ADAPTING THE PLANNING AND CONTROL OF LEGGED ROBOTS TO EXTREME ENVIRONMENTS

A Thesis

Submitted to the College of Engineering

of the University of Notre Dame

in Partial Fulfillment of the Requirements

for the Degree of

Bachelor of Science

in

Electrical Engineering

by

Nolan Fey

Michael Lemmon, Co-Advisor

Patrick Wensing, Co-Advisor

Undergraduate Program in Electrical Engineering

Notre Dame, Indiana

May 2023

ADAPTING THE PLANNING AND CONTROL OF LEGGED ROBOTS TO EXTREME ENVIRONMENTS

Abstract

by

Nolan Fey

The bio-inspired design of legged robots offers them the potential to traverse complex environments like their animal counterparts, making them ideal for deployment either alongside or in place of humans in dangerous environments—whether it be the forest floor during a wildfire, the rubble after a natural disaster, or the Martian surface during a mission for scientific samples. State-of-the-art control methods for quadrupeds achieve impressive performance and robustness over various terrains (i.e. stairs, rocky soil, and snow). As these methods approach greater sophistication, there is a growing interest toward developing planning and control algorithms that can overcome uncertainties in new environments, thereby expanding the capabilities of legged systems on extreme terrains. As a step toward this goal, this thesis aims to develop algorithms for quadrupeds that allow them to plan motions through complex environments and adapt their control input on the fly in response to changing terrain and unmodeled dynamics. The first objective is to develop a planning method for carefully choosing contacts over cluttered and discontinuous terrain. Our approach considers a mixed-integer programming formulation based on a single-rigid-body model with an impulsive stance phase, allowing us to include a treatment of 3D orientation dynamics and actuator limits while keeping the computational demands manageable for real-time deployment. We found that our impulsive stance formulation typically finds a solution to a 4-hop planning problem within 1 second, even in larger environments where an equivalent full stance formulation fails to find a solution within 10 seconds. Our second and third objectives coexist in an overarching learning framework that allows the robot to stabilize itself under unmodeled dynamics in dynamic environments. The second objective is to develop techniques for estimating the steady-state response of the robot in response to an environmental disturbance. Specifically, we introduce a new technique that relies on a transformer neural network trained offline to predict the steady-state output of the system. Before deployment, we remove the last layer of the neural network and replace them with an adaptable layer of linear coefficients, which can be updated online to finetune the model to the environment at hand. The final objective is to implement a technique to learn the robot's unmodeled dynamics and apply a control method to force the robot back to its steady-state behavior. Leveraging the observation that the dynamics are nearly linear about the steady-state behavior, we use dynamic mode decomposition with control to learn a linear approximation of the error system dynamics and then implement a passivity-based control law to regulate the system. We demonstrate the effectiveness of our approach in a MuJoCo simulation of a Ghost Robotics Vision 60 quadruped, applying our framework to adapt it to unmodeled dynamics while walking on a treadmill with a time-varying speed setting.

CONTENTS

FIGURES

TABLES

CHAPTER 1

INTRODUCTION

1.1 Motivation

The bio-inspired design of legged robots makes them uniquely capable for traversing complex environments because they theoretically can maneuver over any terrain that a human can, either alongside them or in their place. Their locomotion capabilities make them a promising platform for future robotic applications that require significant mobility and control—i.e. a construction robot, a cave exploration robot for distant planets (Figure 1.1), or a home robot that can empty the dishwasher and fetch a glass of water. Recent advancements in quadrupedal and bipedal robotics have made impressive progress toward making these possibilities a reality. State-of-the-art control methods for quadrupedal robots achieve considerable robustness over various terrains, including stairs, rocky soil, grass, and snow. The MIT Mini Cheetah robot can maintain its balance even when running over icy terrain (Figure 1.3). Modern control methods also allow legged robots to execute precise dynamic movements. For example, Boston Dynamics often releases online videos of its Atlas robot dancing and performing acrobatic parkour motions (Figure 1.2). The success of these approaches enables the deployment of quadruped robots for industrial applications. Within the Oil & Gas and Chemical industries, engineers deploy quadruped robots to remotely inspect equipment at high risk facilities (Figure 1.3). These applications are impressive and free people from monotonous and unsafe tasks. Still, significant improvement is necessary before legged robots can fully realize their potential.

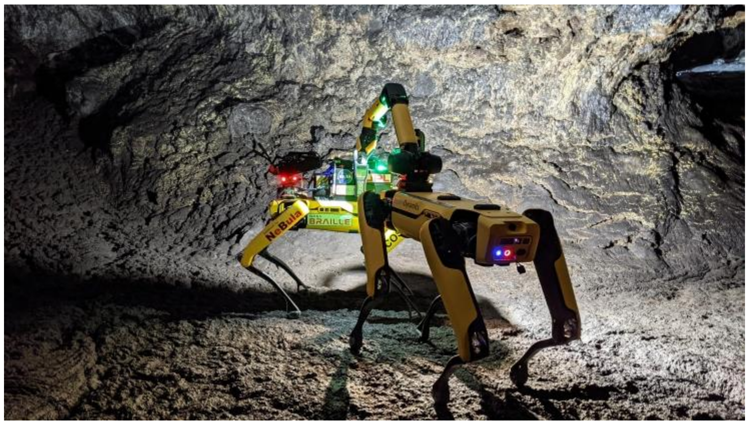

Figure 1.1. Two Nebula-SPOT quadruped robots developed by Boston Dynamics and deployed by NASA to explore a cave as part of the Defense Advanced Research Projects Agency Subterranean Challenge. Engineers at the Jet Propulsion Laboratory equipped the quadruped with sensors and autonomy software to enable it to construct maps of underground cave systems and urban environments. In the future, these robots may be deployed to map and explore cave systems on other worlds, potentially to collect scientific samples in search for signs of alien life.

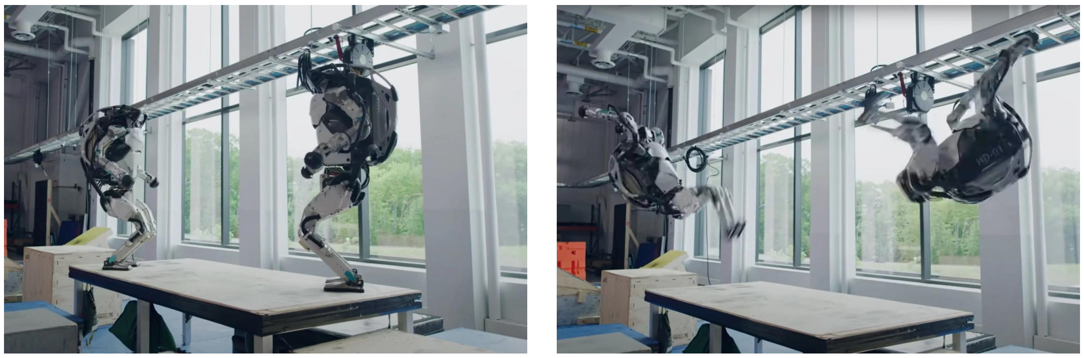

Figure 1.2. Two Atlas humanoid robots developed by Boston Dynamics performing back flips off of an elevated surface.

Figure 1.3. On the left is a Boston Dynamics Spot Robot at a power plant. Recently, these robots have been deployed in industrial settings to inspect high-risk equipment to avoid placing human lives at risk. On the right is a snapshot of the MIT Mini Cheetah Robot running over a patch of ice. MIT researchers have demonstrated the incredible robustness of learning-based control strategies for quadrupeds when blindly maneuvering in diverse environments [33].

In pursuit of this goal, there is an increasing interest in adapting these methods to extreme environments with complex terrain and unmodeled disturbances. In this thesis, we consider two remaining challenges: (1) enabling legged system to carefully choose contact sequences in cluttered or discontinuous terrain and (2) learning-based techniques to adapt to unmodeled dynamics. The next section provides some background on past work in robotic planning and control.

1.2 Previous Research

1.2.1 Planning Methods

The motion planning problem in robotics is the task of finding the optimal (or sometimes just feasible) path for a robot to travel from a starting point to its goal while satisfying some constraints, which can include obstacle avoidance as well as the kinematic and dynamic limitations. The high-order, nonlinear, and hybrid nature of the dynamics of legged robots makes planning their motions especially challenging due to fundamental limitations such as overcoming local optima and keeping the computation time tractable to be solved online.

Planning frameworks for legged robots often formulate the problem as a trajectory optimization (TO) problem over continuous variables. The problem can be directly transcribed as

$$\begin{aligned} \min_{\boldsymbol{x}(t),\boldsymbol{u}(t)} \quad & J(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), \boldsymbol{u}(t)) \\ s.t. \quad & \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ & \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \\ & \text{other constraints on } \boldsymbol{x}(t) \text{ and } u(t) \end{aligned} \tag{1.1}$$

where $\boldsymbol{x}$ is the robot's state trajectory, $\boldsymbol{u}$ is the control input trajectory, J is a cost function, and f is the robot's dynamics. The cost function is often set to minimize the control effort or energy consumption of the robot, and when given a reference trajectory, the cost function can also include terms to minimize tracking error. The problem can also be formulated as a collocation problem where $\boldsymbol{x}$ and $\boldsymbol{u}$ are parametrized by piecewise polynomial functions or a shooting problem where only $\boldsymbol{u}$ is a decision variable.

The TO problem as presented can be passed to state-of-the-art nonlinear program (NLP) solvers (i.e. IPOPT [55]) to solve for the optimal trajectory and its corresponding control tape. However, it's more common to approximate the robot's dynamics f using a simplified model (i.e. a single rigid body model (Section 2.3) or the spring-loaded linear inverted pendulum [45, 53]) or a Taylor linearization to make Equation 1.1 a convex optimization problem, avoiding issues with local optima and leveraging state-of-the-art convex optimization solvers that can solve such problems within milliseconds. After making such approximations, guaranteeing kinematic and dynamic feasibility becomes more challenging without making the robot's motion too conservative.

Popular planning methods for wheeled and tracked mobile robots usually involve some form of random sampling and search methods to find an optimal trajectory that avoids obstacles. For example, the probabilistic road map method involves sampling random points in the environment, checking if the point is in an obstacle-free region, computing a feasible path to there from another node, and then performing a graph search algorithm such as Dijkstra Algorithm or A* [24]. Other common methods are variants of the rapidly-exploring random tree (RRT) algorithm that uses similar principles to build a tree of feasible paths throughout free space. These methods are less popular for legged robots because its difficult to check for kinematic and dynamic feasibility between two sample points within a reasonable time limit. However, recent work has addressed these challenges to extend RRT to dynamic legged robots [37].

1.2.1.1 Contact-Implicit Planning

A grand challenge in legged robotics is simultaneously optimizing the contact sequence and the whole body trajectory within a single problem framework over a sufficiently large time horizon at real-time rates. It's common for motion planning frameworks to use fixed gait patterns to avoid this problem, while contact-implicit TO methods remove the fixed-gait assumption, allowing the robot to simultaneously optimize its state, control inputs, and contact sequence [44, 31]. While recent work shows promising progress [9, 4], it remains a challenge to solve this highly non-convex problem online.

One aspect of the contact planning problem that makes it challenging is that it inherently involves choices over discrete and continuous variables. A walking robot must first choose what surface it will place its next footstep (discrete) before deciding where it will place its foot on that surface (continuous). Such problems fit nicely within a mixed-integer program (MIP), where the decision variables include integer and continuous variables. Within a MIP, mixed-integer convex constraints can be used to replace and approximate the non-convex constraints that arise from contact dynamics [11, 51]. Additionally, these programs can be solved to global optimality.

However, fundamental scalability challenges within MIP make this approach difficult to implement at real-time rates. The work presented in Chapter 2 makes progress toward addressing these challenges to plan 3D hopping motions for a legged robot through cluttered environments.

1.2.2 Control Methods

Due to the high-order nature of legged robots, it's common to approach the control problem using a hierarchical structure. At the lowest level, motor torques are often computed using a proportional-derivative (PD) controller with a feedforward torque that comes from either an online or offline TO problem. Control approaches at the next level generally separate into two camps: model-based control methods and reinforcement learning (RL) based methods.

Model-based control methods leverage dynamics models to compute control inputs online, often by solving an optimization problem with an onboard computer. Computational limitations force tradeoffs in model complexity, so many approaches approximate the dynamics to fit them within a quadratic program (QP). However, recent research has improved the performance of NLP solvers [16, 34] allowing the use of nonlinear dynamics in online control frameworks [18]. Some of these methods are purely reactive, meaning they only compute the control input for the current timestep, while others solve a TO problem to compute the control input over a finite time horizon in a model predictive control (MPC) fashion. In an MPC framework, the robot executes only the first one or two control inputs before solving a new finite horizon TO problem. Most state-of-the-art controllers use this approach for dynamic motions to benefit from rapid replanning to recover from disturbances and anticipatory control actions based on its model [13, 25].

More recently, some of the robotic control community shifted its focus toward deep RL, where the optimal control policy is approximated using a deep neural network

(DNN). In RL, the control policy is learned through an offline training phase by rolling out trajectories with the current policy (occasionally taking random actions), computing a reward function, and then stepping the parameters of the DNN along the gradient of the reward function to iteratively approach an optimal policy. Learning-based methods excel at solving problems that are difficult to model [36, 22, 23] due to their ability to learn a latent representation of the robot's state that includes key details about its dynamics and the environment that are difficult to capture with physics-based models. However, a challenge with RL is that the policy may fail under conditions neglected during training. Lee et al. [26] solved this issue using domain randomization to expose the controller to terrains with varying shapes and friction coefficients and robots with varying mass, inertia, and link length properties, resulting in a control policy with impressive robustness across diverse environments.

1.2.3 Learning-based Adaptive Control

An ongoing challenge in legged robotics is adapting to effects that were ignored in the model of the base controller or during its offline training phase. One promising approach is to formulate learning techniques to estimate the effects of unmodeled dynamics on the robot. In [41], Pandala et al. consider an implementation of convex MPC based on a single-rigid body model with an extra term that accounts for the unmodeled dynamics. They train a neural network through offline RL to compute the set of unmodeled dynamics for their framework. In [40], O'Connell et al. demonstrate a learning-based control framework that allows a quadcopter to adapt to time-varying wind conditions. They train a DNN offline to compute a basis for the unmodeled dynamics on the quadcopter, and they then use an adaptive control law online to update a set of linear coefficients that mix the outputs of the DNN to estimate the dynamic effects due to the wind. In Chapters 3 and 4, we present a framework that is similar in spirit to [41] and [40] but focuses on the response of the system to unmodeled dynamics rather than estimating them directly. Our approach allows a walking robot to return to its stable steady-state behavior when disturbed by unmodeled dynamics.

1.3 Objectives

Our first objective is to develop a motion planning framework to optimize maneuvers through cluttered and discontinuous terrain in real-time. In Chapter 2, we present a MIP formulation to plan dynamic 3D hopping motions for a legged robot while optimizing footstep locations and considering a variety of constraints related to 3D orientation dynamics, actuation limits, and obstacle avoidance. Our approach differs from past approaches in that it simplifies the dynamics via the consideration of an impulsive stance phase, allowing us to extend our MIP formulation to plan aggressive hopping motions in 3D without introducing a significant computational bottleneck. We found that our impulsive stance formulation typically finds a solution to a 4 hop planning problem within 1 second, even in larger environments where an equivalent full stance formulation fails to find a solution within 10 seconds (Section 2.7).

Our second and third objectives are separate components in a single learning-based framework to adapt a legged robot to dynamics neglected by its base controller. Rather than learning a control policy end-to-end as in RL, we consider an approach that simplifies the learning problem into two distinct tasks: predicting the steady-state behavior of the robot and identifying its natural response to a disturbance [27]. In the context of legged locomotion, the steady-state behavior can be understood as the steady-state output of the robot along an asymptotically stable periodic orbit (a.k.a stable limit cycle and attractor), while the natural response is any feasible trajectory that connects distinct periodic orbits. Past work has shown that both the steady-state behavior and natural dynamics of the system can be learned through well-established online regression techniques to adapt the step-length controller for Raibert's hopper to uneven terrain [27].

Decomposing the problem into learning the system's steady-state behavior and natural response offers several benefits. Once we have an estimate of the steady-state output of the system, we can subtract it from the true output to calculate the system's natural response. After removing its periodic component, the remaining output is hyperbolic in nature. It follows from the Hartman-Grobman theorem that the local behavior of a system near a hyperbolic equilibrium point can be captured by linearization techniques. Past work has leveraged this observation by applying the dynamic mode decomposition with control (DMDc) algorithm [46] to learn a linear state-based model of the error system [27]. With a linear approximation of the natural dynamics, we can choose from a suite of available control options (i.e. linear quadratic regulator, linear model predictive control (MPC), passivity-based feedback, etc.) to force the system back to its steady state behavior.

Our second objective is to develop methods for estimating the steady-state response of the robot when driven by a disturbance from the environment (Chapter 3). We introduce a new technique that relies on a transformer neural network [52] trained offline to predict the steady-state output of the system. We also demonstrate fine-tuning the neural network online by removing its final layer and replacing it with an adaptable layer of linear coefficients, which we update in response to new conditions that may vary from previous training environments.

The third objective is to implement the DMDc algorithm to learn a linear approximation of the error system between the true output of the robot and our estimate for its steady-state behavior (Chapter 4). After learning the dynamics of the error system, we implement a passive feedback controller to regulate the system to its stable steady-state behavior. We present preliminary simulation results of applying our framework to adapt a quadruped robot to unmodeled dynamics while walking on a treadmill with a time-varying speed setting.

In Chapter 5, we discuss some key findings from this work and how they inform directions for future research in contact-implicit planning and learning-based methods to adapt to new conditions online.

CHAPTER 2

3D HOPPING IN CLUTTERED TERRAIN USING IMPULSE PLANNING WITH MIXED-INTEGER STRATEGIES

2.1 Introduction

Legged robots have an advantage over other types of robots because they can traverse complex environments with uneven terrain, gaps, and obstacles. However, a number of factors make it difficult to plan optimal motions for a robot in these environments. For example, planning footsteps can be difficult when the available contact locations are discontinuous—many state-of-the-art trajectory optimization (TO) methods for legged robots use numerical optimization techniques on continuous variables that struggle with the inherently combinatorial problem of choosing a contact surface. Also, the planner must navigate the robot through clutter to avoid collisions with the environment. These challenges are further complicated when executing dynamic movements, since each action must satisfy friction constraints, actuator torque limits, and the robot's dynamics to guarantee the motion is physically feasible. The high-order, nonlinear, and hybrid nature of the dynamics makes guaranteeing feasibility challenging. To address these challenges, this chapter describes a method to find an optimal trajectory for a legged robot to hop through a cluttered environment. An example of a simulated hopping environment with convex safe regions to step and obstacles is shown in Figure 2.1

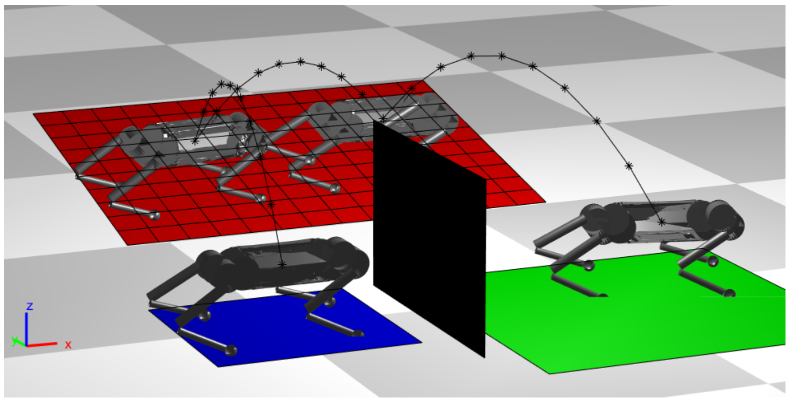

Figure 2.1. An example of a hopping environment. The colored platforms denote convex safe regions for footsteps. The red platform is angled at 10 degrees about the x-axis, the green platform marks the goal position, and the black wall is an obstacle. The black line is the COM trajectory from the MIP solution for this environment.

2.1.1 Previous Work

There are a number of existing approaches to determine feasible trajectories for a legged robot across complex terrain. The problem can be formulated as a nonlinear programming problem (NLP) where all the optimization variables must be continuous. In [42], the MIT Cheetah 2 could solve a nonlinear TO problem online to find a feasible trajectory over a sensed obstacle. Winkler et al. [54] uses a phase-based parameterization of each foot's position and contact force to keep these variables continuous. This approach optimizes the gait sequence of the robot and even allows a full flight phase across gaps. While recent research has improved the numerical stability and speed of NLP solvers (e.g., [16, 34]) with impressive recent demonstrations [18], these methods remain fundamentally prone to become stuck in local optima near the initial guess and lack an ability to search over contact options.

Another approach is to formulate the problem as a mixed-integer program (MIP), in which mixed-integer convex constraints can be used to replace and approximate the non-convex constraints from the dynamics and kinematics of the legged robot.

