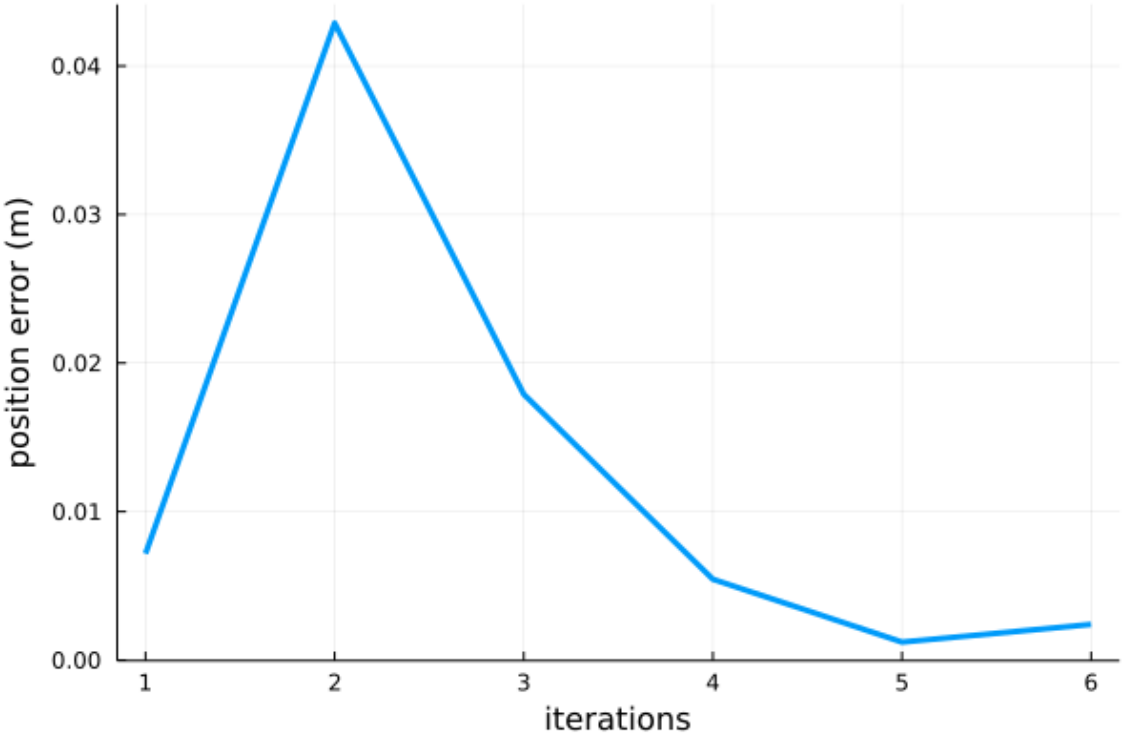

Figure 4.3: Norm of the position error on the CoM estimates over six iterations for a calibration test in simulation.

Table 4.1: Simulated Result of the Calibration Routine

	I_x	I_y	I_z	p_x	p_y
Simulated	0.116	0.349	0.399	-0.0093	0.00085
Estimated	0.0856	0.306	0.363	-0.0073	0.00089

4.4.2 Experimental Results

We implemented the CoM bisection search algorithm on hardware for an Unitree A1 quadruped. While it is hard to experimentally determine the ground truth location of the CoM on hardware, we implemented a balancing controller that balances the robot on two diagonal feet. This under-actuated balancing scenario requires an accurate center of mass position estimate. From our simulation and hardware experiments, we determine that the CoM needs to be within 5mm for the robot to balance properly with our controller. We were able to successfully balance the robot on two legs using the center of mass position estimate we obtained from our calibration routine. A video demonstration of the calibration process is available in Section I.

4.5 Conclusion

In this project, we introduce a method that can reliably identify several important inertial parameters for quadruped model-based controllers. This method is highly applicable to any MPC, including the two methods introduced in Chapter 2 and 3, that uses a simplified centroidal model for a quadruped. We demonstrated that our center of mass search algorithm works on both hardware and simulation. Our current calibration routine only identifies the CoM position on the x and y plane of the robot body frame. However, a similar technique can be used to back out the z-axis CoM position by tilting the robot at an angle and accounting for geometry constraints given a known $x - y$ CoM location. The sinusoidal excitation method, although it can identify the moment of inertia in simulation up to two decimal digit accuracy, have yet to be tested on hardware. Our future work will test this technique for the Unitree A1 robot and adapt it according to the challenges we encounter.

