University of California
Santa Barbara

Accurate Step Length Control Strategies for Underactuated and Realistic Series Elastic Actuated Hoppers via High Order PFL

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Patrick W. Terry

Committee in charge:

Professor Katie Byl, Chair
Professor João Hespanha
Professor Andy Teel
Professor Kimberly Turner

September 2016

The Dissertation of Patrick W. Terry is approved.

Professor João Hespanha

Professor Andy Teel

Professor Kimberly Turner

Professor Katie Byl, Committee Chair

August 2016

Accurate Step Length Control Strategies for Underactuated and Realistic Series Elastic Actuated Hoppers via High Order PFL

by

Patrick W. Terry

Dedicated to my family, friends, lab mates, and everyone who has put up with studying with me in the past.

Acknowledgements

First and foremost, I am incredibly grateful to my advisor Professor Katie Byl. Without her support, guidance, and encouragement my work could not have been completed. She provided not only the opportunity to gain invaluable research and hardware experience working with robots, but also the fun and amazing work environment that is the UCSB Robotics Lab.

I would also like to extend my deepest gratitude to my committee members Professor João Hespanha, Professor Andy Teel, and Professor Kimberly Turner for supporting my academic goals. In addition to research guidance, the Control Systems courses they taught were invaluable in motivating me to pursue future studies.

Since it's founding, I have always found the UCSB Robotics Lab to be one of the best working environments I've been a part of. I would like to thank all of my past and present colleagues: Giulia Piovan, Hosein Mahjoubi, Brian Satzinger, Paul Filitchkin, Min-Yi Chen, Marco Rodriguez-Suarez, Sebastian Sovero, Cenk Oguz Saglam, Marten Byl, Chelsea Lau, Virgile Paris, Tom Strizic, Nihar Talele, Samantha Samuelso, Asutay Ozmen, Sepehr Seifi, and all other international and local students that were with us. Special thanks to Marty and Giulia for the endless hours spent working on FRANK hardware.

I would also like to thank all the friends I have both studied with and spent time with living in Santa Barbara, as well as all my past contacts at Santa Barbara City College, where my academic pursuit began long ago.

I also of course must thank my family for providing the support and encouragement necessary for me to pursue my seemingly never-ending goals.

Curriculum Vitæ

Patrick W. Terry

Education

2016 (expected)	**University of California, Santa Barbara** **Ph.D.** in Electrical and Computer Engineering
2011	**University of California, Santa Barbara** **M.S.** in Electrical and Computer Engineering G.P.A. **3.96**
2010	**University of California, Santa Barbara** **B.S.** in Electrical and Computer Engineering G.P.A. **3.97**

Publications

1. **P. Terry**, G. Piovan, and K. Byl. *SLIP-based and HOPFL Enforced Step Length Control for Stable Body Motions of Realistic SEA Hoppers.* Submitted to ACC 2017, in review.
2. **P. Terry**, G. Piovan, and K. Byl. *Reachability Based and High Order Partial Feedback Linearization Enforced Control Strategies for Realistic Series-Elastic Actuated Hopping Robots.* Submitted to IEEE Transactions on Robotics, in review
3. **P. Terry**, G. Piovan, and K. Byl. *Towards Precise Control of Hoppers: Using High Order Partial Feedback Linearization to Control the Hopping Robot FRANK.* Accepted for IEEE CDC 2016
4. **P. Terry** and K. Byl. *CoM Control for Underactuated 2D Hopping Robots with Series-Elastic Actuation via Higher Order Partial Feedback Linearization.* In Proc. IEEE Conf. on Decision and Control (CDC), 2015.
5. K. Byl, Brian Satzinger, Tom Strizic, **P. Terry** and Jason Pusey. *Toward agile control of a flexible-spine model for quadruped bounding.* in Proc. SPIE 9468, Unmanned Systems Technology XVII, 94680C, 2015; doi:10.1117/12.2177432
6. **P. Terry** and K. Byl. *A Higher Order Partial Feedback Linearization Based Method for Controlling an Underactuated Hopping Robot with a Compliant Leg.* In Proc. IEEE Conf. on Decision and Control (CDC), 2014.