Such an approach enables handling discontinuous terrains with existing MIP solvers (e.g., [19]) that include search strategies via branch and bound to avoid local optima from the non-convex MIP constraints. Due to fundamental scalability limitations in MIP formulations, it is common to adopt simplified models, most often focused on the center of mass (COM) or the centroidal dynamics, to generate a motion sketch for subsequent whole-body planning. Deits and Tedrake [11] use a MIP that considers kinematic reachability constraints to plan footsteps for the Atlas Humanoid robot on uneven terrain. They limit the reachable set of footstep positions to a convex set near the robot and leave considering torque limits in a MIP as future work. Valenzuela [51] extended this work to consider a key subset of the robot's dynamics by using a McCormick Envelope [35] to approximate the bilinear terms in a centroidal dynamics model. Aceituno et al. [1] plan motions for a quadruped robot in 3D using a MIP. They also constrain contact locations to a region near the (COM), so their formulation is not suitable for (nor designed for) hopping long distances. Ding et al. [14, 15] use a MIP to plan impressive dynamic motions for a single-legged and a multi-legged robot, respectively, while satisfying torque limits. Their framework addresses the 2D case, while our approach addresses 3D environments and includes 3D orientation dynamics.

2.1.2 Contribution

The contribution of this chapter is to detail a new method for planning dynamic hopping trajectories over cluttered terrain that simplifies the dynamics via the consideration of an impulsive stance phase. The influence of this simplification is that it reduces the number of decision variables during stance, allowing us to extend the MIP formulation to address many other constraints related to 3D orientation dynamics, contact choice, and obstacle avoidance, without introducing a significant computational slowdown. We found that on average our impulsive stance formulation finds a solution to a 4-hop planning problem within 1 second, even in larger environments where the full stance formulation fails to find a solution within 10 seconds (Section 2.7). Additionally, we show that we can further reduce the average solve time by adding supplementary problem-specific constraints and A*-like heuristics to the cost function. Compared to previous MIP locomotion planners, this approach of simplifying the contact dynamics provides a more coarse motion sketch. However, we show that an NLP polishing step (with contacts fixed) restores dynamic feasibility, while also enabling the MIP to bias the NLP toward a favorable local optimum. While our NLP polishing step considers a single rigid body model, more sophisticated whole-body solvers could, in principle, be used as a stand-in for this step. Overall, the work provides a new perspective on how high-level MIP planners can be teamed with NLP solutions for motion synthesis in challenging terrain.

2.2 Overview

The system architecture developed in this work is shown in Fig. 2.2. The main components of the work are the MIP formulation for dynamic navigation (Section 2.4) and the NLP for motion polishing (Section 2.6), while the remaining components employ a nonlinear model predictive control (MPC) strategy, as in [29], to compute ground reaction forces that simultaneously track the hopping trajectory and stabilize the robot via feedback. The simulation results (Section 2.7.3) also include a state estimator for the base position/orientation and velocities, which shows additional robustness of the architecture proposed.

The MIP planner considers alternating periods of stance and flight and selects contact patches on each jump to enable the quadruped to move dynamically through clutter. The motion plan also considers the evolution of the body orientation, both in terms of the roll and pitch it must achieve to land on each contact patch, as well as the heading yaw angle. A notable distinction of this planner is that it simplifies

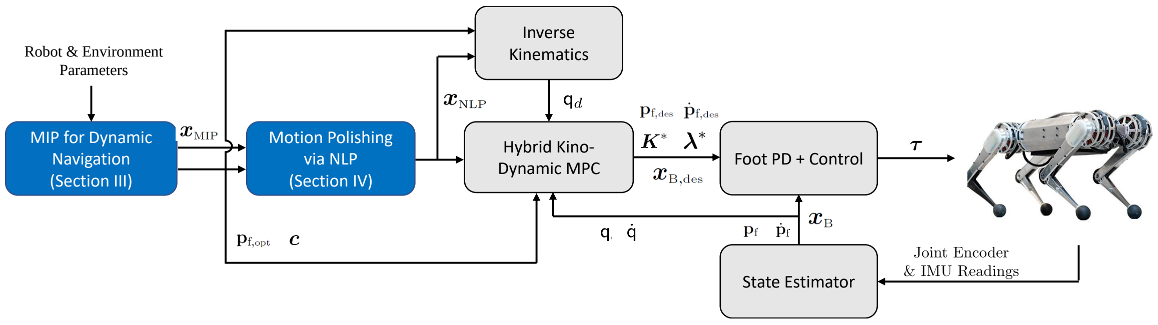
Robot & Environment Parameters
MIP for Dynamic Navigation (Section III)
$\boldsymbol{x}_{\text{MIP}}$
Motion Polishing via NLP (Section IV)
$\boldsymbol{x}_{\text{NLP}}$
Inverse Kinematics
q_d
Hybrid Kino-Dynamic MPC
$\text{p}_{\text{f,des}}$ $\dot{\text{p}}_{\text{f,des}}$
$\boldsymbol{K}^*$ $\boldsymbol{\lambda}^*$
$\boldsymbol{x}_{\text{B,des}}$
Foot PD + Control
$\boldsymbol{\tau}$
$\text{p}_{\text{f,opt}}$ $\boldsymbol{c}$
q $\dot{\text{q}}$
p_{f} $\dot{\text{p}}_{\text{f}}$
$\boldsymbol{x}_{\text{B}}$
State Estimator
Joint Encoder & IMU Readings

Figure 2.2: System overview. Blue blocks are the main components introduced, with the MIP formulation representing the main contribution of the work.

each stance to be a single time step with fixed configuration (i.e., it considers each stance as impulsive). This reduction enables this work to plan 3D orientation and the COM together for navigation in cluttered terrain while maintaining a reasonable average solve time, which has not been demonstrated experimentally in prior MIP locomotion work.

Due to the impulse assumption, the COM trajectories have non-physical velocity discontinuities at transitions to and from stance. The NLP-based motion polisher takes the reference wrench and COM trajectories along with the fixed chosen contact patches to generate smooth COM trajectories that have better dynamic fidelity than with the coarse-grained MIP planner. In this work, our NLP polisher considers a single-rigid-body model, although more sophisticated whole-body trajectory optimizers could also be used. The use of this motion polisher, in particular, smooths the contact impulse during stance and also considers a higher-fidelity model for the orientation evolution during flight. We introduce the Single Rigid Body model used in this work in the following section.

2.3 Single Rigid Body Model

The full body dynamics equations of legged robots are highly nonlinear and nonconvex, making them difficult to incorporate into an optimization problem without facing the challenges of overcoming undesirable local optima. By leveraging key aspects of the hardware design of quadruped robots, namely the low inertia legs and concentration of mass in the torso, we can make simplifying assumptions to reduce model complexity while still capturing key aspects of the robot's dynamics.

We follow spirit of centroidal dynamics planning [38] and adopt a single-rigid-body (SRB) model of the robot for high-level planning. In the SRB model, the total spatial wrench on the robot is the sum of the wrench at each contact. Thus, the net spatial wrench $\boldsymbol{\mathcal{F}} \in R^6$ [17] is given by

$$\boldsymbol{\mathcal{F}} = \sum_{i=1}^{N_{\text{EE}}} \begin{bmatrix} \mathbf{n}_i \\ \mathbf{f}_i \end{bmatrix} - m\, g, \quad \mathbf{n}_i = \mathbf{r}_i \times \mathbf{f}_i, \tag{2.1}$$

where N_{EE} is the number of end effectors of the robot, $\mathbf{f}_i \in R^3$ is the ground reaction force (GRF) at contact point i, m is the total mass of the robot, $g = (\mathbf{0}_{3\times 1}, \mathbf{g})$ is the spatial acceleration due to gravity, $\mathbf{n}_i \in R^3$ is the moment due to the force at contact i, and $\mathbf{r}_i \in R^3$ is the vector pointing from the COM to contact point i.

The net spatial force is then related to the angular acceleration $\dot{\boldsymbol{\omega}} \in R^3$ and COM acceleration $\ddot{\mathbf{p}}_{\text{COM}} \in R^3$ according to the Newton and Euler equations

$$\boldsymbol{\mathcal{F}} = \begin{bmatrix} \mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} \\ m\ddot{\mathbf{p}}_{\text{COM}} \end{bmatrix}, \tag{2.2}$$

where $\mathbf{I}$ is the rotational inertia of the body. The SRB model assumes that the mass of the robot's legs is negligible, which is a valid assumption for many legged robots because the mass of the body greatly outweighs the mass of the legs. In [13], Di

Carlo et al. used an SRB model to achieve a variety of gaits including a flying trot, pronking, and galloping, on the Cheetah 3 robot. The SRB model is sufficient for planning footstep locations in our MIP, and we can later substitute an improved model in our NLP as necessary.

2.4 MIP Formulation for Dynamic Navigation

Our MIP formulation borrows inspiration from the work in [11, 15] while providing a unified treatment of 3D orientation evolution, friction limits, and actuation limits that is lacking in past work. Our stance constraints described in Section 2.4.1 consider a fixed nominal pose at touchdown and consider an impulse in wrench space to address actuation and friction limitations in a unified manner. Since stance is treated as impulsive, we isolate our collision avoidance constraints to flight (Section 2.4.2), and couple flight and stance via the chosen footsteps at touchdown and takeoff (Section 2.4.3). We add an additional kinematic constraint (Section 2.4.4) to avoid local infeasibility errors during the NLP-polishing step. The complete formulation of our MIP is presented in 2.4.5.

2.4.1 Impulsive Stance Formulation

In our MIP, we simplify each stance phase to be impulsive, meaning that each stance is a single time-step with a fixed configuration. This change reduces the number of decision variables in our optimization, leading to a reduction in average solve time compared to a full stance formulation, even when including the extra time for the NLP-polishing step. We further reduce the number of decision variables during stance by planning the net spatial wrench of the robot rather than GRFs, also avoiding the bilinear terms in (2.1). While state-of-the-art MIP solver Gurobi [19] can solve MIPs with bilinear constraints, such constraints increase the complexity of the optimization, sometimes creating a significant computational slowdown. Considering the net spatial wrench is useful when checking the dynamic feasibility of a trajectory, while the nonlinear MPC framework can determine GRFs on-the-fly to track the trajectory from our optimization.

2.4.1.1 Wrench Feasibility

We define the robot's configuration as $\mathbf{C} = (\mathbf{p}_{\text{COM}}, \boldsymbol{\Theta}, \mathsf{q}, c)$, where $\boldsymbol{\Theta}$ are the Euler angles of the trunk, $\mathbf{p}_{\text{COM}}$ is the COM position, q are the joint angles and c is the contact status of each foot. When given the robot's current configuration, we can compute the set of all possible wrenches that the robot can generate on its COM by considering its actuation capabilities, the surface normals at each contact, and the friction coefficient of the terrain. The resulting 6-polytope is commonly referred to as the feasible wrench polytope (FWP) [39]. With the FWP, constraining a wrench to be physically feasible is as simple as adding the linear constraint,

$$\mathbf{A}_{\text{FWP}}(\mathbf{C})\,\boldsymbol{\mathcal{F}} \leq \mathbf{b}_{\text{FWP}}(\mathbf{C}), \tag{2.3}$$

where $\mathbf{A}_{\text{FWP}}(\mathbf{C})$ and $\mathbf{b}_{\text{FWP}}(\mathbf{C})$ is the minimum half-space form of the FWP in configuration $\mathbf{C}$. However, it's too expensive to compute the FWP for a given configuration online. In [15], Ding et al. address this issue by discretizing the configuration space ($\mathbf{C}$-space) into N_d distinct convex polytopes and enforcing the robot to be in one of these regions at all times during stance with integer variables in their MIP. They underapproximate the FWP of each region in $\mathbf{C}$-space by computing the intersection of the FWPs at each vertex of the region. This approach is realizable in a low-dimensional $\mathbf{C}$-space (3-dimensional in [15]) but becomes unwieldy in higher dimensions.

As an alternative, we compute offline the FWP for the nominal configuration defined by the following criteria: zero Euler angles, equidistant COM position from each contact at a fixed height, and feet positioned just outside of the hips. Within our MIP, we constrain the wrench during each impulsive stance phase to be within this nominal FWP, and we then add a constraint to our NLP-polishing step that restricts COM position and orientation during stance to a bounding box centered around the nominal configuration. The bounding box constraint assures the robot stays near the configuration where the FWP is valid and prevents configurations where the feet cannot reach the desired contact positions. We found this approximation to be sufficiently conservative to keep the torques within the actuation limits, while still allowing for longer jumps with rotations. We also add an additional kinematic constraint to our MIP to assure the robot can remain within the bounding box across its entire stance phase (Section 2.4.4), preventing the possibility there's not a feasible solution to the NLP near the MIP solution. The next section describes how we compute the FWP.

2.4.1.2 Computing the FWP

We use the open-source Multi-Parametric Toolbox (MPT) for MATLAB [20] and follow spirit of the method in [39] to precompute the feasible wrench polytope (FWP) in the robot's nominal configuration. The FWP is the intersection of the contact wrench cone (CWC) and the actuation wrench polytope (AWP), where the CWC considers friction constraints and the AWP considers actuation constraints. Rather than compute this intersection directly, we instead compute the intersection of the friction cone and actuation force polyhedron (AFP) at each contact and then calculate their Minkowski sum, thereby computing N_{EE} intersections of polyhedrons (3D) instead of computing a single intersection of two 6-polytopes.

The friction cone at each contact i, FC_i, with friction coefficient μ is

$$\sqrt{\mathbf{f}_{x,i}^2 + \mathbf{f}_{y,i}^2} \leq \mu\, \mathbf{f}_{z,i} \geq 0, \tag{2.4}$$

where $\mathbf{f}_{z,i}$ is the z-component of the force at contact i. We approximate 2.4 as a polyhedron,

$$\begin{bmatrix} 1 & 0 & \frac{-\mu}{\sqrt{2}} \\ -1 & 0 & \frac{-\mu}{\sqrt{2}} \\ 0 & 1 & \frac{-\mu}{\sqrt{2}} \\ 0 & -1 & \frac{-\mu}{\sqrt{2}} \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} \mathbf{f}_i \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \mathbf{f}_{z,\max} \\ 0 \end{bmatrix}, \tag{2.5}$$

where $\mathbf{f}_i \in R^3$ is the force at contact i. This conservative approximation simplifies the second-order cone constraint to a linear constraint and can be made as close as desired to the true friction cone by adding more faces to the polyhedron. We chose a four-sided polyhedron, which is sufficient even for longer hops. Since we will eventually calculate the intersection of the friction cone with the AFP, we can set $f_{z,\max}$ at an arbitrarily large value. Figure 2.3 contains a plot of a polyhedral friction cone with μ =0.7 and $\mathbf{f}_{z,\max}$ =150 Newtons.

We construct AFP$_i$ by calculating each of its vertices,

$$\mathbf{f}_{v,i} = -\boldsymbol{J}(\mathsf{q}, i)^{+T}\, \tau_{\max,v}, \quad \forall v = 1, \ldots, 2^n,\ i = 1, \ldots, N_{\text{EE}} \tag{2.6}$$

where $\mathbf{f}_{v,i}$ is the v^{th} vertex, $\boldsymbol{J}(\mathsf{q}, i)^{+T} \in R^{3\times n}$ is the pseudo-inverse of the transposed contact jacobian for end effector i, and $\tau_{\max,v} \in R^n$ is the v^{th} combination of max torques generated by its n actuated joints. The polyhedron will have 2^n vertices because each element of a $\tau_{\max,v}$ can be the positive or negative of the max torque at the corresponding joint. Once we calculate these vertices, we can pass them to MPT to calculate the minimum half-space form.

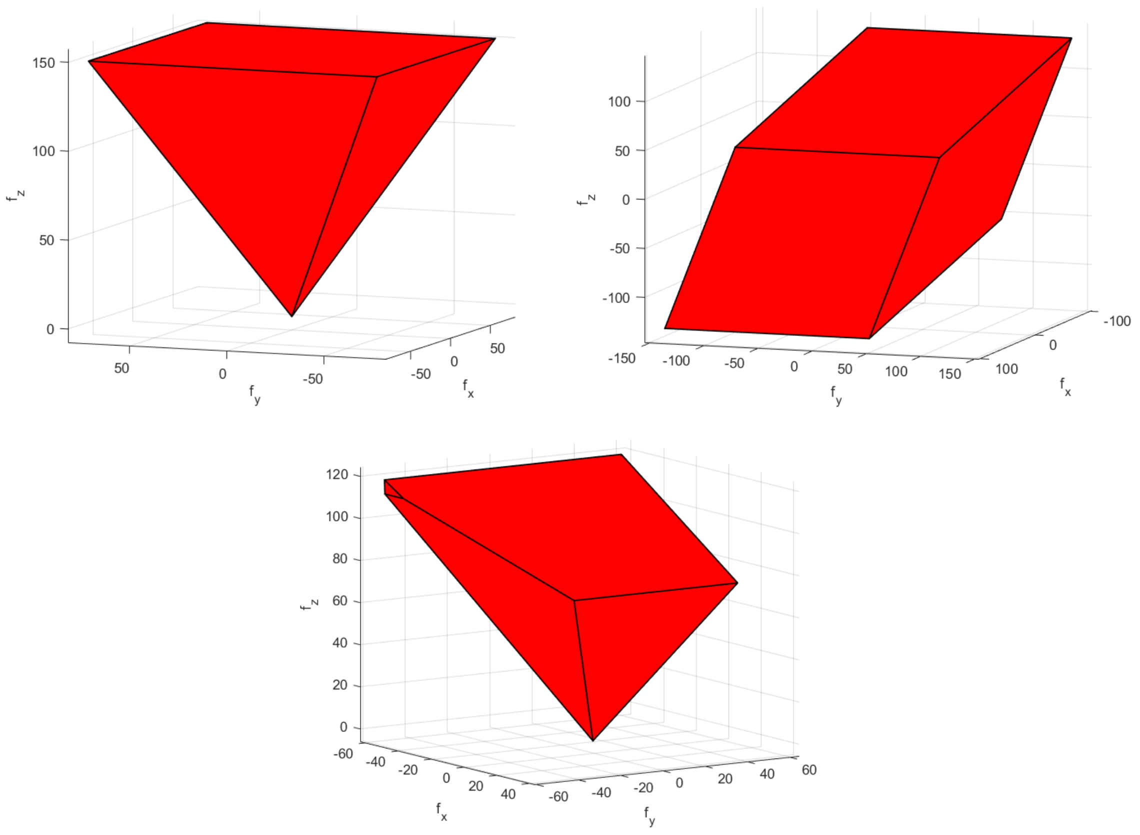

Figure 2.3. Shown on the top left is a polyhedral friction cone with $\mu = 0.7$ and $\mathbf{f}_{z,\max} = 150$ Newtons. On the top right is the AFP for the front right leg of the MIT Mini Cheetah where the maximum torque for each joint is approximated to be ± 17 Newton-meters. The bottom plot is the intersection of the two above polyhedrons—the FFP for the front right leg. Any vector in the FFP satisfies the friction constraints and actuation limits of the MIT Mini Cheetah and thus is a dynamically feasible force for the front right leg.

We calculate the feasible force polyhedron at contact i, FFP_i, as

$$\mathrm{FFP}_i = \mathrm{FC}_i \cap \mathrm{AWP}_i. \tag{2.7}$$

Since we take the intersection, each vector in FFP_i is a force the robot can generate on its COM with contact i, satisfying both friction and actuation constraints while in its nominal configuration. We extend FFP_i to FWP_i by calculating the corresponding moment generated by the force at each vertex and stacking the two vectors into a wrench,

$$\boldsymbol{\mathcal{F}}_{v,i} = \begin{bmatrix} r_i \times \mathbf{f}_{v,i} \\ \mathbf{f}_{v,i} \end{bmatrix} \in R^6, \tag{2.8}$$

where $\boldsymbol{\mathcal{F}}_{v,i}$ is the v^{th} vertex of the FWP of contact i. We calculate $\boldsymbol{\mathcal{F}}_{v,i}$ for all of the vertices of FFP_i and then pass them to MPT to calculate the minimum half-space form of the FWP at contact i, FWP_i. We take the Minkowski sum of the FWPs at each contact to calculate the total FWP,

$$\mathrm{FWP} = \mathrm{FWP}_1 \bigoplus \mathrm{FWP}_2 \bigoplus \cdots \bigoplus \mathrm{FWP}_{N_{\mathrm{EE}}}. \tag{2.9}$$

The Minkowski sum $\mathrm{FWP}_1 \bigoplus \mathrm{FWP}_2$ is the result of adding each vector in FWP_1 to each vector in FWP_2, $\{\boldsymbol{\mathcal{F}}_1 + \boldsymbol{\mathcal{F}}_2 | \boldsymbol{\mathcal{F}}_1 \in \mathrm{FWP}_1, \boldsymbol{\mathcal{F}}_2 \in \mathrm{FWP}_2\}$. Each vector in the FWP is a feasible wrench the robot can generate on its COM while in its nominal configuration.