Bibliography

[1] Aaron D. Ames, Kevin Galloway, Koushil Sreenath, and Jessy W. Grizzle. Rapidly exponentially stabilizing control Lyapunov functions and hybrid zero dynamics. *IEEE Transactions on Automatic Control*, 59(4):876–891, 2014. 1

[2] Christopher G. Atkeson, Chae H. An, and John M. Hollerbach. Estimation of inertial parameters of manipulator loads and links. 5(3):101–119. ISSN 0278-3649, 1741-3176. 4.1

[3] 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. 2.1, 2.2.2

[4] Carlos Casarez, Ivan Penskiy, and Sarah Bergbreiter. Using an inertial tail for rapid turns on a miniature legged robot. In *2013 IEEE International Conference on Robotics and Automation*, pages 5469–5474, 2013. doi: 10.1109/ICRA.2013.6631361. 1

[5] Simon Le Cleac'h, Taylor Howell, Mac Schwager, and Zachary Manchester. Fast contact-implicit model-predictive control, 2021. URL https://arxiv.org/abs/2107.05616. 1.1, 3.2, 3.2.5

[6] 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, Madrid, October 2018. IEEE. ISBN 978-1-5386-8094-0. doi: 10.1109/IROS.2018.8594448. URL https://ieeexplore.ieee.org/document/8594448/. 2.1, 2.2.2, 2.2.4, 2.5.1, 2.6, 2.7.4, 4.1

[7] C. Fisher and A. Patel. Preparation of papers for ifac conferences symposia: Flipbot: A lizard inspired stunt robot. *IFAC Proceedings Volumes*, 47(3):4837–4842, 2014. ISSN 1474-6670. doi: https://doi.org/10.3182/20140824-6-ZA-1003.01479. URL https://www.sciencedirect.com/science/article/pii/S1474667016423633. 19th IFAC World Congress. 1

[8] Mason A. Peck Frederick A. Leve, Brian J. Hamilton. *Spacecraft Momentum Control Systems*. Springer, 2015. 1, 2.1

[9] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017. 1

[10] Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S. Sukhatme. Neuralsim: Augmenting differentiable simulators with neural networks. *arXiv preprint arXiv:2011.04217*, 2020. 1

[11] 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. 2.1

[12] Stephen J. Wright Jorge Nocedal. *Numerical Optimization*. Springer, 2006. 3.2.2

[13] 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. doi: 10.1177/0278364919849235. URL https://doi.org/10.1177/0278364919849235. 3.6.1

[14] Richard Montgomery. Gauge theory of the falling cat. *Fields Inst. Commun.*, 1, 07 1993. doi: 10.1090/fic/001/09. 2.1

[15] 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, November 2013. doi: 10.1109/IROS.2013.6697154. ISSN: 2153-0866. 1, 2.1

[16] Marko Popovic, Andreas Hofmann, and Hugh Herr. Angular Momentum Regulation during Human Walking: Biomechanics and Control. volume 3, pages 2405–2411, January 2004. doi: 10.1109/ROBOT.2004.1307421. 1, 2.1

[17] Marc H Raibert. Research on legged machines can lead to the construction of useful legged vehicles and help us to understand legged locomotion in animals. 29(6):16. 4.2

[18] Marc H. Raibert and Ernest R. Tello. Legged robots that balance. *IEEE Expert*, 1(4):89–89, 1986. doi: 10.1109/MEX.1986.4307016. 2.6

[19] Robert F. Stengel. *Optimal Control and Estimation*. 2012. 1.1

[20] Russ Tedrake. *Underactuated Robotics*. 2022. URL http://underactuated.mit.edu. 1

[21] Guido Tournois, Michele Focchi, Andrea Del Prete, Romeo Orsolino, Darwin G.

Caldwell, and Claudio Semini. Online payload identification for quadruped robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4889–4896. IEEE. ISBN 978-1-5386-2682-5. 4.1

[22] Eric R. Westervelt, Jessy W. Grizzle, and Daniel E. Koditschek. Hybrid zero dynamics of planar biped walkers. *IEEE Transactions on Automatic Control*, 48 (1):42–56, 2003. 1

[23] A. M. Wilson, J. C. Lowe, K. Roskilly, P. E. Hudson, K. A. Golabek, and J. W. McNutt. Locomotion dynamics of hunting in wild cheetahs. *Nature*, 498(7453): 185–189, June 2013. ISSN 1476-4687. doi: 10.1038/nature12295. 2.1

[24] Alexander W. Winkler, C. Dario Bellicoso, Marco Hutter, and Jonas Buchli. Gait and Trajectory Optimization for Legged Systems Through Phase-Based End-Effector Parameterization. *IEEE Robotics and Automation Letters*, 3(3): 1560–1567, July 2018. 1