Abstract

Accurate Step Length Control Strategies for Underactuated and Realistic Series Elastic Actuated Hoppers via High Order PFL

by

Patrick W. Terry

Among the different types of legged robots, hopping robots, aka hoppers, can be classified as one of the simplest sufficient models that capture the important features encompassed in dynamic locomotion: underactuation, compliance, and hybrid features. There is an abundance of work regarding the implementation of highly simplified hopper models, the prevalent example being the spring loaded inverted pendulum (SLIP) model, with the hopes of extracting fundamental control ideas for running and hopping robots. However, real world systems cannot be fully described by such simple models, as real actuators have their own dynamics including additional inertia and non-linear frictional losses. Additionally, implementing feedback control for hopping systems with significant amounts of compliance is difficult as the input variable does not instantaneously change the leg length acceleration. The current state-of-the-art of step length control in the presence of non-steady state motions required for foothold placement is not precise enough for operation in the real world. Therefore, an important step towards demonstrating high controllability and robustness to real-world elements is in providing accurate higher order models of real-world hopper dynamics, along with compatible control strategies.

Our modeling work is based on a series-elastic actuated (SEA) hopping robot prototype constructed by our lab group, and we provide verifying hardware results that high order partial feedback linearization (HOPFL) can be implemented directly on the leg state of the robot. Using HOPFL, we investigate two paths of compatible trajec-

tory generation that can accomplish desirable tasks such as precise foothold planning. We investigate the practicality of using SLIP-based trajectory generation techniques on more realistic hopping robots, and show that by implementing HOPFL directly on the robot's leg, we can make use of computationally fast SLIP-based approximations, account for non-trivial pitch dynamics, and improve the state-of-the-art of precision step length control for SEA hoppers. We also consider control strategies towards hoppers for which SLIP-based trajectories may not be compatible, by planning all ground reaction force vector (GRF) components during the stance phase concurrently, using a lower order and very general model to construct trajectories for the system's center of mass (CoM), and maintain body stability by controlling the orientation of the GRF directly. While not purely analytical as our SLIP-based approaches, this method is general enough to work on a variety of hopping robots that are not necessarily kinematically structured resembling the classical SLIP model.

Contents

Chapter 1

Introduction and Literature Review

In a time with self-driving cars looming on the horizon, it is natural to ask the question "do we really need walking robots in our lives?" While we live in a world where transportation of both ourselves and objects we use is dominated by wheeled vehicles and/or wheel-based tools, it's important to remember that to actually *accomplish tasks* in the real world we typically have to transition between surfaces, walk on uneven ground, traverse stairs, bridge gaps, etc. This is due to the fact that animals, humans, and legged systems in general can operate where only intermittent footholds are available, allowing them to traverse rough terrain and operate in areas where wheeled vehicles typically have no hope of reaching. While seemingly trivial to us as humans, legged locomotion is a highly complex task involving dynamics, mechanics, sensing, stability, and many other features resulting in a very much unsolved and open problem [1, 2]. Building a robot capable of walking and running is a very involved task. First, we must appropriately model the dynamics of locomotion in order to understand how such a system can be controlled. Even a minimalistic model for a walking robot is significantly more complex than most wheeled systems, and the path towards developing controllers for realistic robots with actuator dynamics, sensor requirements, impact dynamics, weight requirements, and other

real-word features is a rich and challenging problem.