2.4.2 Flight Formulation

We plan a single touchdown (TD) point $\mathbf{q}_{\mathrm{TD}} = (\mathbf{p}_{\mathrm{TD}}, \boldsymbol{\Theta}_{\mathrm{TD}}) = (x_{\mathrm{TD}}, y_{\mathrm{TD}}, z_{\mathrm{TD}}, \theta_{\mathrm{TD}}, \phi_{\mathrm{TD}}, \psi_{\mathrm{TD}}) \in R^6$ for each stance phase of the robot. Here, $\mathbf{p}_{\mathrm{TD}}$ marks the midpoint of the robot's contact points in Cartesian space, and the

$\mathbf{\Theta} \in R^3$ gives the Euler angles for the trunk. We denote $\mathbf{p}_{\text{COM},i} \in R^{3\times N_{\text{pts}}}$ as the COM trajectory for flight phase i in Cartesian space, where N_{pts} is the number of sample points in a flight phase. We add flight kinematics constraints,

$$\begin{aligned}
\mathbf{p}_{\text{COM},i,j+1} &= \mathbf{p}_{\text{COM},i,j} + \dot{\mathbf{p}}_{\text{TO},i} T_{\text{air},i,j} + \frac{1}{2}\, g\, T^2_{\text{air},i,j}, \\
T_{\text{air},i,j} &= \frac{1}{N_{\text{pts}}-1} T_{\text{air},i}, \\
\mathbf{\Theta}_{\text{TD},i+1} &= \mathbf{\Theta}_{\text{TD},i} + \dot{\mathbf{\Theta}}_{\text{TO},i} T_{\text{air},i}, \\
\forall i = 1, ..., N_{\text{hops}}&, \forall j = 1, ..., N_{\text{pts}} - 1
\end{aligned} \tag{2.10}$$

where $\dot{\mathbf{q}}_{\text{TO},i} = (\dot{\mathbf{p}}_{\text{TO},i}, \dot{\mathbf{\Theta}}_{\text{TO},i})$ is the i^{th} take-off velocity, $T_{\text{air},i}$ is the flight time for the i^{th} hop, and N_{hops} is the number of hops in the trajectory.

Gurobi can handle the bilinear terms in (2.10) [19]. However, we found a piecewise linear approximation for T^2_{air} as in [14] empirically decreased the average solve time. We can approximate a nonlinear constraint using a piecewise linear approximation by assigning integer variables $z^T_{l,i}$ to each piece l of the function at hop i and adding constraints

$$\begin{aligned}
z^T_{l,i} \Rightarrow &\begin{cases} T^l_{\text{air}} \leq T_{\text{air},i} \leq T^{l+1}_{\text{air}}, \\ T^2_{\text{air}} = m^T_l\, T_{\text{air},i} + b^T_l, \end{cases} \qquad \forall l = 1, \ldots, N_T, \\
&\sum_{l=1}^{N_T} z^T_{l,i} = 1 \quad \forall i = 1, \ldots, N_{\text{hops}},
\end{aligned} \tag{2.11}$$

where N_T is the number of pieces in the approximation. The arrow means that if $z^T_{l,i} = 1$, then the constraints are enforced. We implement the constraints using the Big-M trick through YALMIP's `implies()` syntax [30].

The Big-M trick can be written as

$$A_{j,k}\mathbf{p}_{com,i} \leq b_{j,k} + M\, z^{\text{COM}}_{i,j,k}, \tag{2.12}$$

where M is a large number. If $z_{i,j,k}^{\text{COM}} = 1$, the half-space constraint can be ignored due to the large number on the right side of the inequality, else the half-space constraint is enforced. The bottom constraint in Equation 2.11 enforces that only one piece of the linear approximation is imposed for each hop, while the rest are ignored. We can improve the approximation by adding more pieces, but this also increases the number of integer variables in our formulation, which also increases the worst-case algorithmic complexity. Figure 2.4 shows an example of a piecewise linear approximation to a nonlinear function.

2.4.2.1 Obstacle Avoidance:

We add mixed-integer constraints to avoid collisions with obstacles. We model obstacles as polytopes. For each face, we add half-space constraints so that each $\mathbf{p}_{\text{COM}}$ point is outside of the obstacle. We assign binary variables $z_{i,j,k}^{\text{COM}}$ to the i^{th} $\mathbf{p}_{\text{COM}}$ point and the k^{th} face of the j^{th} obstacle. If $z_{i,j,k}^{\text{COM}} = 0$, the half-space constraint is enforced, but if $z_{i,j,k}^{\text{COM}} = 1$, the constraint can be ignored. To avoid obstacles, we add additional constraints,

$$N_{F,j} - 1 = \sum_{k=1}^{N_{F,j}} z_{i,j,k}^{\text{COM}}, \quad \forall i = 1, \dots, N_{\text{pts}}, j = 1, \dots, N_O \tag{2.13}$$

where $N_{F,j}$ is the number of faces of the j^{th} obstacle [10] and N_O is the number of obstacles. We over-approximate each half-space constraint by a constant equal to the radius of the smallest sphere that encloses the robot to avoid collisions. In [12], Deits and Tedrake restructure the obstacle avoidance constraints in [49] by discretizing free space into convex safe regions. They assign integer variables to each space and enforce that their quadcopter is within one of these regions at all times. This approach can drastically reduce the number of binary variables in the MIP in environments with a large number of obstacles. In this work, we chose not to follow this approach to avoid the challenge of precomputing convex regions of safe space. Additionally, we considered environments with a small number of obstacles (i.e. less than 10) in our experiments, so the approach in [49] was sufficient.

In our approach, there is a trade off between N_{pts} and collision risk because we do not check if the line segment between consecutive point collides with an obstacle. Increasing N_{pts} will decrease the probability of collision. but it will also add more variables to the MIP and likely increase its average solve time. In [12], Deits and Tedrake demonstrate path planning for a drone with polynomial segments and using sums-of-squares tricks to constrain each segment to a convex safe region. We chose not to follow this approach to avoid the challenge of predicting a sufficient number of polynomial segments for each flight phase a priori.

2.4.3 Coupling Flight and Stance

2.4.3.1 Footstep Planning:

Similar tricks to Section 2.4.2.1 can be used to plan footsteps in convex safe regions of the terrain, which we will call platforms. We add binary variables denoted $z_{i,j}^{\text{TD}}$ for the i^{th} platform and j^{th} hop. The half-space constraints that constrain $\mathbf{q}_{\text{TD},j}$ to the i^{th} platform are ignored if $z_{i,j}^{\text{TD}} = 1$. We add additional constraints similar to (2.13) to avoid hops on unsafe terrain,

$$\sum_{i=1}^{N_{\text{P}}} z_{i,j}^{\text{TD}} = 1, \quad \forall j = 1, \ldots, N_{\text{hops}} + 1, \tag{2.14}$$

where N_{P} is the number of platforms. These constraints enforce that each $\mathbf{p}_{\text{TD}}$ satisfies the half-space constraints for exactly one platform. We constrain $\mathbf{p}_{\text{TD},0}$ to the current position of the robot and $\mathbf{p}_{\text{TD},N_{\text{hops}}+1}$ to the goal platform. We under-approximate each half-space constraint by a constant equal to the max distance between a contact and the midpoint of the contacts, ensuring all contacts will be on the platform. Due to our impulsive stance formulation, the robot is in a single pose during stance, so the position and orientation in $\mathbf{q}_{\text{TD}}$ determines the contacts points.

Our assumption that we have full knowledge of the environment allows us to deal with angled platforms. We precompute a body to world rotation matrix $\mathbf{R}_i$ for each platform i. We add constraints

$$z_{i,j}^{\text{TD}} \Rightarrow \begin{cases} \mathbf{p}_{\text{COM},j,0} = \mathbf{p}_{\text{TD},j} + \mathbf{R}_i \mathbf{c}, \\ \theta_j = \theta_i, \phi_j = \phi_i, \end{cases} \qquad \forall j = 1, ..., N_{\text{hops}} + 1. \tag{2.15}$$

where θ_i and ϕ_i are the pitch and roll angles of the i^{th} platform and $\mathbf{c}$ is the the vector pointing from $\mathbf{p}_{\text{TD}}$ to $\mathbf{p}_{\text{COM}}$ in the body frame for the nominal configuration. We do not fix ψ_j and instead allow the MIP to discover that the robot may need to do a "spin jump" to reach the next platform. We also add a constraint that $\mathbf{p}_{\text{COM},f,0} = \mathbf{p}_{\text{COM},f-1,N_{\text{pts}}}, \forall f = 2, ..., N_f$, to connect flight phases together after each impulse.

2.4.3.2 Take-off to Take-off Dynamics:

Since we compute the FWP for the nominal configuration, we plan $\boldsymbol{\mathcal{F}}_i$ in the robot's base frame, avoiding the need to rotate the wrench to check for dynamic feasibility. The rotation would make Equation 2.3 a bilinear constraint. We use rotation matrices to transform the robot's spatial acceleration to the world frame when constraining the impulsive relationship between touchdown velocity and the takeoff velocity. Each rotation matrix can be broken down into two parts such that $\mathbf{R}_j = \mathbf{R}_{\psi,j}\mathbf{R}_i$, where $\mathbf{R}_{\psi,j}$ handles the rotation due to ψ_j and $\mathbf{R}_i$ corresponds to the current platform i. To avoid the trigonometrics terms in $\mathbf{R}_{\psi,j}$, we approximate them with piecewise linear sinusoids (Figure 2.4), as in [11]. We fix the stance time h and

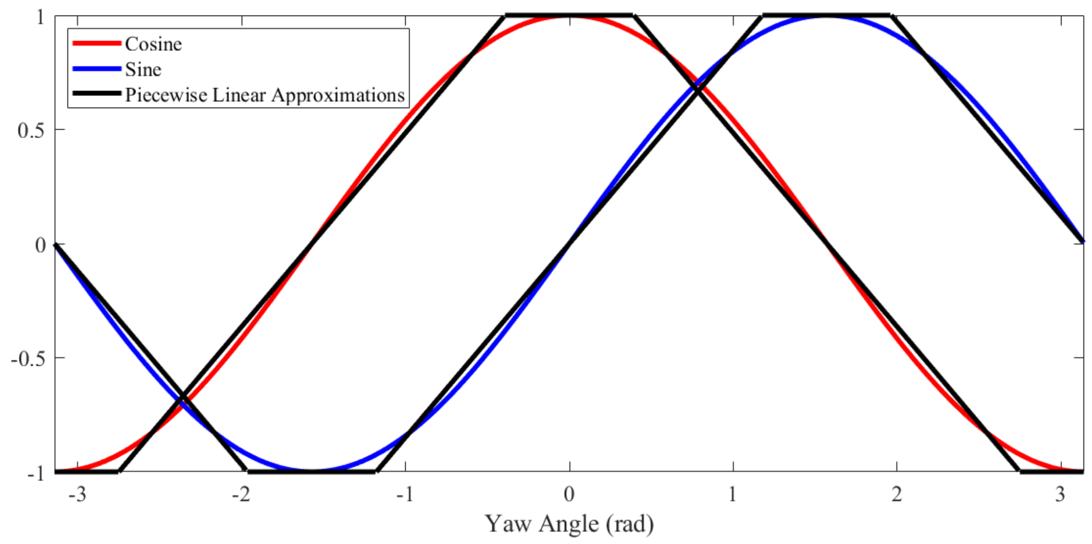

Figure 2.4. We use piecewise linear approximations of sine and cosine for rotation matrices about the z-axis of the robot, replacing the nonlinear constraints due to the sinusoids as mixed-integer linear constraints (as in Equation 2.11).

plan one spatial wrench over this time. The spatial wrench satisfies the constraint

$$z_{i,j}^{\text{TD}} \Rightarrow \dot{\mathbf{q}}_{\text{TO},j} = \dot{\mathbf{q}}_{\text{TO},j-1} + gT_{\text{air},j-1} + h(\tilde{\mathbf{R}}_j \mathbf{H}^{-1} \boldsymbol{\mathcal{F}}_j + g)) \tag{2.16}$$

where $g = (\mathbf{g}, \mathbf{0}_{3\times 1})$ is the spatial acceleration due to gravity and $\mathbf{H} = \text{blkDiag}(m\mathbf{1}_{3\times 3}, \mathbf{I})$ is the mass-inertia matrix with $\mathbf{1}_{3\times 3} \in R^{3\times 3}$ the identity matrix. Likewise the matrix $\tilde{\mathbf{R}}_j = \text{blkDiag}(\mathbf{R}_j, \mathbf{R}_j)$ to rotate both the linear and angular components of the wrench. The sum $\dot{\mathbf{q}}_{\text{TO},j-1} + gT_{\text{air},j-1}$ is the touchdown velocity at the end of hop $j-1$, while the final term is the change in velocity from touchdown to takeoff due to the impulse. Notice that this is a bilinear constraint due to the term $\tilde{\mathbf{R}}_j \mathbf{H}^{-1} \boldsymbol{\mathcal{F}}_j$.

We add an additional constraint

$$z_{g,i}^{TD} = 1 \Rightarrow \dot{\mathbf{q}}_{\text{TO},i} = 0 \quad \forall i = 1, \dots, N_{\text{hops}} + 1 \tag{2.17}$$

to address the case when N_{hops} cannot be predicted a priori. This constrains the takeoff velocity to zero when the robot is on the goal platform g, preventing any additional hops away from (and back to) the goal. With this constraint, the robot will take the number of hops needed to reach the goal and then remain stationary.

2.4.4 Bounding Box Constraint

In our proposed framework, we provide the MIP solution as a reference trajectory for an NLP with a higher fidelity dynamics model. The MIP solution biases the NLP toward a favorable local optima, but we must be careful there is a dynamically feasible solution to the NLP near the MIP solution, else the NLP solver may return a local infeasibility error. To avoid this issue, we add an additional kinematic constraint that considers the evolution of the robot's position and orientation across an impulsive stance phase. The difference in the body state $\Delta\mathbf{q}_i = (\Delta\mathbf{p}_{\text{COM},i}, \Delta\mathbf{\Theta}_i)$ at the beginning and end of the impulsive stance phase i is

$$\Delta\mathbf{q}_i = h\,(\dot{\mathbf{q}}_{\text{TO},i-1} - g\,T_{\text{air},i-1})) + \frac{h^2}{2}\,(\tilde{\mathbf{R}}_j\mathbf{H}^{-1}\boldsymbol{\mathcal{F}}_i - g). \tag{2.18}$$

We add constraints that bound $\Delta\mathbf{q}_i$ to assure that the robot can stay within a bounding box during each stance phase of the NLP. We can write this constraint only in terms of velocities, avoiding the bilinear terms,

$$z_{i,j}^{\text{TD}} \Rightarrow \begin{cases} \frac{h}{2}(\dot{\mathbf{q}}_{TO,i-1} - g\,T_{\text{air},i-1} + \dot{\mathbf{q}}_{TO,i}) \leq \tilde{\mathbf{R}}_j\Delta\mathbf{q}_{\max} \\ -\frac{h}{2}(\dot{\mathbf{q}}_{TO,i-1} - g\,T_{\text{air},i-1} + \dot{\mathbf{q}}_{TO,i}) \leq \tilde{\mathbf{R}}_j\Delta\mathbf{q}_{\max} \end{cases} \tag{2.19}$$

The bounds, $\Delta\mathbf{q}_{\max}$, which are also used for the bounding box constraint in the NLP, can be set empirically to avoid local infeasibilty errors and constrain the robot near the nominal configuration where the FWP constraint is invalid.

2.4.5 Full MIP Formulation

We now present the complete formulation of our MIP over the optimization variables $x_{\text{opt}} = (\mathbf{p}_{\text{COM}}, \mathbf{q}_{\text{TD}}, \dot{\mathbf{q}}_{\text{TO}}, \boldsymbol{\mathcal{F}}, T_{\text{air}}, T^2_{\text{air}}, C, S, z^{\text{COM}}, z^{\text{TD}}, z^T, z^C, z^S)$:

$$\min_{x_{\text{opt}}} \mathbf{J}(x_{\text{opt}})$$

subject to:

initial condition and terminal condition:

$$\mathbf{q}_{\text{TD},0} = (\phi_0, \theta_0, \psi_0, \mathbf{p}_{\text{TD},0}), \quad \mathbf{p}_{\text{TD},N_{\text{hops}}+1} \in \boldsymbol{P}_g, \quad \theta_{N_{\text{hops}}+1} = \theta_g, \quad \phi_{N_{\text{hops}}+1} = \phi_g,$$

bound free variables to improve efficiency of Gurobi's branch and bound search:

$$\mathbf{p}^l_{\text{COM}} \leq \mathbf{p}_{\text{COM},i,j} \leq \mathbf{p}^u_{\text{COM}} \quad \forall j = 1, \ldots, N_{\text{pts}},$$
$$\mathbf{q}^l_{\text{TD}} \leq \mathbf{q}_{\text{TD},i,j} \leq \mathbf{q}^u_{\text{TD}}, \quad \dot{\mathbf{q}}^l_{\text{TO}} \leq \dot{\mathbf{q}}_{\text{TO},i,j} \leq \dot{\mathbf{q}}^u_{\text{TO}}, \quad \boldsymbol{\mathcal{F}}^l \leq \boldsymbol{\mathcal{F}}_{i,j} \leq \boldsymbol{\mathcal{F}}^u, \quad \forall i = 1, \ldots, N_{\text{hops}} + 1,$$

piecewise linear approximation of T^2:

$$z^T_{l,i} \Rightarrow \begin{cases} T^l_{\text{air}} \leq T_{\text{air},i} \leq T^{l+1}_{\text{air}}, \\ T^2_{\text{air}} = m^T_l \, T_{\text{air},i} + b^T_l, \end{cases} \qquad \forall l = 1, \ldots, N_T,$$

$$\sum_{l=1}^{N_T} z^T_{l,i} = 1 \quad \forall i = 1, \ldots, N_{\text{hops}},$$

piecewise cosine and sine with N_C and N_S pieces respectively:

$$z^C_{l,i} \Rightarrow \begin{cases} \psi^{l,C} \leq \psi_i \leq \psi^{l+1,C}, \\ C_i = m^C_l \, \psi_i + b^C_l, \end{cases} \qquad \forall l = 1, \dots, N_C,$$

$$z^S_{l,i} \Rightarrow \begin{cases} \psi^{l,S} \leq \psi_i \leq \psi^{l+1,S}, \\ S_i = m^S_l \, \psi_i + b^S_l, \end{cases} \qquad \forall l = 1, \dots, N_S,$$

$$\mathbf{R}_{\psi,i} = \begin{bmatrix} C_i & -S_i & 0 \\ S_i & C_i & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \sum_{l=1}^{N_C} z^C_{l,i} = 1, \quad \sum_{l=1}^{N_S} z^S_{l,i} = 1, \quad \forall i = 1, \dots, N_{\text{hops}} + 1,$$

flight kinematics constraints:

$$\begin{aligned}
&\mathbf{p}_{\text{COM},i,j+1} = \mathbf{p}_{\text{COM},i,j} + \dot{\mathbf{p}}_{\text{TO},i} T_{\text{air},i,j} + \frac{1}{2}\, g\, T^2_{\text{air},i,j}, \\
&T_{\text{air},i,j} = \frac{1}{N_{\text{pts}} - 1} T_{\text{air},i}, \\
&\boldsymbol{\Theta}_{\text{TD},i+1} = \boldsymbol{\Theta}_{\text{TD},i} + \dot{\boldsymbol{\Theta}}_{\text{TO},i} T_{\text{air},i}, \\
&\forall i = 1, ..., N_{\text{hops}}, \forall j = 1, ..., N_{\text{pts}} - 1
\end{aligned}$$

impulsive stance with nominal FWP constraint:

$$z^{\text{TD}}_{i,j} \Rightarrow \begin{cases} \dot{\mathbf{q}}_{\text{TO},j} = \dot{\mathbf{q}}_{\text{TO},j-1} + g T_{\text{air},j-1} + h(\tilde{\mathbf{R}}_{i,j} \mathbf{H}^{-1} \boldsymbol{\mathcal{F}}_j + g)), \\ \mathbf{R}_{i,j} = \mathbf{R}_{\psi,j} \mathbf{R}_i, \end{cases}$$

$$\mathbf{A}_{\text{FWP}}\, \boldsymbol{\mathcal{F}}_j \leq \mathbf{b}_{\text{FWP}}, \quad \forall i = 1, \dots, N_{\text{P}}, j = 1, \dots, N_{\text{hops}} + 1$$

footstep planning:

$$
\begin{aligned}
&z_{i,j}^{\mathrm{TD}} \Rightarrow \begin{cases} \mathbf{p}_{\mathrm{TD},j} \in \boldsymbol{P}_i, \\ \mathbf{p}_{\mathrm{COM},j,0} = \mathbf{p}_{\mathrm{TD},j} + \mathbf{R}_i \mathbf{c}, \\ \theta_j = \theta_i, \phi_j = \phi_i, \end{cases} \\
&\sum_{i=1}^{N_{\mathrm{P}}} z_{i,j}^{\mathrm{TD}} = 1, \quad \forall i = 1, \dots, N_{\mathrm{P}}, \quad j = 1, \dots, N_{\mathrm{hops}} + 1,
\end{aligned}
\tag{2.20}
$$

obstacle avoidance:

$$
\begin{aligned}
&z_{i,j,k}^{\mathrm{COM}} = 0 \Rightarrow A_{j,k}\, \mathbf{p}_{\mathrm{COM},i} \leq b_{j,k}, \quad N_{F,j} - 1 = \sum_{k=1}^{N_{F,j}} z_{i,j,k}^{\mathrm{COM}}, \\
&\forall i = 1, \dots, N_{\mathrm{pts}}, \quad j = 1, \dots, N_O, \quad k = 1, \dots, N_F,
\end{aligned}
\tag{2.21}
$$

bounding box constraint:

$$
\begin{aligned}
&z_{i,j}^{\mathrm{TD}} \Rightarrow \begin{cases} \frac{h}{2}(\dot{\mathbf{q}}_{TO,i-1} - g * T_{\mathrm{air},i-1} + \dot{\mathbf{q}}_{TO,i}) \leq \tilde{\mathbf{R}}_j \Delta \mathbf{q}_{\max}, \\ -\frac{h}{2}(\dot{\mathbf{q}}_{TO,i-1} - g * T_{\mathrm{air},i-1} + \dot{\mathbf{q}}_{TO,i}) \leq \tilde{\mathbf{R}}_j \Delta \mathbf{q}_{\max}, \end{cases} \\
&\qquad \forall i = 1, \dots, N_{\mathrm{P}},\ j = 1, \dots, N_{\mathrm{hops}} + 1,
\end{aligned}
\tag{2.22}
$$

where z^C and z^S are binary variables corresponding to each piece of the piecewise linear approximations of sinusoids, $\mathbf{J}(x_{\mathrm{opt}})$ is a linear or quadratic cost function of x_{opt}, $\boldsymbol{P}_i$ is platform i, and $(A_{j,k}, b_{j,k})$ is the half-space form corresponding to the k^{th} face of the j^{th} obstacle.

2.5 Problem-Specific Strategies to Decrease Average Solve Time

This section describes two constraints and cost function terms that we can add to our MIP formulation in some cases to further reduce the average solve time. We provide an assessment of the effectiveness of these strategies in Section 2.7.2

2.5.1 Hard Reachability Constraint

The binary variables z_{TD} determine the sequences of platforms the robot will hop on during its trajectory. In our current formulation, there is a binary variable for every platform during each stance phase. However, there are often combinations of z^{TD} that are nonphysical. For example, it's often impossible for the robot to hop directly from the starting position to the goal. It would be ideal if we could eliminate these nonphysical combinations within our problem formulation to reduce the size of our MIP.

We address this issue by adding extra constraints to our MIP that allow the solver to easily evaluate nonphysical combinations of z_{TD} as infeasible. Specifically, we approximate the greatest distance the robot can jump, $d_{\max}$, and add constraints

$$z^{\text{TD}}_{i,j} \Rightarrow z^{\text{TD}}_{i+1,k} = 0, \quad \forall k \quad \text{s.t.} \quad d_{j,k} \geq d_{\max}, \quad i = 1, \ldots, N_{\text{hops}} \tag{2.23}$$

where $d_{j,k}$ is the shortest distance between platforms j and k. Gurobi can dismiss nonphysical combinations of z_{TD} because they will not satisfy this constraint.

2.5.2 Forward Progress Constraint

We can further reduce the number of undesirable combinations of z_{TD} by eliminating combinations that hop on the same platform twice. We can accomplish this by adding the constraint

$$z^{\text{TD}}_{i,j} \Rightarrow z^{\text{TD}}_{k,j} = 0 \quad \forall k \quad \text{s.t.} \quad i < k \leq N_{\text{hops}} + 1, \quad i = 1, \ldots, N_{\text{hops}}. \tag{2.24}$$

Note there are some cases when adding this constraint is undesirable, such as when a platform is large and the robot can take multiple hops on it to move toward its goal.

2.5.3 A*-Inspired Cost Function

In some cases, we can further reduce the MIP's average solve time by constructing a cost function in the spirit of the A* graph search algorithm. A key component of the algorithm that differentiates it from Dijkstra's algorithm is a heuristic function for each node that often is an estimate of the cost to go to the goal. Similarly, we assign a reward B_i to each platform i and add terms to the cost function

$$-B_i z_{i,j}^{\text{TD}}, \quad \forall i = 1, \ldots, N_{\text{P}}, \quad j = 1, \ldots, N_{\text{hops}} + 1. \tag{2.25}$$

In practice, we assign rewards based on their distance from the goal and nearby obstacles, with the highest reward being at the goal and lowest reward closest to obstacles, but reward assignments could in principle be done by a higher level planner. The additional terms in the cost function bias the solver to pick a combination of $z_{i,j}^{\text{TD}}$ that directly leads to the goal.

Constructing the cost with the terms in Equation 2.25 allows Gurobi to ignore solutions that take a nonoptimal footstep trajectory toward the goal. When solving a MIP, Gurobi relaxes all the binary variables to continuous variables from range 0 to 1 and solves the resulting convex optimization problem. If the solution returns the relaxed variables as binary values, then Gurobi found the optimal solution, else it found a lower bound on the optimal cost. Gurobi also can round the binary variables in the solution to their nearest integer and solve the program to find an upper bound on the optimal cost. When the lower and upper bound converge to an established threshold, Gurobi returns a solution. With the additional cost terms in 2.25, Gurobi can dismiss a combination of $z_{i,j}^{\text{TD}}$ that offers a low reward by comparing its cost function with its upper bound.

2.6 Motion Polishing via NLP

This section discusses how we formulate our NLP that receives the MIP solution as a reference trajectory and outputs a physical COM and wrench trajectory for the robot. We fix the contact sequence, positions, and timings to those of our MIP solution, and the NLP polishes the wrench, twist, and position trajectories to remove the effects of the impulsive stance assumption. We parametrize the wrench, twist, and position trajectories in our NLP with Bézier curves to make the trajectories more smooth and feasible for the robot to track online.

Bézier curves are defined by a set of control points, in which the first and last control points are always the beginning and end of the curve. The curve is completely contained within the convex hull of all the points, and the number of control points is one more than the order of the polynomial. Within our optimization, we make the control points of the Bézier curves our optimization variables, allowing us to sample the NLP solution at a finer timestep without adding additional variables to the optimization. We denote $\boldsymbol{\alpha}^{\mathbf{q}} \in R^{(d+3)\times 6\times (N_{\text{hops}}+1)}$ as the Bézier control points for the position trajectories during stance, $\boldsymbol{\alpha}^{\dot{\mathbf{q}}} \in R^{(d+2)\times 6\times (N_{\text{hops}}+1)}$ as the control points for the spatial twist trajectories during stance, $\boldsymbol{\alpha}^{\boldsymbol{\mathcal{F}}} \in R^{(d+1)\times 6\times (N_{\text{hops}}+1)}$ as the control points for the spatial wrench trajectories during stance, $\boldsymbol{\alpha}^{\mathbf{q}_f} \in R^{3\times 6\times (N_{\text{hops}}+1)}$ as the control points for the position trajectories during flight, and $\boldsymbol{\alpha}^{\dot{\mathbf{q}}_f} \in R^{2\times 6\times (N_{\text{hops}}+1)}$ as the control points for the spatial twist trajectories during flight.

We begin by interpolating the MIP solution to a smaller timestep and choosing our desired trajectory as the Bézier curves that match our MIP solution. We set the desired control points as

$$\begin{aligned}
&\boldsymbol{\alpha}_{j,i}^{\boldsymbol{\mathcal{F}},\text{des}} = \boldsymbol{\mathcal{F}}_i, \quad \boldsymbol{\alpha}_{j,i}^{\dot{\mathbf{q}},\text{des}} = \dot{\mathbf{q}}_{TO,i}, \quad \boldsymbol{\alpha}_{j,i}^{\mathbf{q}} = (\mathbf{p}_{COM,i,0}, \boldsymbol{\Theta}_{\text{TD},i}),\\
&\boldsymbol{\alpha}_{j,i}^{\dot{\mathbf{q}}_f,\text{des}} = \dot{\mathbf{q}}_{TO,i} - g\,T_{\text{air}}/2, \quad \boldsymbol{\alpha}_{j,i}^{\mathbf{q}_f} = (\mathbf{p}_{COM,m,0}, \frac{1}{2}(\boldsymbol{\Theta}_{\text{TD},i} + \boldsymbol{\Theta}_{\text{TD},i})), \qquad (2.26)\\
&\forall i = 1, \dots, N_{\text{hops}}+1
\end{aligned}$$

where m is $\frac{1}{2}N_{\text{pts}}$ rounded to the nearest integer. In theory, we could solve a least squares regression problem to find the best fit Bézier curves for our MIP solution, but the impulse assumption makes our approach sufficient.

We formulate the cost function to minimize the squared difference between the Bézier control points of the position trajectories for the MIP solution and the NLP solution:

$$\begin{aligned}\mathcal{L} = &\sum_{i=1}^{N_{\text{hops}}+1}\sum_{j=1}^{d+3}(\boldsymbol{\alpha}^{\mathbf{q}}_{j,i} - \boldsymbol{\alpha}^{\mathbf{q},\text{des}}_{j,i})^T\mathbf{Q}(\boldsymbol{\alpha}^{\mathbf{q}}_{j,i} - \boldsymbol{\alpha}^{\mathbf{q},\text{des}}_{j,i}) \\ &+\sum_{i=1}^{N_{\text{hops}}}\sum_{j=1}^{3}(\boldsymbol{\alpha}^{\mathbf{q}_f}_{j,i} - \boldsymbol{\alpha}^{\mathbf{q}_f,\text{des}}_{j,i})^T\mathbf{R}(\boldsymbol{\alpha}^{\mathbf{q}_f}_{j,i} - \boldsymbol{\alpha}^{\mathbf{q}_f,\text{des}}_{j,i}),\end{aligned} \tag{2.27}$$

where $\mathbf{Q}$ and $\mathbf{R}$ are weight matrices. One could also consider adding quadratic terms of $\boldsymbol{\alpha}^{\boldsymbol{\mathcal{F}}}$ to the cost function to minimize the control effort, but we chose to solely minimize the position tracking error to prioritize the obstacle avoidance task.

The control points of a Bézier curve and its integral are related by a linear operation, so we relate $\boldsymbol{\alpha}^{\boldsymbol{\mathcal{F}}}$ and $\boldsymbol{\alpha}^{\dot{\mathbf{q}}}$ through the linear constraint [15],

$$\frac{M_1}{h}\boldsymbol{\Phi}\boldsymbol{\alpha}^{\dot{\mathbf{q}}}_i = [\tilde{\mathbf{R}}_i\mathbf{H}^{-1}\boldsymbol{\alpha}^{\boldsymbol{\mathcal{F}}}_i - g, \dot{\mathbf{q}}_{i,0}], \tag{2.28}$$

where $\dot{\mathbf{q}}_{i,0}$ is the initial spatial twist for the stance phase i. The matrix $\Phi \in R^{(M+2)\times(M+2)}$ is defined as [15],

$$\Phi_{i,j} := \begin{cases} -1, & j = i = 1,\ldots,M+1 \\ 1, & j = i+1 = 2,3,\ldots,M+2 \\ \frac{h}{M+1}, & i = M+2, j = 1 \\ 0, & \text{otherwise.}\end{cases} \tag{2.29}$$

We plan the wrench trajectory in the body frame, so we can check for feasibility via the nominal FWP constraint (Equation 2.3) without requiring a rotation. We then rotate the wrench trajectory for stance i into the world frame using the matrix $\tilde{\mathbf{R}}_i$ from our MIP solution. Otherwise, we plan all motion in the world frame. We can add similar constraints to Equation 2.28 to relate $\boldsymbol{\alpha}^{\dot{\mathbf{q}}}$ and $\boldsymbol{\alpha}^{\mathbf{q}}$ as well as $\boldsymbol{\alpha}^{\dot{\mathbf{q}}_f}$ and $\boldsymbol{\alpha}^{\mathbf{q}_f}$.

We constrain the first control point of $\boldsymbol{\alpha}_0^{\mathbf{q}}$ and the last control point of $\boldsymbol{\alpha}_{N\text{hops}}^{\mathbf{q}}$ to the start and end positions of the MIP trajectory, and we constrain the first control point of each $\boldsymbol{\alpha}_i^{\boldsymbol{\mathcal{F}}}$ to mg and the last to zero. This makes the wrench trajectory more smooth and feasible for the robot [14]. We leave all other control points as optimization variables. During stance, we constrain the position control points to a bounding box around the MIP solution

$$\begin{aligned} \tilde{\mathbf{R}}_i(\boldsymbol{\alpha}_i^{\mathbf{q}} - \boldsymbol{\alpha}_{i,\text{des}}^{\mathbf{q}}) &\leq \Delta q_{\max} \\ -\tilde{\mathbf{R}}_i(\boldsymbol{\alpha}_i^{\mathbf{q}} - \boldsymbol{\alpha}_{i,\text{des}}^{\mathbf{q}_i}) &\leq \Delta q_{\max}, \quad \forall i = 1, \ldots, N_{\text{hops}} + 1 \end{aligned} \tag{2.30}$$

to remain near the nominal configuration where the FWP constraint is valid and to avoid undesirable configurations where the end-effectors cannot reach the contact positions.

As presented, our smoothing NLP is a quadratic program (QP), but we could use more sophisticated whole-body solvers as a stand-in for this step. We found this formulation to be sufficient for smoothing out the MIP solution to a dynamically feasible trajectory for our tracking controller. Additionally, Gurobi typically solves this QP well within 500 ms, affirming that we could plausibly deploy our approach as an online global path planning framework.

2.7 Results

In this section, we present an example of our approach planning a hopping trajectory in an environment with an angled platform and an obstacle (Section 2.7.1), a performance comparison of various MIP formulations (Section 2.7.2), and preliminary simulation results with the MIT Mini Cheetah (Section 2.7.3). All computation was done on a laptop computer with an Intel i7 processor clocked at 2.60 GHz.

2.7.1 Example Environment with 3D Rotations

This section validates our MIP dynamic motion planner by testing it in an environment with an angled platform and an obstacle. The obstacle prevents the robot from jumping directly to the goal, so it must use the angled platform to maneuver around it. The platform to the side of the obstacle is angled at 11.5 degrees about the x-axis and its center is positioned 0.2 meters above the ground.

Gurobi found a feasible solution to the MIP for this environment in 40 milliseconds. The robot hops to its right while rolling to the left to land upright on the angled platform. It then jumps to the left off of the angled platform and onto the goal platform. It also yaws to the right on its first jump and then to the left on its second jump to fit all four of its contacts on the square platforms. The MIP solution was polished with the QP formulated in Section 2.6. Gurobi solved the QP in 50 milliseconds. A comparison of COM and orientation trajectories for the MIP and QP solutions is presented in Figure 2.5. The QP removes the discontinuities in the velocity of the COM, resulting in a smooth position trajectory for the robot. Figure 2.6 compares the wrench trajectories of the MIP and NLP. The NLP finds a smooth net wrench trajectory near the approximation provided by the MIP.

2.7.2 MIP Formulation Comparisons

In this section, we benchmark the performance of our MIP formulation with an impulsive stance phase against a formulation with a full stance phase. We also compare the performance of our MIP formulation with and without the hard reachability constraint (Section 2.5.1), the forward progress constraint (Section 2.5.2), and the A* inspired terms in the cost function (Section 2.5.3). We perform our testing in

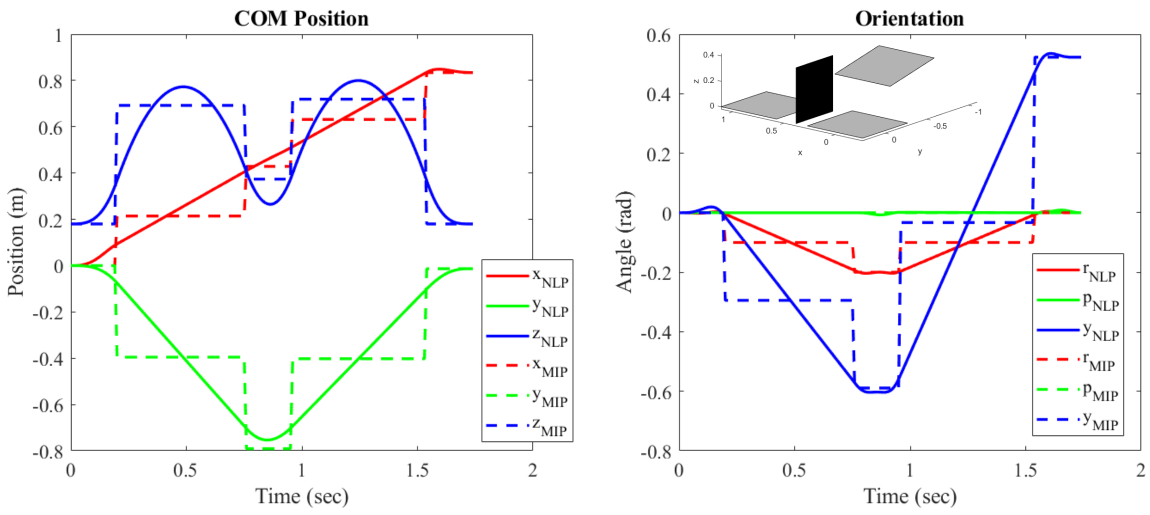

Figure 2.5. Plots of the COM and the orientation trajectories of the robot from the MIP and QP solutions to the path planning problem through the environment shown in the figure. The MIP solution (dotted lines) has squared edges due to the impulse assumption. The QP finds a smooth solution (solid lines) near the MIP solution.

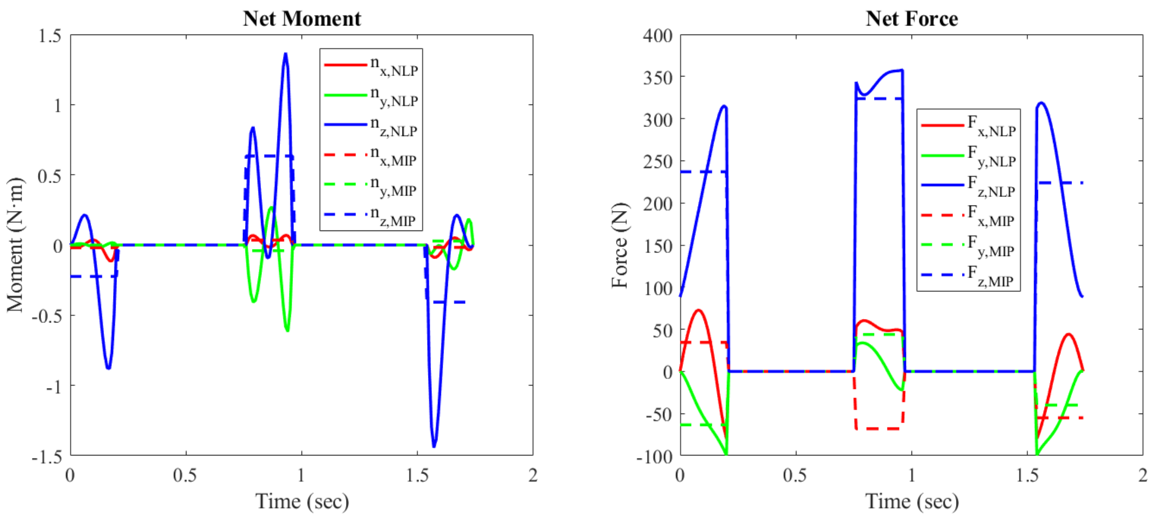

Figure 2.6. The net moment and net force trajectories corresponding to the position trajectories presented in Figure 2.5. The NLP solution is a smooth polynomial while the MIP solution has discontinuities due to the impulse assumption.

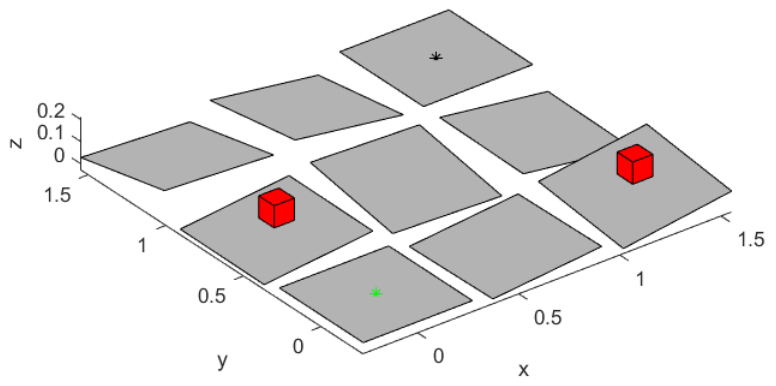

Figure 2.7. An example of a random environment for our experimental comparison. The red cubes are obstacles the robot must avoid. Across tests, we fix the x and y positions of the platforms as well as the start and goal platforms, but we randomly vary their height, orientations, and the positions of the obstacles.

randomly generated environments with 9 platforms and 2 obstacles. We fix the x and y positions of the platforms to a grid, but we randomly choose each z position from a uniform random distribution from 0 to 0.1 meters and the roll and pitch orientation of each platform from a uniform distribution from 0 to 0.2 radians (0 to 11.5 degrees). We also randomly choose two platforms where we place obstacles. An example of a random test environment is shown in Figure 2.7, where the green star marks the start platform and the black star marks the goal platform. The start and goal platforms are constant across all tests. For each formulation, we solve 100 path planning problems, limiting the max solve time to 10 seconds.

2.7.2.1 Full Stance Phase Formulation

When designing the MIP formulation with a full stance phase, we started with our proposed formulation and changed as little as possible to make a fair comparison. We removed the variable $\boldsymbol{\mathcal{F}}$ and replaced it with Bézier control points for the spatial wrench, spatial twist, and position/orientation trajectories for the robot. We removed the constraints for the take-off to take-off dynamics (Equation 2.16) and replaced them with constraints that establish the linear relationships between the Bézier control points as in Equation 2.28. We constrain the control points for the wrench to be within the nominal FWP, and we replaced the MIP box constraint with that of the NLP, which constrains the position and orientation control points to a bounding box (Equation 2.30). Otherwise, the full stance formulation is identical to the impulsive stance formulation.