For robots to be feasible to operate in the real world, it is paramount to consider energy efficiency. Motivated by the need for responders to send help in disaster scenarios where damaged and/or uneven terrain prevents wheeled-based transport but humans also cannot go, such as the 2011 Fukushima disaster, the DARPA Robots Challenge (DRC) was recently held to varying levels of success [3]. While able to accomplish a wide range of tasks, these robots only had roughly one hour of battery life. The ATLAS robot in particular was reported to have an unbearably high cost of transport of 20, which is over 130 times worse than a human. Since legged robots impact the ground at every step, using springs as energy storage devices [4] is one potential way to gain the much needed improvement of state-of-the-art robot energy efficiency. The idea that we should consider compliance as a key feature encompassed in locomotion is also inspired by nature [5]. The key difference between running and walking is the inclusion of an aerial, or *flight*, phase, where energy stored during the ground contact *stance phase* is released, typically through compression and the subsequent expansion of a spring-like element within the leg [6, 1, 7]. It is for these reasons that compliant legged locomotion has been and will be an active area of study for many years.

1.1 Why Study "Simple" Hopping Robots?

The simplest sufficient model in legged locomotion to capture the critical features of compliance, underactuation, and hybrid dynamics is a monopod hopping robot, or "hopper". It has been shown there is a direct correlation between single-legged locomotion and multi-legged locomotion [8, 9], where in fact multiple legs acting in unison can be reasonably approximated as a single "virtual" leg acting as a function of the others towards the center of mass (CoM) of the system being considered. A typical hopping robot therefore

consists of a rigid body attached to a sprung leg, resulting in a system with potentially very high energy efficiency at the cost of exhibiting difficult to regulate *stance phase* dynamics. In general, control of the resulting *flight phase* entails control of a *touch-down angle* and, if available, a *thrust actuator* during stance [10, 11, 12, 13, 14]. The prevalent and most fundamental model used to describe the motion of hopping robots is the Spring Loaded Inverted Pendulum (SLIP) [15]. The classical SLIP consists of a massless leg and single point-mass, and has 2 degrees of freedom (DoF), a leg length and angle. SLIP can also be augmented with an additional thrust actuator in the form of active spring compression [16, 17, 18]. It has been shown that biological data for the gait patterns of various insects and animals can be approximately fit to trajectories generated from solutions to the SLIP model [19, 20, 21]. Despite it's simplicity, the stance phase dynamical equations are not analytically solvable due to the time-varying leg length, presenting a serious limitation in terms of computing control inputs and/or state prediction online. This, along with the extremely short duration ground contact phase in which typically all control authority must be accomplished in order to regulate the longer, largely uncontrolled ballistic phase, is what makes controlling these systems to land in predefined footholds precisely a complex, difficult, and for the most part unsolved problem for more complex hopping systems.

1.2 State of the Art for Hopping Legged Systems

One of the ultimate objectives in legged locomotion is the ability to follow a set of predefined footholds in the environment, which can be evenly spaced, irregularly spaced, or even time-varying depending on the terrain. It is therefore useful to correlate the state-of-the art of hopping robotics systems directly to step length accuracy. The majority of recent work with hopping robots, and even legged robots in general, has been geared

towards providing a discrete set of stable gaits that is robust to external disturbances. While certainly a desirable feature for real-world robots to have, steady-state forward motion alone is not sufficient for real-world terrain navigation. In terms of precise step length regulation results, there are a wide set of results ranging from SLIP to real robots. The general trend seen is that relatively precise control can be achieved for simulations of simple SLIP-like models, while both real robot hardware and human subjects still remains far less accurate in foothold accuracy.

Accurate control of the SLIP model has improved considerably due to fairly recent advancements in approximation techniques regarding the system's stance phase dynamics. The problem of approximating the analytically unsolvable dynamics of the classical SLIP model has been well addressed in the case of symmetric, steady state motion [22, 23, 24, 25, 26, 27]. These results alone however are not sufficient for non-steady-state operation, therefore new work [28] has proposed to augment the SLIP model with an active term to simplify the system's dynamics and improve accuracy of the approximations, particularly in the non-symmetric case. Many different forms of actuation and control strategies for adding energy to SLIP-based systems have been investigated, including direct modification to the leg length [29, 30], torques at the hip [31], and most commonly active spring compression and decompression [28, 16, 32]. It is shown in [28, 16] that maximizing the reachable space of SLIP requires the actuator moving throughout the entire stance phase, and the reachable space can in fact be analytically approximated using two actuator switching times. This parameterization is powerful, as it allows for the implementation of algorithms online, and it has been shown that precision foothold placement can be achieved with Active SLIP online using these methods [28]. Although highly simplified in the case of Active SLIP, this thrust actuator is a series-elastic actuator (SEA), which has been shown to be an effective means of actuating a variety of hopping robots [17, 18, 33, 34]. The UCSB Robotics Lab Group has successfully proto-

typed several SEA 1D Hoppers [32] in order to test leg mechanisms for the 2D hopping robot FRANK, which is presented later in this dissertation.

It has been hypothesized that we can use fundamental behavior and perhaps control strategies from SLIP on more realistic robots [35]. Transitioning from SLIP to real-world robots requires the inclusion of many realistic elements such as actuator dynamics, leg mass and inertia, and most importantly the inclusion of a robot body, where electronics and actuator mechanisms are typically mounted. Consisting simply of a point-mass, SLIP does not have any notion of body dynamics, and most work augmenting SLIP with such dynamics consists of simply adding an underactuated joint exactly on the hip [36, 37, 28, 38], where the SLIP mass is located. This approximation simplifies the pitch dynamics considerably [38], however, expecting the robot body CoM to be exactly collocated to the hip can be quite restrictive and not representative of how many real robots are constructed. The problem of stabilizing non-trivial pitch dynamics for these so-called "asymmetrical hoppers" has had considerably less attention in the literature but has been previously studied [39, 40, 41], however, the problem remains largely unsolved in terms of precise foothold placement.

Fairly recent work by Poulakakis and Grizzle based on the "asymmetrical" hopping robot Thumper [42, 43, 41], has shown that Hybrid Zero Dynamics (HZD) based methods can be applied to embed classical SLIP solutions as steady-state trajectories for these kind of systems. While body stability is analytically provable in the case of steady state gaits, these methods are highly complex and somewhat ill-suited for the case of foothold selection, as this can require uneven ground levels and non-steady-state gaits to be feasible. Thumper is only one of a number of SLIP-like robots that have been designed and built throughout the years, starting from the famous Raibert hoppers [38, 44]. Notable are Zeglin's Bow Leg Hopper [45, 46], the ARL-Monopod II [47], BiMASC [48], and ATRIAS [49], a human scaled bipedal robot with dynamics modeled as a SLIP

with the goal to achieve similar energetics. The majority of experiments performed with these robots have advanced the state-of-the-art in terms energy efficiency and robustness to external disturbances in steady-state motion for both 2D and 3D locomotion, but few since Raibert have focused on step length regulation for realistic hoppers.

The most well known studies on hopping robots were those conducted by Raibert in the '80s for both planar 2D hoppers [38, 50, 51], and a 3D Hopper [52]. The hardware experiments in step length regulation conducted by Hodgins and Raibert [44], despite having a very basic and fundamental control formulation, still more or less represent the current state-of-the-art in terms of precision step length control on real hopping robots, shown below in Fig. 1.1.