2.7.2.2 Experimental Comparison

For each formulation, we solved 100 path planning problems in randomly generated environments, as explained in Section 2.7.2. We limited the max solve time to 10 seconds and present the average solve time (mean), the maximum solve time (max), the percentage of problems where the solver failed to find a solution (fail), and the percentage of problems where the solver found the optimal solution (optimal). We compared our impulsive stance (IS) formulation with the full stance (FS) formulation. We also compared formulations with and without a cost function (cost), with and without the hard reachability constraint (reachability), with and without the forward progress constraint (forward), and with and without the extra A* inspired cost function terms (A* terms). In formulations with a cost function, we chose to use a cost function

$$\mathbf{J}(x_{\text{opt}}) = \sum_{i=1}^{N_{\text{hops}}+1} i\,\boldsymbol{\mathcal{F}}_{6,i} \quad \text{or} \quad \sum_{i=1}^{N_{\text{hops}}+1} \sum_{j=1}^{d+1} i\,\boldsymbol{\alpha}^{\boldsymbol{\mathcal{F}}}_{j,6,i} \tag{2.31}$$

that penalizes the net force in the z-direction for each stance phase. We use the cost function to the left for the impulsive stance formulation, and the one to the right for the full stance formulation. By including a factor i in each term we penalize taking hops later in the trajectory, biasing the solver toward solutions that hop to the goal in a small number of hops. We set $N_{\text{hops}} = 4$, but the robot typically reached the goal within 3 hops.

The performance of the various MIP formulations considered in this work is presented in Table 2.1. The solver struggled to find a solution to problems in the full stance formulation within the 10 second time limit, while it never failed to find a solution to problems in the impulsive stance formulation, validating our claim that the impulsive stance formulation allows us to add constraints that consider 3D orientation dynamics, contact choice, and obstacle avoidance without introducing a significant computational slow down.

The hard reachability constraints and the forward progress constraints had varying successes. When adding the reachability constraint to the feasibility problem, we saw a slight reduction in the maximum solve time from 1.284 seconds to 1.07 seconds, and when adding the forward reachability constraint to the formulation with a cost function, Gurobi solved 96% of the problems to global optimality, whereas without the constraint it did not solve any of the problems optimally. However, there is an inherent tradeoff with adding these constraints because it adds more binary constraints to the optimization. Future work could consider methods to eliminate nonphysical combinations of binary variables before the optimization without introducing such constraints. Adding the A* terms to the cost function seemed to significantly reduce the average solve time, with and without the cost function in Equation 2.31.

2.7.3 Preliminary Simulation Results

This section presents preliminary simulation results where we track optimal hopping trajectories generated by our framework with the MIT Mini Cheetah using the nonlinear MPC strategy presented in [29]. We chose to use an MPC framework for our tracking controller to benefit from on-the-fly replanning in case the robot departs from the desired reference trajectory. We found that purely reactive control strategies

Formulation	**Mean (s)**	**Max (s)**	**Fail (%)**	**Optimal (%)**
IS w/o Cost	0.345	1.284	0	N/A
FS w/o Cost	N/A	N/A	100	N/A
IS w/ Cost	10.015	10.055	0	0
FS w/ Cost	9.9364	10.063	98	2
IS w/o Cost + Reachability	0.347	1.07	0	N/A
IS w/ Cost + Reachability	10.013	10.048	0	0
IS w/o Cost + Forward	2.876	10.016	0	N/A
IS w/ Cost + Forward	5.2487	10.012	0	96
IS w/o Cost + A* Terms	0.187	0.819	0	N/A
IS w/ Cost + A* Terms	3.927	10.017	0	95
FS w/o Cost + A* Terms	10.0179	10.0592	96	N/A
FS w/ Cost + A* Terms	N/A	N/A	100	0

TABLE 2.1

BENCHMARK TEST RESULTS FOR VARIOUS MIP FORMULATIONS can struggle with longer hops, often failing to maintain the small rotational velocities needed at takeoff to set up a stable landing without excessive offline gain tuning.

The MPC strategy we selected uses a variant of a constrained Differential Dynamic Programming (DDP) solver [28], which provides both a feedforward control tape for its replanned trajectory and a feedback control gain to regulate the body coordinates, $\boldsymbol{x}_{\mathrm{B}} = (\boldsymbol{\Theta}, \mathbf{p}, \boldsymbol{\omega}, \boldsymbol{v})$, to the desired trajectory as closely as possible. We provide the controller with the optimized footstep positions and contact status from our MIP solution as well as the position and orientation trajectories from our NLP. Additionally, we perform inverse kinematics to provide the controller with reference joint position angles. The controller uses the DDP solver to repeatedly solve a TO problem with the running cost function

$$l = \int_{t_0}^{t_f} \left(\|\delta\boldsymbol{\Theta}^T, \delta\mathbf{p}^T, \delta\boldsymbol{\omega}^T, \delta\boldsymbol{v}^T\|^2_{\mathbf{Q}_b} + \|\hat{\boldsymbol{S}}\delta\mathsf{q}\|^2_{\mathbf{Q}_f} + \|\boldsymbol{S}\delta\mathbf{p}_f\|^2_{\mathbf{Q}_J} + \|\boldsymbol{S}\delta\boldsymbol{\lambda}\|^2_{\boldsymbol{R}_\lambda} \right) dt \quad (2.32)$$

where δ represents the deviation from the desired trajectory, $\mathbf{Q}_b$, $\mathbf{Q}_J$, $\mathbf{Q}_f$, $\boldsymbol{R}_\lambda$ are weight matrices, $\boldsymbol{S}$ is a diagonal matrix of the contact status of each foot, $\hat{\boldsymbol{S}}$ is a diagonal matrix of the swing status of each foot, $\mathbf{p}_f$ is the vector of foot positions, and $\boldsymbol{\lambda}$ is the vector of GRFs. The notation $\|x\|^2_{\mathbf{Q}}$ is shorthand for $x^T\mathbf{Q}x$. The TO problem also includes friction constraints and a hybrid kinodynamic model that extends the SRB model to consider foot positions during stance and joint angles during swing. The MPC controller outputs a new plan that minimizes the tracking error with respect to our desired trajectory from the MIP and NLP solutions. It provides a desired body trajectory $\boldsymbol{x}_{\mathrm{B,des}}$, desired foot positions $\mathbf{p}_{\mathrm{f,des}}$, foot velocities $\dot{\mathbf{p}}_{\mathrm{f},des}$, desired GRFs $\boldsymbol{\lambda}^*$, and feedback control gains $\boldsymbol{K}^*$. For the stance leg controller, we used proportional-derivative (PD) tracking for the Cartesian foot positions as well as the feedforward GRFs and feedback control law from the MPC output. We

Gain	**Stance**	**Swing**	**Early Contact**	**Units**
$\boldsymbol{K}_p$	0	700	1000	$\mathrm{N} \cdot \mathrm{m}^{-1}$
$\boldsymbol{K}_d$	0	7	100	$\mathrm{N} \cdot \mathrm{s} \cdot \mathrm{m}^{-1}$

TABLE 2.2

PD GAINS FOR CARTESIAN FOOT TRACKING

implemented the control law,

$$\boldsymbol{\tau} = \boldsymbol{J}^T(\boldsymbol{K}_p(\mathrm{p}_{\mathrm{f,des}} - \mathrm{p}_{\mathrm{f}}) + \boldsymbol{K}_d(\dot{\mathrm{p}}_{\mathrm{f,des}} - \dot{\mathrm{p}}_{\mathrm{f}}) + \boldsymbol{\lambda}^* + \boldsymbol{K}^*(\boldsymbol{x}_{\mathrm{B,des}} - \boldsymbol{x}_{\mathrm{B}})), \tag{2.33}$$

where $\boldsymbol{\tau}$ is the vector of joint torques and $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ are the PD control gains. During swing, $\boldsymbol{\lambda}^*$ and $\boldsymbol{K}^*$ are set to zero, and we only use the Cartesian foot PD terms to track the foot swing trajectories, which we parametrize as Bézier curves after each iteration of the MPC.

We implemented a contact estimator by checking if any of the knee joint velocities changed past a certain threshold in between iterations of the leg controller. The leg controller runs at around 700 Hz, and we set the threshold at 4 meters per second. At an early contact, we increase the Cartesian foot PD terms to dampen the impact of the landing. We provide the PD gains for each phase of the trajectory in Table 2.2.

We used the MPC framework and stance controller to track a trajectory of the MIT Mini Cheetah taking two consecutive jumps. We planned the trajectory offline using our proposed planning framework. We set the duration of stance as 0.2 seconds. Figure 2.8 shows images from the simulation captured at various points of the trajectory, and Figure 2.9 is a plot of the position and orientation tracking. The robot

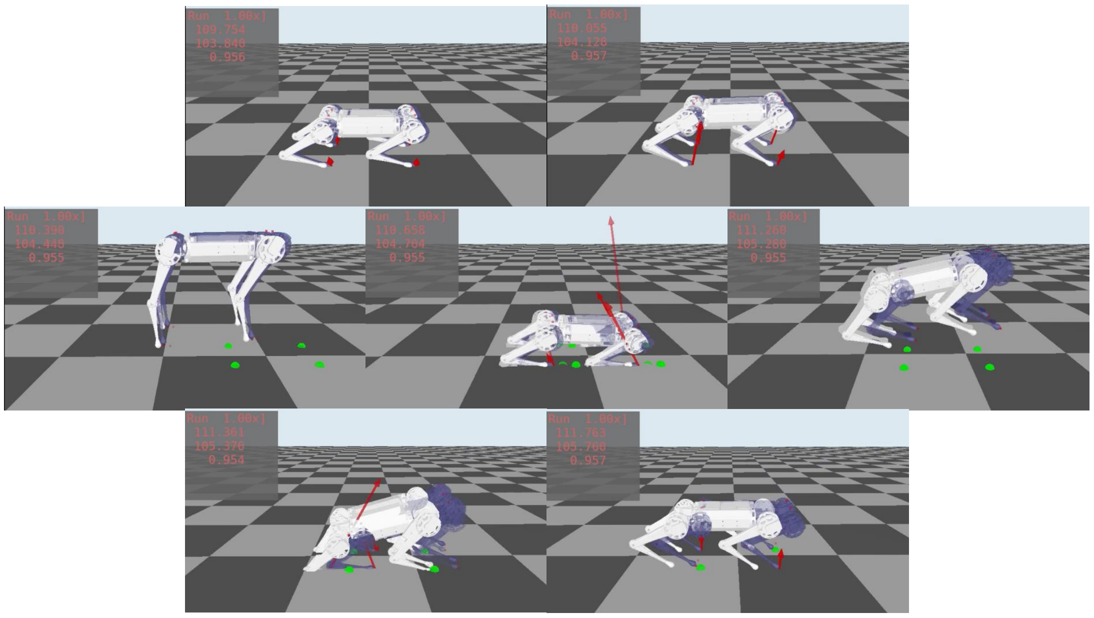

Figure 2.8. Images captured from the simulation as the MIT Mini Cheetah tracked the double jump trajectory planned with our proposed framework. During the first flight phase, the robot maintains near zero Euler angles, and it makes a clean first landing. On the second hop, the robot begins to pitch backward, causing the hind knees to hit the ground on its second landing. The robot quickly recovers and sticks the landing.

landed both jumps cleanly, although the hind knees hit the ground on the second landing due to the pitch tracking error. The controller slightly under tracked the x and z position trajectories, but we suspect that we could improve the tracking by fine tuning the MPC's cost function and the parameters of the DDP solver, both of which we left untouched.

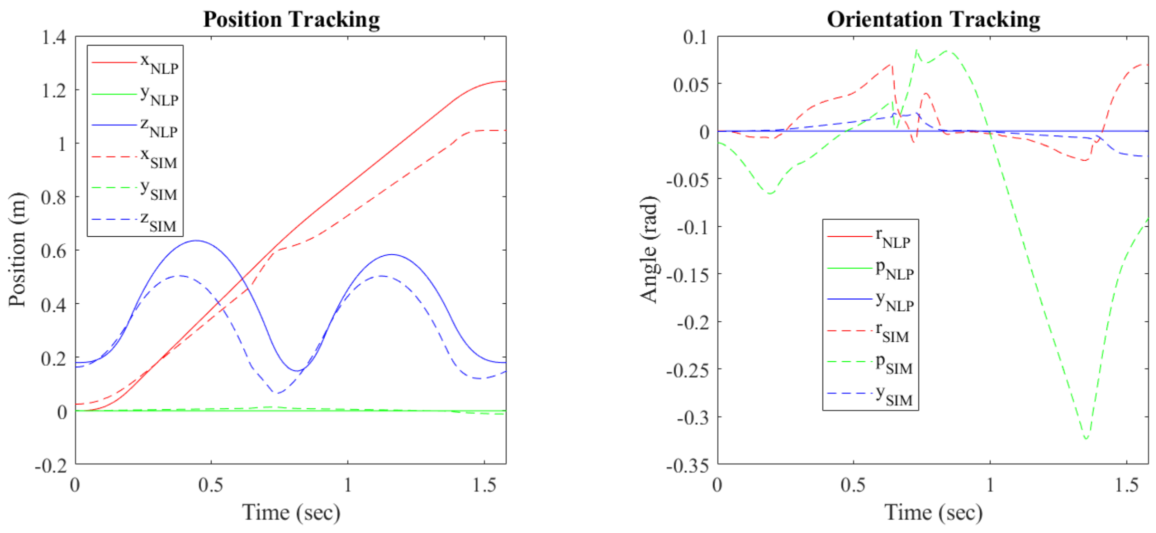

Figure 2.9. The position and orientation tracking from the MIT Mini Cheetah simulation as it tracks the double jump trajectory planned with our proposed framework. The controller under tracked the x and z position trajectories, and the pitch tracking error diverges during the second jump. We suspect that we can reduce the tracking error by tuning the MPC framework to optimize its performance for hopping motions.

CHAPTER 3

MACHINE LEARNING TECHNIQUES FOR ESTIMATING THE STEADY-STATE RESPONSE

3.1 Introduction

In this chapter, we focus our efforts toward comparing techniques that estimate the steady-state output of a legged robot when driven by an external disturbance. We propose a new method that relies on a transformer neural network [52] trained offline to predict the output of the system. Popular within the field of natural language processing, transformers excel at processing sequence-to-sequence data. We pass a sequence of the robot's past outputs and environment information as input to the network and set the robot's current output as the target. We compare the performance of the transformer with another popular network architecture for sequence-to-sequence data: the long short-term memory (LSTM) neural network. We also demonstrate finetuning the neural network by removing its final layer and replacing it with adaptable linear coefficients, offering a pathway toward recursive online updates to our model. Additionally, we compare our approach to a moment-matching reduced order model (MM-abstraction) [3], with the key difference being that MM-abstractions assume we have a dynamics model for the disturbance generator and that the disturbances enter the robot's dynamics in a known, structured manner that can be described through physics-based models. However, it is difficult, if not impossible, to anticipate all the ways that disturbances will enter a system deployed in uncertain environments. Neural networks address this issue by learning a set of features common

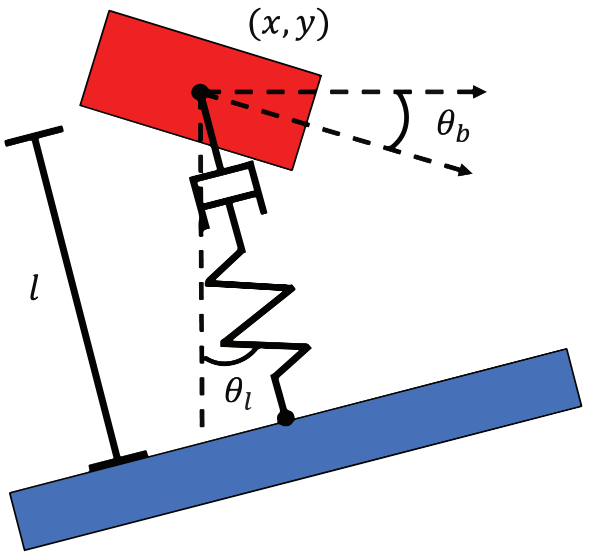

Figure 3.1. A model of the 2D Raibert Hopper. The state of the hopper is completely described by the center of mass position (x, y), the body angle θ_b, the leg angle θ_l, and the leg length l.

across all environments seen during their offline training period. These features can subsequently be used as nonlinear basis functions for an online regression to estimate the steady-state output of the system.

We validate our approach by training a transformer to estimate the steady-state output of Raibert's hopper [47] as it traverses moving terrain, demonstrating its superior robustness to changes in the environment relative to moment-matching models. The training environment is comparable to the conditions on the pitching deck of a naval ship, which has a time-varying slope and velocity. The remainder of this section reviews related work in the field of legged locomotion control relevant to the framework we present in this chapter and Chapter 4. We then provide some background on MM-abstractions, transformers, and LSTMs (Section 3.2). We applied these methods to estimate the steady-state output of Raibert's hopper (Section 3.3) and compared their performance (Section 3.4).

3.1.1 Related Work

State-of-the-art legged locomotion control methods often use physics-based models for the robot and choose an optimization-based control approach that offers some desirable performance guarantees. However, these control guarantees rely on the assumption the models are correct. This often fails to hold true because physics-based models struggle to capture some details about the environment. Our work offers an avenue toward addressing the issue of model uncertainty by detailing a method that isolates the robot's response to unmodeled dynamics. Once this response is known, we can apply a feedback control law to drive the system back to its stable, steady-state behavior.

More recently, some of the legged locomotion community has shifted their focus toward offline policy optimization through deep reinforcement learning (RL). Past work has demonstrated the use of domain randomization to achieve impressive robustness in a diverse range of environments [26, 36]. Still, the resulting policy is often implemented in an open-loop fashion, leading to the inability to stabilize the robot in response to dynamics neglected during training. Further, the resulting policy lacks performance guarantees, and the end-to-end training process can be extremely data inefficient. Our approach addresses the efficiency issue by (1) considering the combined system of the robot and a base controller, thereby avoiding the need to learn a control policy from scratch, and (2) simplifying the learning problem to the identification of the steady-state behavior and natural dynamics of the system. Additionally, our approach allows for improving performance through online updates.

3.2 Background

This section offers brief overviews of three different methods for capturing the steady-state behavior of a system: MM-abstractions (Section 3.2.1), transformers 3.2.2, and LSTM models (Section 3.2.3). For a more in-depth description of MM-abstractions, we recommend [3] or Section III of [27]. For more background on transformers, we recommend [52] and Chapter 11 of [8], while for LSTMs, we recommend [21] and Chapter 10 of [8].

3.2.1 MM-Abstractions

When provided a model of how disturbances enter a system, one can set up an MM abstraction to estimate its steady-state behavior. To illustrate this concept, we consider a dynamical system called the plant with state equations

$$\dot{\boldsymbol{x}}\boldsymbol{(t)} = \boldsymbol{F}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{\nu}(t)), \qquad \boldsymbol{y}(t) = \boldsymbol{G}(\boldsymbol{x}(t)), \tag{3.1}$$

where $\boldsymbol{F} : R^n \times R^c \times R^d \to R^n$ is Lipschitz, $\boldsymbol{G} : R^n \to R^o$ is continuous, $\boldsymbol{x} : R^+ \to R^n$ is the plant's state, $\boldsymbol{u} : R^+ \to R^c$ is the plant's control input, $\boldsymbol{\nu} : R^+ \to R^d$ is the disturbance from the environment, and $\boldsymbol{y} : R^+ \to R^o$ is the plant's output. In this case, we assume that the disturbance $\boldsymbol{\nu}$ is generated by a linear time-invariant system called the generator with state equations

$$\dot{\boldsymbol{\omega}}(t) = \boldsymbol{S}\boldsymbol{\omega}(t), \qquad \boldsymbol{\nu}(t) = \boldsymbol{L}\boldsymbol{\omega}(t), \tag{3.2}$$

where $\boldsymbol{\omega}(t) : R^+ \to R^g$ is the generator's state, $\boldsymbol{S} \in R^{g\times g}$ is the generator's state matrix, and $\boldsymbol{L} \in R^{d\times g}$ is its output matrix. When the generator's dynamics take the form in Equation 3.2, the moment-matching abstraction takes the following form [3]

$$\dot{\boldsymbol{\xi}}(t) = \boldsymbol{S}\boldsymbol{\xi}(t) + \boldsymbol{\Delta}(\boldsymbol{\nu}(t) - \boldsymbol{L}\boldsymbol{\xi}(t)), \tag{3.3a}$$

$$\boldsymbol{\psi}(t) = [\boldsymbol{G} \circ \boldsymbol{\Pi}](\boldsymbol{\xi}(t)), \tag{3.3b}$$

where $\boldsymbol{\xi} : R^+ \to R^g$ is the abstraction's state, $\boldsymbol{\Delta} \in R^{g\times d}$ is a gain matrix such that $\boldsymbol{S} - \boldsymbol{\Delta}\boldsymbol{L}$ is Hurwitz, $\boldsymbol{\psi} : R^+ \to R^o$ is the MM-abstraction's output, and $\boldsymbol{G} \circ \boldsymbol{\Pi}$ is a map from $\boldsymbol{\xi}$ to the plant's output $\boldsymbol{y}$. When $\boldsymbol{\Pi}$ satisfies the partial differential equation (PDE)

$$\boldsymbol{F}(\boldsymbol{\Pi}(\boldsymbol{\xi}), \boldsymbol{L}(\xi)) = \frac{\partial \Pi(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\boldsymbol{S}\boldsymbol{\xi}, \tag{3.4}$$

we can guarantee that the abstraction's steady-state response is equal to the plant's steady-state response. Forming this PDE requires a perfect dynamics model, $\boldsymbol{F}$, of the plant, so it is impractical to solve for $\boldsymbol{\Pi}$ analytically. However, the map $\boldsymbol{G} \circ \boldsymbol{\Pi}$ only depends on the abstraction state $\boldsymbol{\psi}$, which we can calculate using Equation 3.2, and its output should be equal to the plant's output $\boldsymbol{y}$, which we can observe. Therefore, we can avoid solving Equation 3.4 and instead learn $\boldsymbol{G}\circ\boldsymbol{\Pi}$ using regression techniques.