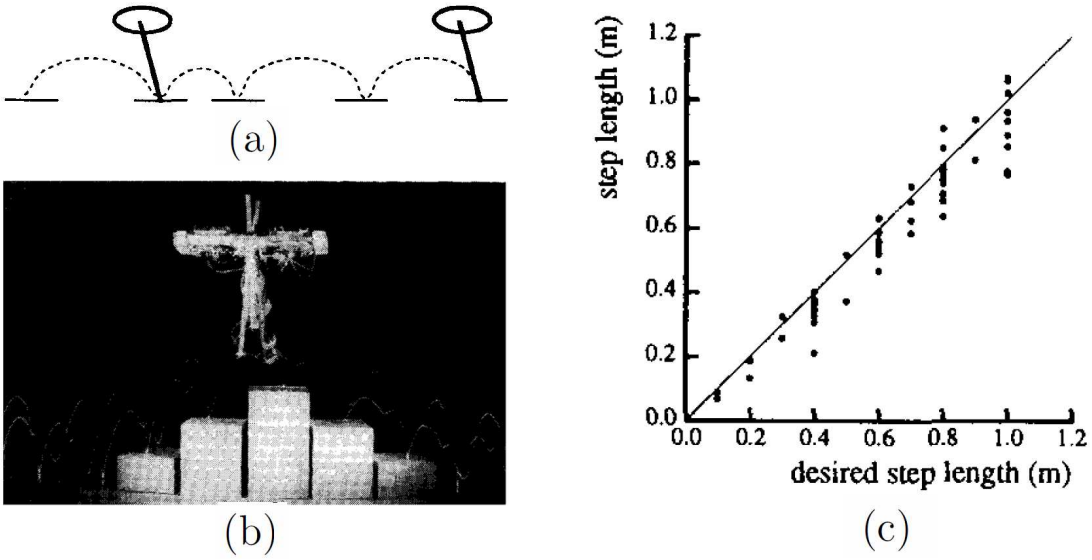


Figure 1.1: Foothold selection (a) was implemented experimentally on Raibert's famous hopping robots (b). Experimental results were impressive and could accomplish tasks such as traversing uneven terrain, but the step length accuracy (c) was not very precise. From J. K. Hodgins and M. N. Raibert, *Adjusting step length for rough terrain locomotion*, IEEE Transactions on Robotics and Automation, 1991, [44].

The control implementation with Raibert hoppers was essentially decoupled into three components: forward velocity regulation, body attitude stability, and hopping energy level. The technique used to regulate step length was to modify the touch-down angles

with a feedback law and use a feed-forward guess to inject a precomputed amount of energy with the thrust actuator. As we see in Fig. 1.1, the results do indeed suggest that the fundamental trends have been captured, however, the mean step length error is well above 10%, with maximum errors up to 25-50% in some cases. In order to improve these results, we need to more accuracy account for the dynamical coupling between the body, leg angle, leg compression, and actuator dynamics and develop suitable control methods capable of non steady state motion.

1.3 Towards Better Step Length Control

The work presented here strives to increase foothold placement control accuracy for non-steady-state gaits. Footholds in the real world may be extremely irregularly spaced, so methods that rely on forcing the robot's gait to converge to steady state motions are not always going to be useful. It is typically desirable to construct control frameworks that are very general and can be easily deployed on different kinds of systems. In reality, however, for any specific piece of hardware (or equivalently, a realistic dynamic model of a robot) design of accurate control for various degrees of freedom of the robot will inevitably involve specific aspects of the dynamics unique to that system. Thus, much of the work presented here involves applying approximations and control laws such that analytically unsolvable dynamics can be approximated and forward solved online specifically for SEA Hoppers, to obtain precise results. The methods and control variables vary depending on the hopper model considered, and Section II overviews dynamical modeling of hopping robotic systems, beginning with SLIP and building towards more realistic implementations such as the robot FRANK [53].