3.2.2 Transformers

The transformer architecture revolutionized natural language processing, outperforming recurrent neural network architectures (such as the LSTM), and serving as the foundation for large language models [6], such as ChatGPT. Each data point in the input sequence to a transformer is often referred to as a token. The key component to the transformer's success is known as the self-attention mechanism, which attunes the representation of each token to that of related tokens within its sequence. At a higher level, it allows tokens within a sequence to communicate contextual information with each other. Transformers typically have multiple heads of self-attention, each of which learns different representations of the input.

As in [52], a self-attention head has three embedding layers that learn linear transformations of the input denoted as the key, query, and value of the token. The query vector can be interpreted as a representation for what information the token is looking for in other tokens in its sequence, while the key vector resembles what kind of information the token contains. The value is a representation for the actual information the token contains. Often, the key and value can be the same vector. The output of a self-attention head is computed as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q\,K^T}{\sqrt{d_k}}\right)V, \tag{3.5}$$

where Q, K, and V are matrices of the queries, keys, and values for each token in the input sequence and d_k is the dimension of the query and key vectors. The softmax acts a normalization mechanism and proves useful when adding a mask to remove unwanted connections. In our architecture, we use a lower triangular mask to enforce causal relationships from past outputs to the current output. The matrix product $Q\,K^T$ computes the dot products between the queries and values of all the data points. These dot products can be interpreted as affinities—the dot product between the query of one token and the key of another is large when the former is looking for information contained by the latter. The division by $\sqrt{d_k}$ is another normalization technique. Lastly, the multiplication by V performs a weighted average of the values.

The input travels through an embedding layer before it's passed to the self-attentions heads. In natural language processing, the token are integers, and this layer is a word embedding table. Since our tokens are vectors of floats, we replace this table with a linear layer. We also include a position embedding table and sum its output with that of the embedding layer, giving the network context about the position of each token within the sequence. The transformer architecture includes additional normalization layers, dense layers, and feedforward connections.

We implement our transformer neural network in Pytorch [43]. Our network includes 8 self-attention heads, and we add an additional linear decoder layer at the output that can later be replaced by adaptable linear coefficients for online updates. In total, the network had about 150 thousand parameters. We can scale the size of the network as needed to leverage the onboard computational abilities of state-of-the-art legged robots.

3.2.3 LSTM Models

An LSTM is a recurrent neural network (RNN) structure, meaning that it incorporates feedback of past outputs into its current output. It excels at processing time-series data where there is a causal relationship between past outputs and the current output. Relative to LSTMs, simple RNNs struggle to establish long term dependencies in time-series data. Their shortcomings can be attributed to the "vanishing gradient problem": In deep neural networks, the gradient of the loss function can be very small, making it difficult to update the weights of the network in a meaningful way using backpropagation [5]. The LSTM architecture addresses this issue by explicitly saving information from past outputs to prevent it from vanishing over time [21].

Each node of an LSTM network has three inputs: the input x_t, the previous output y_{t-1}, and the carry c_t. The carry term "carries" relevant information from outputs prior to $t-1$. The output of each node takes the form [8]

$$y_t = \boldsymbol{A}[W_{x,o}x_t + W_{y,o}y_{t-1} + W_{c,o}c_t + b_o] \tag{3.6}$$

where y_t is the current output, $\boldsymbol{A}$ is an activation function, b_o is a bias, and $W_{x,o}$, $W_{y,o}$, and $W_{c,o}$ are weight matrices. The carry term c_t is determined by the following equations [8]

$$\begin{aligned} i_t &= A[W_{x,i}x_t + W_{y,i}y_t + b_i], \\ f_t &= A[W_{x,f}x_t + W_{y,f}y_t + b_f], \\ k_t &= A[W_{x,k}x_t + W_{y,k}y_t + b_k], \\ c_{t+1} &= i_t * k_t + c_t * f_t, \end{aligned} \tag{3.7}$$

where $*$ symbolizes element by element multiplication. The term $c_t * f_t$ can be thought of as a way to purposely forget irrelevant information, while the term $i_t * k_t$ adds new information [8].

In our work, we use an implementation of the LSTM architecture provided by the Pytorch [43]. We found that LSTM layers achieved a lower mean average error than simple feed-forward networks and gated recurrent units (GRUs) when predicting the steady-state behavior of the hopper.

3.3 Methods

This section describes how we formulated our MM-abstraction (Section 3.3.1) and how we trained a transformer (Section 3.3.2) to predict the steady-state behavior of the step-length controller for Raibert's hopper traversing moving terrain. It also briefly covers key details about the dynamics in our MATLAB-based simulation environment (Section 3.3.3).

3.3.1 Moment Matching Model Formulation

Our approach for formulating the MM-abstraction for Raibert's hopper follows suit of [27], although we make adjustments to account for the effects of moving terrain. We select the forward velocity of the hopper, $\dot{\boldsymbol{x}}$, as the output to predict with the MM-abstraction because the step-length controller accepts the desired forward velocity as input. In Chapter 4, we will close the loop, adapting this control input to mitigate a legged robot's natural response to a disturbance (see Figure 3.2).

We assume the disturbance is the output of a generator with the state equations given by Equation 3.2, where $\boldsymbol{S}$ and $\boldsymbol{L}$ take the form

$$\boldsymbol{S}_a = \begin{bmatrix} 0 & -\frac{1}{T_a} \\ \frac{1}{T_a} & 0 \end{bmatrix} \quad \boldsymbol{S}_b = \begin{bmatrix} 0 & -\frac{1}{T_b} \\ \frac{1}{T_b} & 0 \end{bmatrix}$$

$$\boldsymbol{S} = blkDiag(\boldsymbol{S}_a, \boldsymbol{S}_a, \boldsymbol{S}_b, \boldsymbol{S}_b)$$

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

With these choices of $\boldsymbol{S}$ and $\boldsymbol{L}$, we assume the generator takes the form of a harmonic oscillator with two fundamental periods $T_a \in R$ and $T_b \in R$. The disturbance output $\nu(t) : R^+ \rightarrow R^2$ is the current slope and velocity of the terrain, which we know will have the same fundamental harmonics but will vary in phase. We set the gain matrix, $\boldsymbol{\Delta}$, for the abstraction state observer (Equation 3.3a) as the gains of a steady-state Kalman filter.

In practice, the robot can estimate the current slope and velocity of the terrain using a perception system. If it collects this data at a sufficiently large sampling rate, we know by the Nyquist Theorem that we can perform a Fourier analysis to

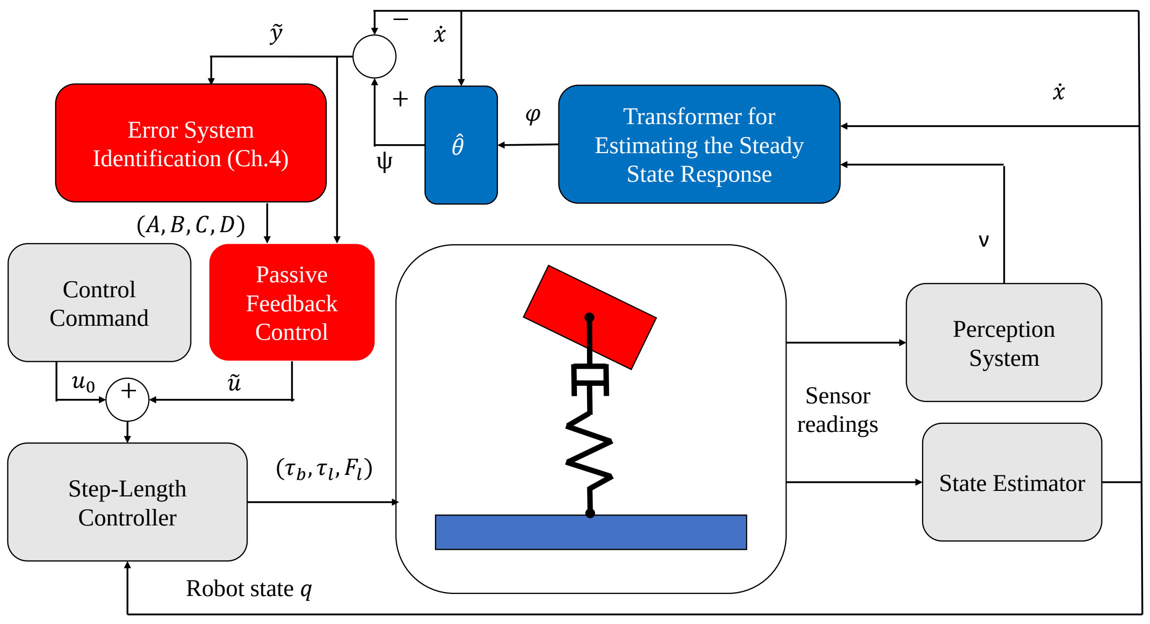

Figure 3.2. System overview. The blue blocks are the main components introduced in this chapter, while the red block are introduced in Chapter 4.

determine the terrain's fundamental harmonics. If we find the terrain exhibits more than two dominant fundamental harmonics, we can easily extend the abstraction state $\boldsymbol{\xi}$ and adjust the state matrices accordingly.

To learn the lifting map $\boldsymbol{G} \circ \boldsymbol{\Pi}$ from the abstraction state $\boldsymbol{\xi}$ to the output $\boldsymbol{\psi}$, we perform a standard least squares regression to estimate the parameters $\hat{\boldsymbol{\theta}}_{mm} \in R^{g+1}$ so that the prediction of the steady-state behavior, $\hat{\boldsymbol{\psi}}(t_i)$, takes the form

$$\hat{\boldsymbol{\psi}}(t_i) = \hat{\boldsymbol{\theta}}_{mm}^T(t_{i-1}) \begin{bmatrix} \boldsymbol{\xi}(t_i) \\ 1 \end{bmatrix}, \quad i = 1, ..., N. \tag{3.8}$$

The parameters $\boldsymbol{\theta}^*_{mm}$ minimize the squared estimation error between $\hat{\boldsymbol{\psi}}$ and the robot's true output, y. Here, the abstraction state, $\boldsymbol{\xi}$, and a bias term serve as the basis functions for the regression, while $\hat{\boldsymbol{\theta}}_{mm}$ is a set of adaptable linear coefficients. In practice, we perform the regression on about 40 seconds of training data. We step the abstraction state observer once in each phase of the gait cycle and record the output of the hopper at each takeoff.

3.3.2 Transformer Training

Transformers accept sequence data as input and learn a mapping between past outputs and the current output. We chose to provide the Transformer with the same inputs as the MM-abstraction: the time between samples h, the disturbance $\boldsymbol{\nu}(t)$ and the plant's output $\dot{\boldsymbol{x}}(t)$. We pass these inputs through a delay embedding block (Figure 3.2), which outputs the sequence $\begin{bmatrix} x(t_{i-L}) & x(t_{i-L+1}) & \dots & x(t_{i-1}) \end{bmatrix}$, where $x(t_i)$ is the stacked vector of inputs at time t_i and L is the length of the sequence. In our experiments, we set L equal to 10 and train the transformers to predict the next element i in the output sequence, $x(t_i)$, which includes the plant's current output $\dot{\boldsymbol{x}}(t_i)$. We use a loss function that minimizes the mean-squared error between the output and its target. We set our optimizer as the Adam algorithm with its default learning rate of 0.001.

During the training period, the final layer of the network is a linear layer of weights and biases. For deployment, we demonstrate in Section 3.4 removing this layer and using the new outputs (in the embedding space) as basis functions for a regression to finetune the network to the current environment. As in the previous section, we find the optimal parameters $\boldsymbol{\theta}^*$ through a standard least squares regression on 40 seconds of training data. In practice, finetuning the network may be unnecessary as the estimation error of the offline-trained network is sufficiently small. However, it may prove useful when deploying the network in settings that slightly vary from the training environment.

We collect training data for the transformer with a MATLAB-based simulation of Raibert's hopper equipped with the step-length controller. The simulation is set up such that the slope and velocity of the terrain varies sinusoidally over time. On each run of the simulation, we randomly choose two fundamental periods and an amplitude for the terrain from uniform distributions. For data collection, we record data from 1000 episodes, and then we randomly distribute the data into 800 episodes for the training set and 200 for the testing set. When computing batch gradients for training, we mix rollouts from different episodes to encourage learning a common representation for all the training environments. We set the batch size as 16 and the maximum number of gradient descent steps as 5000. We found that our approach offered training, validation, and testing data sets that were sufficiently uncorrelated by observing the network's tendency to overfit to its training data in less than 1000 gradient descent steps.

Each episode of the simulation during data collection terminates if it meets one of the following conditions: (1) part of the hopper's body touches the terrain (fall) or (2) the total simulation time surpasses 40 seconds. If the hopper falls, we drop the last few gait cycles from the recorded data before adding it to the dataset.

3.3.3 Contact and Stance Dynamics on Moving Terrain

In our MATLAB-based simulation of Raibert's hopper (Figure 3.3), the terrain varies sinusoidally over time. We designed the simulation so that the terrain continues to accelerate during a stance phase. Thus, the hopper's foot also has a nonzero acceleration during stance. Additionally, the hopper's foot has a nonzero velocity at the instant it makes contact with the terrain. We show how these two observations our reflected in the hopper's equation of motion during stance and our hard contact model.

The equation of motion of the hopper is

$$\boldsymbol{A}\ddot{\boldsymbol{q}} + \boldsymbol{C}\dot{\boldsymbol{q}} + \tau_g = \tau + \boldsymbol{J}^T \boldsymbol{f} \tag{3.9}$$

where $\boldsymbol{A}$ is its mass-inertia matrix, $\boldsymbol{C}$ is a Coriolis matrix, τ_g is the generalized force due to gravity, τ is the actuator torques, $\boldsymbol{J}$ is the contact Jacobian, $\boldsymbol{f}$ is the contact force, and $\boldsymbol{q} = (x, y, \theta_b, \theta_l, l)$ is the state of the hopper (see Figure 3.1). During a stance phase, the acceleration of the foot is equal to the acceleration of the terrain. Thus, we add an additional equation to the system

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{J}^T \\ \boldsymbol{J} & 0 \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{f} \end{bmatrix} = \begin{bmatrix} \tau - \boldsymbol{C}\dot{\boldsymbol{q}} - \tau_g \\ -\dot{\boldsymbol{J}}\dot{\boldsymbol{q}} + \boldsymbol{a}_g \end{bmatrix}, \tag{3.10}$$

where $\boldsymbol{a}_g$ is the acceleration of the terrain. Notice that in the bottom row, the time derivative of the foot velocity, $\frac{d}{dt}(\boldsymbol{J}\dot{\boldsymbol{q}}) = \boldsymbol{J}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}$, is equal to the acceleration of the terrain.

We use a hard contact model in the simulator. At the instant of impact, the hopper's acceleration, $\ddot{\boldsymbol{q}}$, is equal to the difference between the hopper's velocity just after and before the impact $(\dot{\boldsymbol{q}}^+ - \dot{\boldsymbol{q}}^-)$. Thus, the dynamics are

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{J}^T \\ \boldsymbol{J} & 0 \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{f} \end{bmatrix} = \begin{bmatrix} 0 \\ -\boldsymbol{v}_{f/g} \end{bmatrix}, \tag{3.11}$$

where $\boldsymbol{v}_{f/g}$ is the velocity of the foot relative to the ground. With Equation 3.11, the velocity of the foot just after impact, $\boldsymbol{J}\dot{\boldsymbol{q}}^+$, will be equal to the velocity of the terrain.

3.4 Results

This section compares the performance of the proposed methods for estimating the steady-state response of the hopper over moving terrain. We compare three cases: (1) MM-abstraction from Section 3.3.1, (2) the transformer 3.3.2, and (3) an LSTM. We train the LSTM on the same training data as the Transformer. We used 5

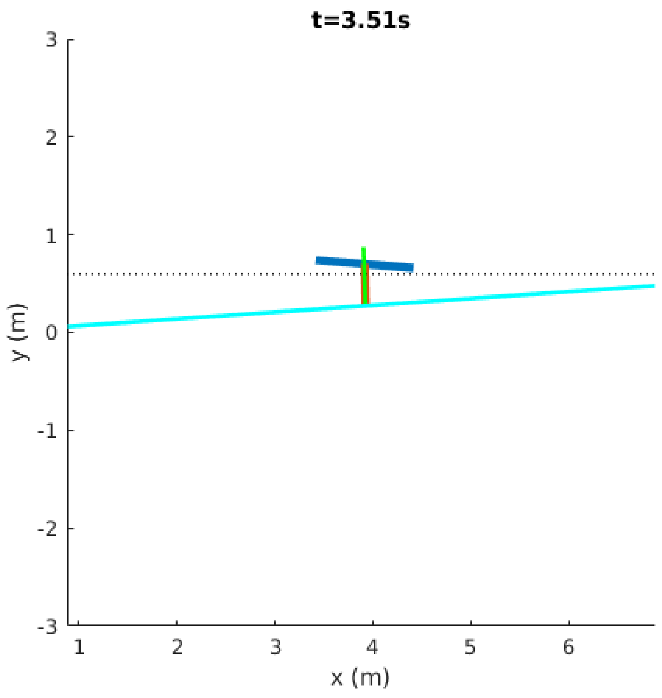

Figure 3.3. A snapshot of the animation of our Matlab-based simulation of Raibert's hopper. The blue line marks the terrain height. Within our simulation, the terrain has a nonzero velocity, similar to a ship deck rocking at sea. The green line along the hopper's leg is the GRF.

stacked LSTM layers and a total network size about equal to that of the transformer ($\sim$150 thousand parameters). Our data collection period takes approximately 20-30 minutes on a laptop computer with an Intel i7 processor, 16 GB of RAM, and an Nvidia GTX 1660 Ti graphics card (GPU). It takes approximately 2 minutes to train the transformer network and 5 minutes to train the LSTM. The small difference in training time is attributed to the transformer architecture being more parallelizable relative to the LSTM, allowing Pytorch to better leverage computation on the GPU. The transformer, unlike the LSTM, has no recurrent connections. At inference time, the LSTM takes 800 microseconds on average, while the Transformer took about 700 microseconds. The difference in training time and inference time seemed negligible for our use case, but the transformer may be noticeably faster with longer sequences of input data and a higher dimension for each token.

After formulating our MM abstraction and training the transformer and LSTM, we evaluated each approach's mean absolute error (MAE) on the test dataset. In Table 3.1, we provide the test MAEs from one dataset of 200 simulation episodes, each having terrain with two harmonics.

For the method MM-Fit, we fit a new MM abstraction to each set of test data, including the true fundamental harmonics in $\boldsymbol{S}$ for each episode. In MM-Test, we fit a single MM abstraction to a random episode, with the true fundamental harmonics for that episode, and then tested its performance across all of the test data. The order of magnitude difference in MAE between these two cases shows that the MM-abstraction may track the steady-state output less closely when the parameters of the environment change online. It's important to note, however, that the MM-abstraction error remains bounded, likely due to the feedback within the disturbance generator's observer. The transformer and LSTM had a lower MAE across all the test environments, demonstrating an aptitude for robustness when parameters of the environment change online. This robustness can be attributed to our training scheme

Method	MM-Fit	MM-Test	Transformer	LSTM
MAE (m/s)	0.0072	0.0425	0.0016	0.0022

TABLE 3.1

MAE FOR ESTIMATING THE HOPPER'S FORWARD VELOCITY.

that leverages domain randomization to learn an effective model across a diverse set of environments. The MAE for the neural networks was on the same order as MM-Fit, but next, we show that we can further reduce the estimation error by finetuning the transformer to a specific environment.

3.4.1 Finetuning

We also compared the outputs of the offline-trained transformer and the finetuned transformer on specific environments. In Figure 3.4, we show a snapshot of the outputs of both networks (left) as well as a plot of the estimation error (right). The desired forward velocity of the hopper was 1 meter per second, but the moving terrain caused the hopper's velocity to diverge and oscillate from its reference. However, the transformer predicts the change in velocity due to the terrain, estimating the hopper's steady-state response to the disturbance. There is a small steady-state estimation error for the baseline transformer, but we demonstrate eliminating this error by finetuning the network with the input-output data for this specific environment. Finetuning decreases the mean average estimation error from 0.0044 to 5.6×10^{-5} meter per second. Note, however, that this is an aggressive finetuning framework because the model forgets the information embedded within its final layer and replaces it with information relevant to the current environment.