Applying feedback control for highly compliant systems, and hopping robots in particular, is a difficult problem as the second-order dynamics are largely dominated by the

spring force. The classical method of controlling hoppers is to take discrete measurements at each hop, perhaps at the apex height during flight, and to use these measurements with a discrete controller to generate constant feed-forward commands for the subsequent step. These results are highly inaccurate, however, as we saw in Fig. 1.1, as they do not incorporate any feedback correction continuously on the system during stance. In order to develop feedback control laws for our underactuated hoppers, we illustrate how to adopt the method of Partial Feedback Linearization (PFL) to regulate either the leg length directly, or auxiliary variables that are a function of the leg length. Applying feedback linearization techniques to this system is a challenging problem not only because the system is underactuated, but also because the compliant leg makes traditional acceleration-based PFL methods impossible due to the force being set instantaneously by the spring length. It is however possible to extend the PFL technique by linearizing a higher order term, and thus in Section III we develop control laws for compliant hopping systems via this extension. By applying high order partial feedback linearization (HOPFL) directly on the leg state of the robot, we gain the ability to reasonably predict the take-off leg velocity, and Section IV illustrates how apex height regulation can be accomplished both in 1D and 2D using such control methods, and provides hardware results implementing HOPFL on the robot FRANK.

The practicality of using specific SLIP-based solutions as references for more realistic robots is another topic of interest, as one would hope the recent SLIP approximation methods by Piovan and Byl [28, 16] result in SLIP not only being a source of inspiration when studying qualitative features of hopping/running gaits, but also a tool for generating analytical trajectories with low computational cost that can be used as references in more complex hopping systems. It is illustrated in Section V that for the operating ranges of many hopping robots indeed this is the case, by building on [28] and modifying the construction slightly, we can use these SLIP-based approximations as references for our

HOPFL leg controller [53, 54], and obtain state-of-the-art precision for step length control of underactuated SEA hoppers, even in the presence of non-trivial leg-body coupling dynamics [55]. Since these methods are based on approximations constructed analytically as a function of a very small number of input parameters, they make full use of the increasing power of parallel processing and are well suited for future deployment online.

The problem of applying HOPFL to regulate non-SLIP-like hopping robots is also considered. The majority of hopping robots in hardware are equipped with a prismatic SEA leg that mimics the structure of the classic SLIP model. For these kinds of systems, using HOPFL to control the leg state is a feasible approach. However, HOPFL-based methods can also be applied to robot models exhibiting more complicated leg kinematics, including knee-like rotational joints. The construction of a HOPFL controller can be adopted to work directly on the center of mass [56, 57] for compliant hopping robots during stance phase, where both control inputs (torque at the leg and torque at the hip) are planned concurrently to orient the ground reaction force (GRF) vector and maintain body stability. This method makes use of a lower dimensional model, more general than SLIP, consisting of a point mass traversing a pre-defined quadratic trajectory with the GRF oriented in the direction of the mass location. While allowing for more general systems, we lose the ability to have any hope of using an online optimization approach here, but we can still regulate potentially asymmetric gaits. Section VI illustrates this approach for both SEA 2D hoppers and the Compliant 3-link Robot, a hopper with more complicated leg geometry that makes application of SLIP-based step length regulation algorithms quite difficult or impossible. Lastly, Section VII contains concluding remarks.

Chapter 2

Dynamic Modeling of Hopping Robots

This chapter overviews the structure and dynamics of hopping robots, starting with the most fundamental SLIP model and work towards more realistic and complex robots. For the SEA 1D and 2D robots models discussed, we also present hopping robot hardware constructed by the UCSB Robotics Lab.

2.1 The Active SLIP model

As its name suggests, the SLIP model is a point mass attached to a massless spring leg. Classically, this system is conservative, i.e., there is no net energy change. To allow for energy variations, with the goal of undergoing different terrain profiles and being able to change stride throughout motion, we consider an actuated version of the SLIP model. In the active SLIP, the leg is equipped with a series-elastic actuator to artificially compress/extend the spring, thus varying the system's net energy. The SLIP model is a hybrid system, and its dynamics can be divided in two phases, as in Fig. 2.1: a *flight*

phase, purely ballistic with respect to the system's center of mass; and a *stance phase*, characterized by ground contact between the foot and the ground. The stance phase can be described as being divided into two phases: *compression* and *expansion*, which corresponds to the direction the system spring is accelerating. L is the leg length and θ

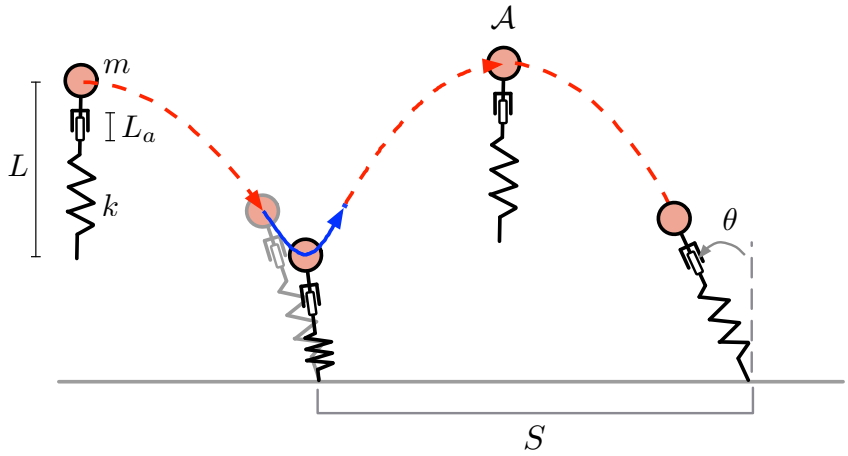


Figure 2.1: The Spring Loaded Inverted Pendulum (SLIP) is a classical model often used to describe the motion legged systems, most particularly hopping robots. The typical control variable of interest is the step length S, defined as the distance between two successive footholds, which is a function of the ballistic trajectory *and* the take-off and touch-down angles.

is the angle the leg forms with respect to the ground, as shown in Fig. 2.1. Additionally, k is the spring stiffness, m is the mass of the body, L_0 is the natural leg length, and L_a is the actuator's length. The leg is assumed to be massless and has no inertia. The positions x and y are defined as the horizontal and vertical coordinates of the mass. The flight dynamics for all hopping robots are purely ballistic and can be written as:

$$\ddot{y} = -g \tag{2.1}$$

$$\ddot{x} = 0 \tag{2.2}$$

The transition between flight and stance phase is called *touch-down* (TD), while the transition between stance and flight phase is called *take-off* (TO). The highest point of

the flight phase is called the *apex state*, $\mathcal{A}$. We can therefore characterize the apex state as the state during flight with $\dot{y} = 0$. Then, $\mathcal{A} = [y, \dot{x}]$. The space in the x-dimension traveled by the system in one step, i.e., the distance between consecutive footholds, is called the *step length* S, illustrated in Fig. 2.1. It is important to note that the step length is not simply a function of the ballistic path, but also the leg angle at the times of take-off and touch-down. Therefore, attempting to regulate step length by simply controlling the forward speed and height energy, as has been classically done [38], will likely exhibit errors.

The stance phase of the system occurs when the leg is in constant contact with the ground, beginning at touch-down and ending when the spring returns back to it's equilibrium position at take-off. The stance phase equations of motion can be generated using the Lagrangian method, starting from the coordinates for the mass as:

$$x = Lcos(\theta)$$

$$y = Lsin(\theta)$$

Next, the kinetic co-energy T^* and potential energy V can be expressed as:

$$T^* = \frac{m}{2}(\dot{\theta}^2 L^2 + \dot{L}^2)$$
$$V = \frac{k}{2}(L - L_0 - L_a)^2 + mgL\sin(\theta)$$

Resulting in the dynamics

$$\ddot{L} = -\frac{k}{m}(L - L_0 - L_a) - g\cos\theta + L\dot{\theta}^2 \tag{2.3}$$

$$\ddot{\theta} = 2\frac{\dot{L}\dot{\theta}}{L} - \frac{g}{L}\sin\theta \tag{2.4}$$

As previously discussed, the above dynamics are not analytically solvable due to the time-

varying leg length, however many recent approximations are present in the literature that we will make use of later in this dissertation.