A potential less aggressive alternative to our finetuning strategy is to freeze all

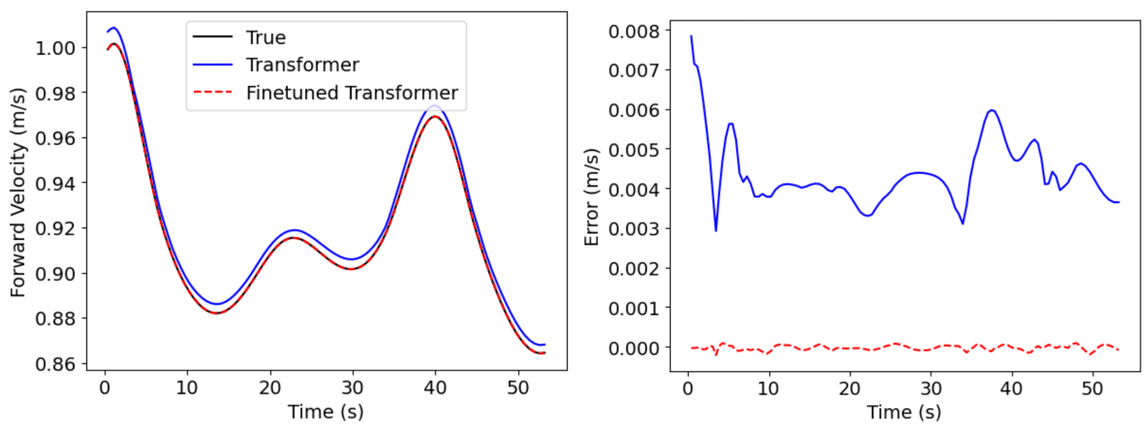

Figure 3.4. On the left is a plot of the true forward velocity output of the robot alongside the estimates of the forward velocity from the baseline transformer (blue) and the finetuned transformer (dotted red). The finetuned model has a final layer of adaptive coefficients trained exclusively on output data when driven by the disturbance generator in the test environment. On the right is a plot of the corresponding estimation error.

layers of the baseline transformer except the final layer, then perform 1-3 gradient descent steps with training data from the target environment. Another alternative is to update the least squares parameters $\hat{\boldsymbol{\theta}}$ online with other methods, such as recursive least squares with a forgetting factor, to control how much the new training data affects the parameters. The transformer's estimation error (right plot in Figure 3.4) is the robot's natural response to disturbances not considered in the training environment. Online updates must occur strategically, and likely at a low frequency, to avoid capturing components of the transient response due to new disturbances. We leave considering online update strategies other than finetuning as future work.

CHAPTER 4

ONLINE ADAPTATION TO UNMODELED DYNAMICS VIA DYNAMIC MODE DECOMPOSITION AND PASSIVE FEEDBACK CONTROL

4.1 Introduction

In Chapter 4, we proposed methods for estimating the steady-state response of a legged robot when driven by a disturbance generator, and we interpreted the estimation error as the robot's transient response, which includes the effects of dynamic modes that we ignored in prior training but were later excited online. In this chapter, we demonstrate how to close the loop on the estimation error to mitigate the effects of these dynamic modes and regulate the robot back to its steady-state behavior. Our approach leverages the observation that the robot's dynamics about a stable periodic orbit are approximately linear and learns a dynamics model for the error system via the DMDc algorithm. With a linear dynamics model for the error system, we can apply a variety of linear control techniques to regulate it to its equilibrium. We implement a passive feedback control law to stabilize the robot under passive dynamics neglected by its base controller.

In this chapter, we apply our framework in a MuJoCo simulation [50] of a Ghost Robotics Vision 60 quadruped equipped with a convex MPC framework (Section 4.2.2) to adapt to unmodeled dynamics while walking on a treadmill with a time-varying speed setting. Our results act as a proof of concept of our approach and will serve as a baseline as we improve our implementation in future iterations.

In the next section, we provide background on DMDc and the Vision 60's base controller. In Section 4.3, we outline our approach to formulating and implementing our learning-based adaptation framework, and in Section 4.4, we present preliminary results from our MuJoCo simulation.

4.2 Background

4.2.1 Dynamic Mode Decomposition With Control

DMDc is a method for learning a linear dynamics model for a system with measurable outputs [46]. Since it's purely data-driven, no prior knowledge of a dynamics model for the system is necessary to implement this approach. Assume we have a dynamical system $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$ with an output equation $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x})$ and we do not know the functions $\boldsymbol{f}$ and $\boldsymbol{g}$, but we have a length N sequence of measured outputs. We can construct matrices

$$\boldsymbol{Y} = \begin{bmatrix} | & | & & | \\ \boldsymbol{y}_0 & \boldsymbol{y}_1 & \cdots & \boldsymbol{y}_{N-1} \\ | & | & & | \end{bmatrix}, \quad \boldsymbol{Y}' = \begin{bmatrix} | & | & & | \\ \boldsymbol{y}_1 & \boldsymbol{y}_2 & \cdots & \boldsymbol{y}_N \\ | & | & & | \end{bmatrix}, \tag{4.1}$$

and solve for the best-fit linear model

$$\boldsymbol{Y}' = \boldsymbol{A}\boldsymbol{Y}, \quad \text{s.t.} \quad \boldsymbol{A} = \boldsymbol{Y}'\boldsymbol{Y}^{+}, \tag{4.2}$$

where $\boldsymbol{Y}^{+}$ is the psuedo-inverse of $\boldsymbol{Y}$. Now, we have a linear model to predict the next output of the system based on the current output. However, there are a few concerns: (1) if N is large, then A may be too large to compute, (2) the resulting system has no control inputs, and (3) if the system is highly nonlinear, then the dynamics cannot be captured with a linear approximation.

To address the first concern, the DMD algorithm computes the singular value decomposition, $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}$ and looks at the singular values along the diagonal of $\boldsymbol{\Sigma}$ to determine the number of dominant modes in the system. In practice, we scale the singular values by their sum and consider all modes with singular values past a certain threshold. The columns of $\boldsymbol{U}$ are the modes of the system ordered from left to right in order of importance in capturing the behavior of $\boldsymbol{Y}$. If there are r singular values above the threshold, we take the first r columns of $\boldsymbol{U}$ to construct $\boldsymbol{U}_r$. Then we can calculate the projection of $\boldsymbol{A}$ onto these most dominant modes as

$$\tilde{\boldsymbol{A}} = \boldsymbol{Y}'\boldsymbol{V}\boldsymbol{S}_r(\boldsymbol{S}_r\boldsymbol{S}_r^T)^{-1}\boldsymbol{U}_r \tag{4.3}$$

where $\boldsymbol{S}_r$ is the first r rows of S. The model $\boldsymbol{y}_{i+1} = \tilde{\boldsymbol{A}}\boldsymbol{y}_i$ captures most of the dynamic behavior of the system, $\boldsymbol{y}_{i+1} = \boldsymbol{A}\boldsymbol{y}_i$.

To consider systems with control inputs, we can follow the same algorithm only replacing $\boldsymbol{Y}$ with

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{y}_0 & \boldsymbol{y}_1 & \cdots & \boldsymbol{y}_{N-1} \\ \boldsymbol{u}_0 & \boldsymbol{u}_1 & \dots & \boldsymbol{u}_{N-1} \end{bmatrix}. \tag{4.4}$$

Then, we can take the first n rows and n columns of $\boldsymbol{K} = \boldsymbol{Z}\boldsymbol{Z}^+$ as the matrix $\boldsymbol{A}$, where n is the dimension of the output, and the next m columns of the first n rows as the matrix $\boldsymbol{B}$, where m is the dimension of the control input. One issue with this approach is that we need a nonzero sequence of $\boldsymbol{u}$. The standard approach to solving this issue is the periodically kick the system (i.e. provide a random exponentially decaying control input).

If the system is highly nonlinear, we can look for a Koopman Operator [7], $\boldsymbol{h}$, where $\boldsymbol{\phi} = \boldsymbol{h}(\boldsymbol{y})$, such that the dynamical system within this embedding space, $\boldsymbol{\phi}_{i+1} = \boldsymbol{L}\boldsymbol{\phi}_i$, is approximately linear. Then, we can recover the output with the inverse of $\boldsymbol{h}$. One approach to finding a Koopman Operator is to project the output to a higher dimension with an encoder neural network. In this work, we use a delay embedding so that the state of our system is a vector of the last d outputs. As the size of d increases, the learned model approaches the system's true dynamics.

4.2.2 Convex Model Predictive Control

The base controller of the Vision 60 quadruped in our MuJoCo simulation is a variant of a convex MPC framework first presented in [13] to control the MIT Cheetah 3. The controller is based on a single rigid body model (Section 2.3) with the additional assumptions that the pitch and roll angles of the trunk are small and the robot's contact sequence is fixed. With these assumptions, the dynamics model can fit within a convex program. On each iteration, the controller solves for GRFs that minimize a standard cost function quadratic in the COM and orientation trajectory of the robot and the magnitude of its GRFs, subject to a polyhedral friction cone (Section 2.4.1.2).

4.3 Methods

4.3.1 Treadmill and Bobbling Head

We add additional objects to our MuJoCo simulation as disturbances for the robot to validate our learning-based adaptation framework. Specifically, we add a long box to the environment to simulate a treadmill. We directly control the velocity of the treadmill, v_T, setting it to vary periodically with two modes,

$$v_T(t) = v_0 + \frac{1}{T_a} v_a \sin(2\pi\, T_a\, t) + \frac{1}{T_b} v_b \sin(2\pi\, T_b\, t), \tag{4.5}$$

where v_0 is its center speed, T_a and T_b are the its fundamental periods, and v_a and v_b are their corresponding amplitudes. The treadmill is the disturbance generator in our framework.

We also add passive dynamics to the robot that are not considered within the model of its base controller. We mount an inverted pendulum to the robot's back, modeled as a long, skinny box with a hinge joint oriented such that it rotates about the y-axis in the robot's body frame. We call this inverted pendulum a bobbling head, although it is similar in spirit to a manipulator arm or some other object the robot is carrying on its trunk. We add spring and damping constants such that the bobbling head is stable about its equilibrium position, which we arbitrarily chose to be the upright position.

When the robot is driven by the disturbance due to the treadmill's time-varying speed, it may excite the modes of the bobbling head. As it oscillates, it causes the trunk of the robot to wobble. The control task for our learning-based framework is to stabilize the quadruped under these oscillations. We can adjust the difficulty of the control task by increasing the mass of the head or by decreasing the spring constant. In our experiments (Section 4.4), we set the mass of the head as 5 kilograms, the length as 40 cm, the spring constant at 30 Newton-meters, and the damping coefficient as 2 Newton-meters-seconds. Figure 4.1 is a snapshot of our simulation with the treadmill and the bobbling head.

4.3.2 Moment-Matching Model

We formulate a moment-matching model as in Section 3.3.1 to estimate the steady-state output of the quadruped in response to the time-varying treadmill speed. The base controller accepts a twist command as input, so we consider the robot's velocity as the relevant output and fit a moment matching model from the disturbance generator state to the 6-dimensional velocity. We provide the moment-matching model the true fundamental periods of the disturbance, assuming the robot has a perception system and is capable of performing a Fourier analysis to identify them.

One challenge when formulating the moment-matching model is to separate oscillations in the robot's forward speed due to the time-varying treadmill from the natural oscillations of its gait cycle. We show an example of the forward speed trajectory of the robot while driven by the disturbance in Figure 4.2. We command the robot to trot at zero velocity in the world frame. The tall oscillations in the forward

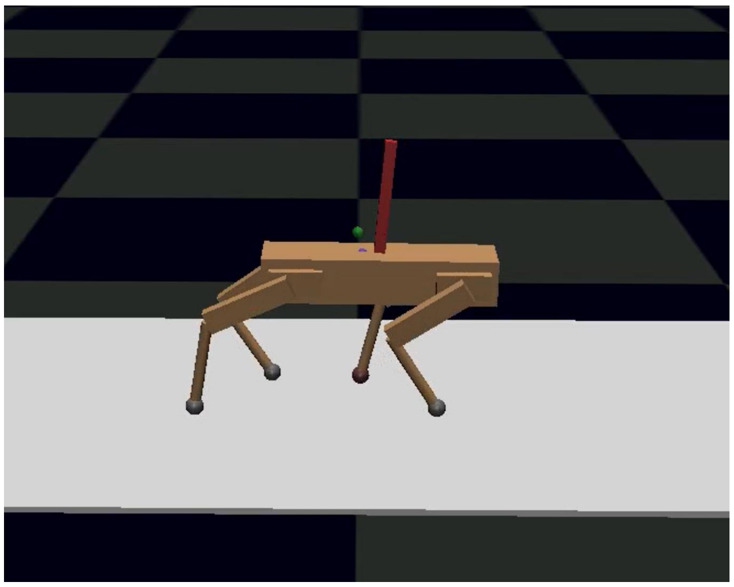

Figure 4.1. A snapshot of our MuJoCo simulation with a model of the Ghost Robotics Vision 60 Quadruped. The simulation environment includes a model of a treadmill (white) with a time-varying speed setting, as well as a bobbling head (red) mounted on the trunk of the robot. We model the bobbling head as a long, skinny box with a hinge joint oriented along the y-axis in the robot's body frame.

speed are due to the robot's gait cycle. If we were to fit a moment-matching to all of the available output data, we would mistakenly capture part of the dynamics of the robot's gait cycle rather than the steady-state response to the disturbance generator. We addressed this challenge by strategically choosing when to sample the forward speed. The circles on the figure are samples when the robot is midway through the full-stance portion of its trot gait. The black circles correspond to the full-stance phase after an odd step (left front and hind right), while the red circles correspond to that of an even step (right front and hind left). We noticed that when choosing one of these options (odd or even), the resulting trajectory appeared to have the same fundamental harmonics as the disturbance generator. We chose to sample the robot's output for our moment-matching model midway through each full stance phase after odd steps.

4.3.3 Model-Following Abstraction

We have a moment-matching model that outputs the steady-state behavior of the robot when driven by the time-varying treadmill. Now, we can collect the model-following error and the robot's input data and apply the DMDc algorithm (Section 4.2.1) to learn a linear dynamics model for the 6-dimensional error system. We train the DMDc model exclusively on input/output data from the quadruped without the bobbling head, and we test its performance both with and without it.

After removing the steady-state response of the robot, we can capture the dynamics of the remaining output with a linear approximation. Still, we choose to use a delay embedding of the robot's past output to improve our linear fit. This choice adds additional robustness when the moment-matching model does not fully capture the steady-state response. In this case, some of the output from the nonlinear dynamics will be included in the error system. With a delay embedding of sufficient length, we can still capture the dynamics of the error system.

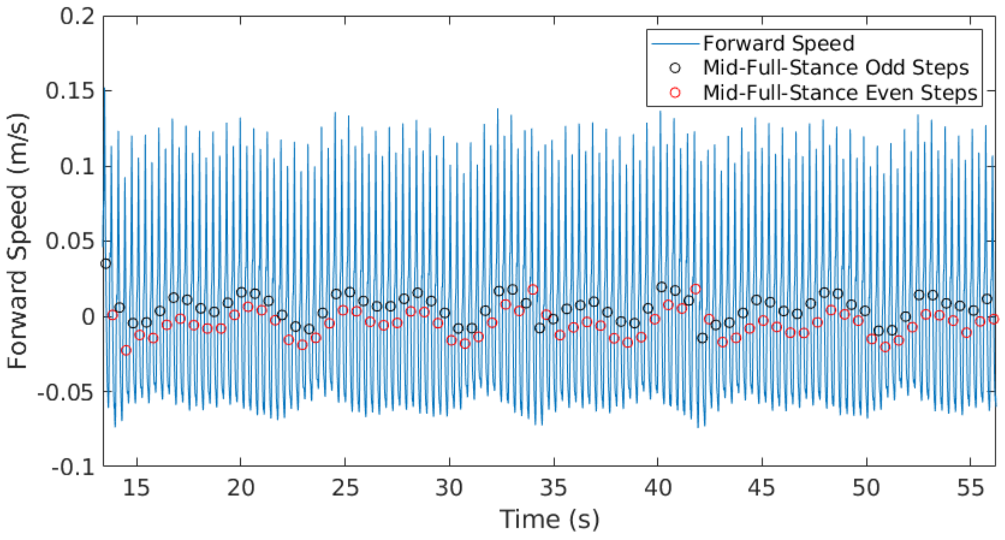

Figure 4.2. Shown is an example of a forward speed trajectory of the robot when driven by the disturbance due to the time-varying treadmill. The large oscillations in the forward speed (blue line) are due to the robot's gait cycle. We show that when sampling once per gait cycle midway through a stance phase, either after an odd step (left front and hind right) or an even step (right front and hind left), we can isolate the robot's response due to the disturbance generator. The sampled output in either of these cases appears to have the same fundamental harmonics as the disturbance generator.

The model-following abstraction takes the form,

$$\boldsymbol{z}_{i+1} = \boldsymbol{A}\boldsymbol{z}_i + \boldsymbol{B}\boldsymbol{u}_i, \tag{4.6}$$

where $\boldsymbol{z}$ is a delay embedding of the robot's output and $\boldsymbol{u}$ is its twist command. After we learn the linear approximation, we formulate a discrete-time passive feedback controller to regulate the system to its steady-state behavior. The intuition behind a passive feedback law is to construct the output of the system such that it's always dissipating energy. A discrete-time linear system is strictly passive if there exists a positive definite matrix P such that [27]

$$\begin{bmatrix} \boldsymbol{P} - \boldsymbol{A}^T\boldsymbol{P}\boldsymbol{A} & \boldsymbol{C}^T - \boldsymbol{A}^T\boldsymbol{P}\boldsymbol{B} \\ \boldsymbol{C} - \boldsymbol{B}^T\boldsymbol{P}\boldsymbol{A} & \boldsymbol{D} + \boldsymbol{D}^T - \boldsymbol{B}^T\boldsymbol{P}\boldsymbol{B} \end{bmatrix} > 0, \tag{4.7}$$

where the system's passive output takes the form

$$\boldsymbol{y}_i = \boldsymbol{C}\boldsymbol{z}_i + \boldsymbol{D}\boldsymbol{u}_i. \tag{4.8}$$

We determine the matrices $\boldsymbol{C}$ and $\boldsymbol{D}$ by finding a $\boldsymbol{P}$ that satisfies the discrete-time Lyapunov equation, setting $\boldsymbol{C} = \boldsymbol{B}^T\boldsymbol{P}\boldsymbol{A}$, and choosing a $\boldsymbol{D}$ sufficiently large so that $\boldsymbol{D} + \boldsymbol{D}^T - \boldsymbol{B}^T\boldsymbol{P}\boldsymbol{B} > 0$.

4.3.4 Training Pipeline

We follow a training pipeline to learn a moment-matching model and a model-following abstraction for the Vision 60 quadruped walking on a time-varying treadmill with a bobbling head on its trunk. We collect data of the quadruped walking on the time-varying treadmill for 40 seconds. We then fit the moment-matching model with the collected data to estimate the steady-state velocity of the robot when it's driven by this disturbance. Next, we collect data for a second period of 180 seconds, this time recording the model-following error and the robot's input data. Every 20 seconds, we add an exponentially decaying control kick to the robot's input with a max amplitude and time constant sampled from a uniform distribution. The control kick allows us to learn the matrix $\boldsymbol{B}$ of our model-following abstraction. We then perform the DMDc algorithm to learn a linear approximation of the error system. In our experiments, we used a delay-embedding of length 10. Finally, we formulate the output matrices $\boldsymbol{C}$ and $\boldsymbol{D}$ according to Equation 4.7. When we deploy our learning framework, we set the control input to the quadruped's base controller as

$$\boldsymbol{u}_i = \boldsymbol{u}_0 + K(\boldsymbol{C}\boldsymbol{z}_i + \boldsymbol{D}\boldsymbol{u}_{i-1}) \tag{4.9}$$

where $\boldsymbol{u}_0$ is the base control input and K is a scalar feedback control gain.

4.4 Results

Figure 4.3 shows an example of the output of the generator's observer (top), the corresponding moment-matching output plotted alongside the true forward velocity (middle), and the model following error (bottom), which is the moment-matching output after subtracting out the robot's true output. We sample the output of the disturbance generator at a faster rate than the robot's output because we found that when sampling at lower frequencies the observer had a larger estimation error. The moment-matching model follows the general trend of the forward velocity but has noticeable estimation error. We speculate that the machine learning techniques presented in Chapter 3 would outperform the moment-matching model in this setting, but we left validating these techniques on a quadruped as future work.

Figure 4.4 presents an example plot of the model-following error alongside the robot's input data for the quadruped's forward velocity during a model-following

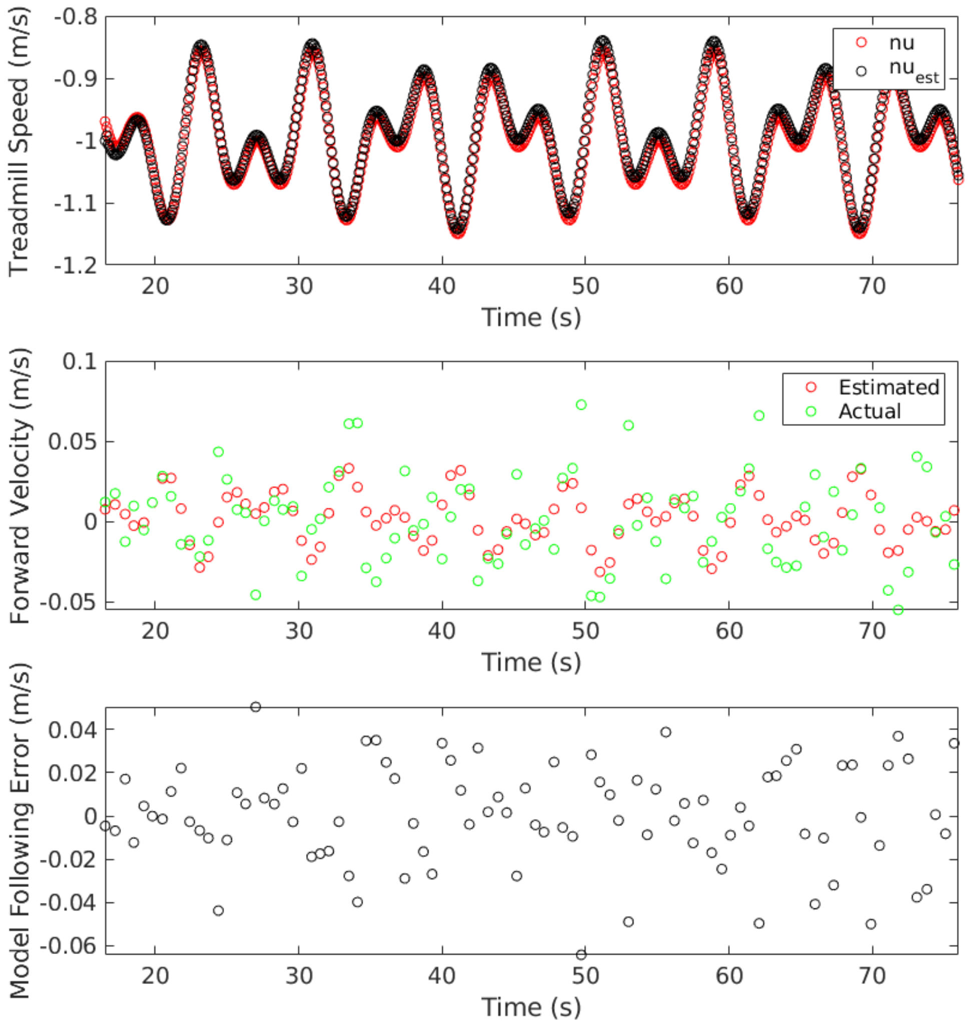

Figure 4.3. The top plot is an example of the estimates of the disturbance generator's output from its observer. We sample the disturbance generator's output at a faster speed than the robot's output to minimize estimation error. The middle plot is the corresponding moment-matching output along the x-dimension plotted alongside the true forward speed of the robot, while the bottom plot is the model following error.

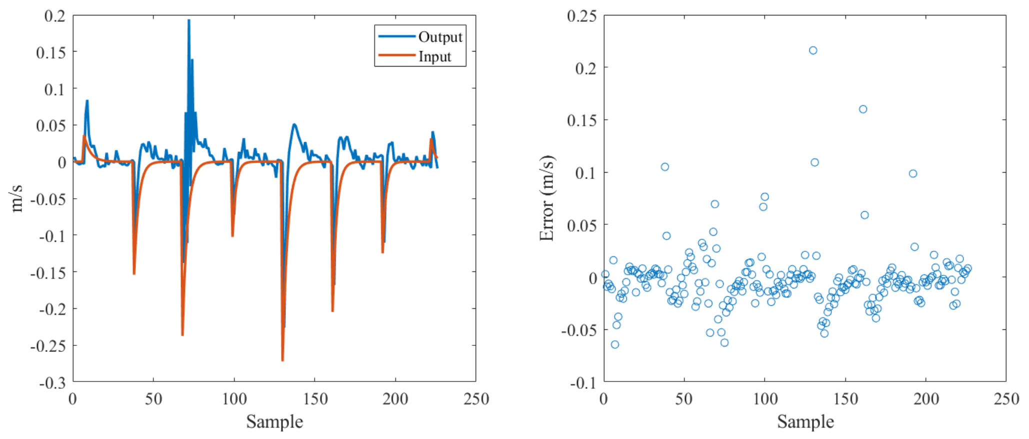

Figure 4.4. On the left is a plot of the model-following error (blue) alongside the robot's input (orange) for the forward velocity of the robot (along the x-direction in the body frame). The periodic control kicks can be seen as sudden spikes in the input, which exponentially decay to zero. The control kicks cause a sudden spike in the model-following error as the robot strays from its steady-state behavior. On the right is the DMDc fitting error for each sample (i.e. the difference between the true output and the predicted output from the learned linear approximation).

training period (left) as well as the fitting error for each sample (right). The sharp spikes in the robot's input are the periodic control kicks.

We present the root mean squared model following error for various cases with our framework. In "Baseline", there is no bobbling head and the control gain K is set to zero, and in "DMDc", there is no bobbling head and the control gain was set to 0.04. In "Baseline w/ Head", we added the bobbling head and set the control gain to 0, while in "DMDc with Head", we added the bobbling head and set the control gain to 0.02. After adding the bobbling head, the baseline model-following error more than doubles. As the robot is driven by the time-varying treadmill, the bobbling head oscillates, causing the robot to wobble and depart from its steady-state behavior. We expect the model-following error to decrease after adding the passive feedback control law, regulating the robot to the estimated steady state output of the

Errors	**Baseline**	**DMDc**	**Baseline w/ Head**	**DMDc w/ Head**
$\dot{\boldsymbol{x}}\,(\frac{\mathrm{m}}{\mathrm{s}})$	0.0063	0.0060	0.0173	0.0172
$\dot{\boldsymbol{y}}\,(\frac{\mathrm{m}}{\mathrm{s}})$	0.0109	0.0113	0.0218	0.0209
$\dot{\boldsymbol{z}}\,(\frac{\mathrm{m}}{\mathrm{s}})$	0.0106	0.0087	0.0247	0.0218
$\dot{\boldsymbol{\omega}}_x\,(\frac{\mathrm{rad}}{\mathrm{s}})$	0.0193	0.0149	0.0872	0.0767
$\dot{\boldsymbol{\omega}}_y\,(\frac{\mathrm{rad}}{\mathrm{s}})$	0.0120	0.0082	0.0493	0.0430
$\dot{\boldsymbol{\omega}}_z\,(\frac{\mathrm{rad}}{\mathrm{s}})$	0.0022	0.0017	0.0081	0.0073

TABLE 4.1

ROOT MEAN SQUARE MODEL FOLLOWING ERROR

moment-matching model. In general, we saw small reductions in the model-following error, validating the soundness of our approach.

We would expect the model-following error to decrease more significantly after adding the passive feedback control law. We speculate that we could reduce the tracking error by improving the fit of our moment-matching model to the robot's steady-state output. The current estimation error is on the same order as the robot's variation from its commanded speed. Ideally, these estimation errors would be smaller. We provide more discussion on the results and how they will inform future work in the next chapter.

CHAPTER 5

CONCLUSIONS

As legged robots become more capable of traversing various types of terrain, a next-level challenge relates to developing planning and control algorithms that can overcome uncertainty, expanding their capabilities to more extreme environments. Examples of such environments may include terrains where the robot must carefully plan its footsteps in real-time to avoid stepping on unsafe regions of the terrain, or settings where there are disturbances from the environment that cause the robot to diverge from its desired behavior. With these examples in mind, we considered two major challenges in this work: contact implicit planning and learning-based methods to adapt to new conditions online.

5.1 Contact-Implicit Planning

At the heart of the challenge in contact-implicit planning is its inherently combinatorial and continuous nature. As we include more contact options, whether they be additional end-effectors, potential contact surfaces, or contact modes (i.e. floating, static, or sliding), the exponential algorithmic complexity of the problem quickly grows out of hand. In this work, we choose to formulate the planning problem as a mixed-integer program to allow the solver to consider a wider range of contact options relative to local methods such as an NLP. We demonstrate that we can decrease the average solve time by simplifying the stance phase to be impulsive and then restoring dynamic feasibility of the trajectory with a smoothing NLP equipped with a higher fidelity dynamics model. We show that we can further decrease the average solve time of our MIP with heuristics. However, it's important to note that our approach still has exponential complexity in the worst case. Our impulsive stance technique decreases the number of continuous variables in the formulation, speeding up the evaluation of each convex relaxation during Gurobi's branch and bound search, but it can still take a large number of iterations to find the combination of integer variables which gives the globally optimal solution.

Future work may choose to take on the exponential algorithmic complexity more directly by reducing the number of available contact options in the formulation. A good place to start would be to remove unphysical combinations of contact options, such as in Section 2.5.1, or undesirable combinations, such as in Section 2.5.2.

Another possible direction is to improve the "tightness" of the MIP formulation. The convex relaxation of a tight MIP formulation always returns an integer solution. In [32], Marcucci et al. demonstrate how MIPs like the one considered in our work can be reformulated as the problem of finding the shortest path in a graph of convex sets. Their approach is inspired by the shortest path problem in a graph, which is directly transcribed as an integer program, but its convex relaxation returns an integer solution. The first convex relaxation of the formulations in [32] often returns a near (sometimes exact) integer solution. This is exciting because it could make considering the true nature of the contact optimization problem, as both a combinatorial and continuous optimization, feasible online.

In the immediate future, we plan to finetune the MPC tracking controller in Section 2 for hopping motions, enabling us to perform further simulation and hardware tests with our framework on the MIT Mini Cheetah robot. As is, the MPC controller assumes that the ground is flat, so it often fails to reliably jump onto elevated platforms. We plan to update the MPC controller to remove this assumption.

We also could extend our impulsive-stance MIP formulation to include more gait options for the robot. Currently, the framework can only plan hopping motions on all

4 contacts, but there is room to extend the formulation to other contact options such as a short stance phase on the robot's two hind legs or its two front legs, allowing for bounding behaviors. We can add this functionality by adding binary constraints that enforce the robot to be in only one contact configuration during each stance phase while allowing the solver to pick the optimal option. It also would be exciting to incorporate a perception system into our framework that can identify convex safe regions for the robot to step online.

5.2 Learning-based Adaptation

State-of-the-art control methods for legged robotics exhibit impressive robustness on a variety of terrains. However, it's often unclear how these controllers will respond in uncertain environments, where disturbances may excite dynamical modes ignored by its base controller. This problem is relevant to model-based and model-free RL approaches alike. A model-based controller may fail under dynamics neglected by its model, and a model-free RL approach may fail under dynamics neglected in previous training. Often, the training environment for an RL approach is a simulation that also fundamentally relies on physics models. Even when training on real-world data, it's possible some key dynamical modes were not excited during the data collection period.

In this work, we consider an approach that addresses the challenge of dealing with unmodeled dynamics by first learning a moment-matching model that estimates the steady-state behavior of the robot in response to a disturbance. Once we can reliably estimate this stable behavior, we can subtract it from the true output of the robot to estimate its transient response to the disturbance due to dynamical modes ignored by the moment-matching model. We implement DMDc to learn a linear approximation of the dynamics about the robot's steady-state behavior and add a passive feedback controller to regulate it back to this behavior. We demonstrate how LSTMs and transformer neural networks offer a more robust approach than moment-matching models when estimating the steady-state output of the robot. Specifically, the neural networks maintain a higher level of accuracy across a wider domain of environments. We also demonstrate how our approach can adapt a Vision 60 quadruped to the unmodeled dynamics of an inverted pendulum when driven by the disturbance of a treadmill with a time-varying speed setting. Further simulation tests, varying the properties of the pendulum and the treadmill, will inform future analysis on the performance of our approach.

A challenge in learning-based adaptation is both deciding and controlling how much new information to incorporate into your framework without losing key old information. In Chapter 3, we demonstrated how we can remove the last layer from our neural network and replace it with adaptable linear coefficients. This transfer learning technique maintains information relevant across all domains that's embedded within the parameters of the neural network's base. However, it's unclear how we should update the coefficients of the last layer to improve our estimate of the steady-state behavior online. An aggressive strategy risks capturing some of the undesirable transient behavior in our model. We finetune the network with a single linear regression to data collected from the current environment, acknowledging the captured behavior will be what our adaptation strategy regulates towards, regardless of its quality. Future work may consider more clever recursive adaptation strategies.

A promising direction for future work would be to restructure the learning problem to minimize tracking error rather than estimate the dynamics of the error system. In [48], Richards et al. propose a learning-based adaptive control framework for dealing with unmodeled dynamics and demonstrate its effectiveness in adapting a quadcopter to time-varying wind conditions. They label their approach as "control-oriented" rather than "regression-oriented," citing past work in adaptive control literature [2] to support their intuition that the downstream control objective should be prioritized over the regression objective. They demonstrate that perfect estimation of the parameters of the disturbance is unnecessary for tracking convergence. In our approach, we learn a linear approximation of the dynamics of our error system via DMDc and formulate a passive feedback controller, but a better approach may be to learn a control policy with the objective of minimizing the tracking error between the robot's current output and its steady-state output.

A next-level challenge is to devise control methods that can adapt to new environments and tasks far outside of the domains considered in the training environment. One potential approach is to learn a set of action primitives from which any relevant task could be constructed. However, there is uncertainty as to whether such a set even exists, and even if we had the set, it's unclear how we could adaptively mix it to achieve the desired behavior to perform a new task or traverse a new environment online.

We also have plans for our learning-based adaptation framework in the immediate future. Our first step will be to add an interface to the MuJoCo simulation to allow for large-scale data collection, allowing us to train a transformer to estimate the steady-state output of the quadruped. We suspect that the machine learning techniques proposed in this work will outperform the moment-matching model considered in Chapter 4. If we can better capture the steady-state behavior, it also should improve the performance of our model following abstraction because more of the nonlinear dynamics from the steady-state behavior will be removed from the error system, leading also to an improvement in our feedback control law. Before deploying our framework on hardware, we may need to adjust the number of parameters in our transformer to make our approach computationally feasible with the robot's onboard computer. We also may investigate alternative methods for sampling the robot's output to isolate the steady-state behavior. One possibility may be to project the robot's output onto the fundamental modes of the disturbance generator, removing the oscillations due to the gait cycle. When we deploy our framework on hardware, we plan to add some form of passive unmodeled dynamics on the robot's back. It could be a mass-spring-damper system, an inverted pendulum, or even a box of water.

BIBLIOGRAPHY

1. B. Aceituno-Cabezas, C. Mastalli, H. Dai, M. Focchi, A. Radulescu, D. G. Caldwell, J. Cappelletto, J. C. Grieco, G. Fernández-López, and C. Semini. Simultaneous contact, gait, and motion planning for robust multilegged locomotion via mixed-integer convex optimization. *IEEE Robotics and Automation Letters*, 3 (3):2531–2538, 2018. doi: 10.1109/LRA.2017.2779821.

2. J. Aseltine, A. Mancini, and C. Sarture. A survey of adaptive control systems. *IRE Transactions on Automatic Control*, 6(1):102–108, 1958. doi: 10.1109/TAC.1958.1105168.

3. A. Astolfi. Model reduction by moment matching for linear and nonlinear systems. *IEEE Transactions on Automatic Control*, 55(10):2321–2336, 2010. doi: 10.1109/TAC.2010.2046044.

4. A. Aydinoglu and M. Posa. Real-time multi-contact model predictive control via admm, 09 2021.

5. Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

6. T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

7. S. L. Brunton, M. Budišić, E. Kaiser, and J. N. Kutz. Modern koopman theory for dynamical systems, 2021.

8. F. Chollet. *Deep Learning with Python, 2nd ed.* Manning Publications, New York, NY, 2022.

9. S. L. Cleac'h, T. Howell, S. Yang, C.-Y. Lee, J. Zhang, A. Bishop, M. Schwager, and Z. Manchester. Fast contact-implicit model-predictive control, 2023.

10. Z. Colter. Graduate artificial intelligence lecture 9: Mixed integer programming, February 2014.

11. R. Deits and R. Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 279–286, 2014.

12. R. Deits and R. Tedrake. Efficient mixed-integer planning for uavs in cluttered environments. In *IEEE International Conference on Robotics and Automation*, pages 42–49, 2015. doi: 10.1109/ICRA.2015.7138978.

13. J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1–9, 2018. doi: 10.1109/IROS.2018.8594448.

14. Y. Ding, C. Li, and H.-W. Park. Single leg dynamic motion planning with mixed-integer convex optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 10 2018. doi: 10.1109/IROS.2018.8594161.

15. Y. Ding, C. Li, and H.-W. Park. Kinodynamic motion planning for multi-legged robot jumping via mixed-integer convex program. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 10 2020. doi: 10.1109/IROS45743.2020.9341572.

16. F. Farshidian, E. Jelavic, A. Satapathy, M. Giftthaler, and J. Buchli. Real-time motion planning of legged robots: A model predictive control approach. In *IEEE-RAS International Conference on Humanoid Robotics*, pages 577–584. IEEE, 2017.

17. R. Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

18. R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter. Perceptive locomotion through nonlinear model predictive control. 2022.

19. Gurobi Optimization Inc. Gurobi optimizer reference manual. *[Online]*, 2022.

20. M. Herceg, M. Kvasnica, C. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, pages 502–510, Zürich, Switzerland, July 17–19 2013.

21. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

22. C. Johnson. Solving the rubik's cube with stepwise deep learning. *Expert Systems*, 38, 05 2021. doi: 10.1111/exsy.12665.

23. J. M. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Zídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. A. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W.

Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021.

24. L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. doi: 10.1109/70.508439.

25. D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim. Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control, 2019. URL `https://arxiv.org/abs/1909.06586`.

26. J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47): eabc5986, 2020. doi: 10.1126/scirobotics.abc5986. URL `https://www.science.org/doi/abs/10.1126/scirobotics.abc5986`.

27. M. D. Lemmon, P. M. Wensing, V. Kurtz, and H. Lin. Learning to control robot hopping over uneven terrain. In *2022 American Control Conference (ACC)*, pages 520–525, 2022. doi: 10.23919/ACC53348.2022.9867630.

28. H. Li and P. M. Wensing. Hybrid systems differential dynamic programming for whole-body motion planning of legged robots. *IEEE Robotics and Automation Letters*, 5(4):5448–5455, 2020. doi: 10.1109/LRA.2020.3007475.

29. H. Li, T. Zhang, W. Yu, and P. M. Wensing. Versatile real-time motion synthesis via kino-dynamic mpc with hybrid-systems ddp, 2022.

30. J. Lofberg. Yalmip: A toolbox for modeling and optimization in matlab. *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 284–289, 2004.

31. Z. Manchester and S. Kuindersma. Variational contact-implicit trajectory optimization. In *International Symposium of Robotics Research*, 2017.

32. T. Marcucci, J. Umenberger, P. A. Parrilo, and R. Tedrake. Shortest paths in graphs of convex sets, 2022.

33. G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal. Rapid locomotion via reinforcement learning, 2022.

34. C. Mastalli, W. Merkt, J. Marti-Saumell, H. Ferrolho, J. Sola, N. Mansard, and S. Vijayakumar. A feasibility-driven approach to control-limited DDP. *Autonomous Robots (to appear)*, 2022.

35. G. P. McCormick. Computability of global solutions to factorable nonconvex programs. *Mathematical programming*, 10(1):147–175, 1976.

36. T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7, 01 2022. doi: 10.1126/scirobotics.abk2822.

37. J. Norby and A. 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, 2020. doi: 10.1109/IROS45743.2020.9341438.

38. D. Orin, A. Goswami, and S.-H. Lee. Centroidal dynamics of a humanoid robot. *Autonomous Robots*, 35, 10 2013. doi: 10.1007/s10514-013-9341-4.

39. R. Orsolino, M. Focchi, C. Mastalli, H. Dai, D. Caldwell, and C. Semini. Application of wrench-based feasibility analysis to the online trajectory optimization of legged robots. *IEEE Robotics and Automation Letters (RA-L)*, 2017. doi: 10.1109/LRA.2018.2836441.

40. M. O'Connell, G. Shi, X. Shi, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung. Neural-fly enables rapid learning for agile flight in strong winds. *Science Robotics*, 7(66):eabm6597, 2022. doi: 10.1126/scirobotics.abm6597. URL `https://www.science.org/doi/abs/10.1126/scirobotics.abm6597`.

41. A. Pandala, R. T. Fawcett, U. Rosolia, A. D. Ames, and K. A. Hamed. Robust predictive control for quadrupedal locomotion: Learning to close the gap between reduced- and full-order models. *IEEE Robotics and Automation Letters*, 7(3): 6622–6629, 2022. doi: 10.1109/LRA.2022.3176105.

42. H.-W. Park, P. M. Wensing, and S. Kim. Online planning for autonomous running jumps over obstacles in high-speed quadrupeds. In *Robotics: Science and Systems*, 2015.

43. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

44. M. Posa, C. Cantu, and R. Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014. doi: 10.1177/0278364913506757. URL `https://doi.org/10.1177/0278364913506757`.

45. I. Poulakakis and J. Grizzle. The spring loaded inverted pendulum as the hybrid zero dynamics of an asymmetric hopper. *Automatic Control, IEEE Transactions on*, 54:1779 – 1793, 09 2009. doi: 10.1109/TAC.2009.2024565.

46. J. L. Proctor, S. L. Brunton, and J. N. Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016. doi: 10.1137/15M1013857. URL `https://doi.org/10.1137/15M1013857`.

47. M. Raibert, H. B. Brown, and M. Chepponis. Experiments in balance with a 3d one-legged hopping machine. *The Internaional Journal of Robotics Research*, 1984.

48. S. M. Richards, N. Azizan, J.-J. Slotine, and M. Pavone. Control-oriented meta-learning, 2022.

49. T. Schouwenaars, B. De Moor, E. Feron, and J. How. Mixed integer programming for multi-vehicle path-planning. *European Control Conf.*, 01 2001.

50. E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

51. A. K. Valenzuela. *Mixed-integer convex optimization for planning aggressive motions of legged robots over rough terrain.* PhD thesis, Massachusetts Institute of Technology, 2016.

52. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

53. P. M. Wensing and D. E. Orin. High-speed humanoid running through control with a 3d-slip model. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5134–5140, 2013. doi: 10.1109/IROS.2013.6697099.

54. A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567, 2018. doi: 10.1109/LRA.2018.2798285.

55. A. Wächter and L. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 03 2006. doi: 10.1007/s10107-004-0559-y.

This document was prepared & typeset with pdfLaTeX, *and formatted with* NDdiss2$_\varepsilon$ *classfile (v3.2017.2[2017/05/09])*