The SLIP model is an idealized system: its strength lies in being a simple way to model hopping gaits in animals and humans. Several layers of complexity can be added to this simplified system to match real-world hardware more closely. In particular, a realistic model would include an unsprung mass at the foot/lower part of the leg, as well as inertia in both leg and body. Without mass in the leg, the SLIP model lacks any notion of energy loss due to impact at TD and TO. More realistic dynamics should also include damping and friction terms. A separate issue arises when considering the motion of the series-elastic actuator, which is typically modeled as an instantaneous impulse from an initial position to a desired one [33]. In the SLIP model $L_a(t)$ is the control variable and can typically be chosen freely: the input/output pair is (L_a, L). For more realistic implementations, however, $L_a(t)$ itself is typically driven by a motor torque, τ (or, equivalently, an input current u), thus the SEA position $L_a(t)$ should be considered as an additional degree of freedom, with its own dynamics including mass, frictional and damping terms, and physical limits.

2.2 The Compliant 3-link Robot

Not all hopping robots can be described as an evolution of the SLIP model. Therefore before proceeding to develop our more realistic SEA SLIP-based models, we briefly introduce a 3-link hopping system with a compliant leg where no SEA is used. All input terms are torques directly applied between links. This system has three state angles, one length state, and only two actuators, and is therefore highly underactuated. The essential model consists of three serial links. The first link is modeled as massless and contains a passive spring-damper element, as depicted in Fig. 2.2, while the other two

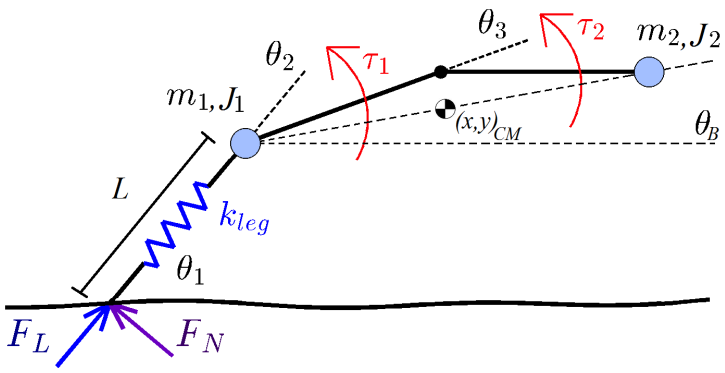


Figure 2.2: The compliant 3-link robot is an example of a hopper that is structured significantly different from SLIP. The system is underactuated, with no torque actuation between the first link and the ground. This system is also general enough to loosely represent the side view of a bounding quadruped, if an additional leg is added at the end of the link.

links have mass and inertia. Two actuators apply torques at the hip (θ_2) and spine (θ_3) joints. The stance phase dynamics can be computed using the Lagrangian method, with the state vector written as:

$$X = [\theta_1, \theta_2, \theta_3, L, \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{L}]^T$$

Assuming the foot makes contact with the terrain at the origin, the locations of the three body points during the stance phase are defined as follows using relative angle

coordinates:

$$
\begin{aligned}
x_1 &= Lcos(\theta_1) \\
y_1 &= Lsin(\theta_1) \\
x_h &= x_1 + l_1 cos(\theta_1 + \theta_2) \\
y_h &= y_1 + l_1 sin(\theta_1 + \theta_2) \\
x_2 &= x_h + l_2 cos(\theta_1 + \theta_2 + \theta_3) \\
y_2 &= y_h + l_2 sin(\theta_1 + \theta_2 + \theta_3)
\end{aligned}
$$

Where L is the time varying leg length, and l_1 and l_2 are constant linkage distances. Next, we define the kinetic co-energy T^* and potential energy V as

$$
\begin{aligned}
T^* &= 0.5(m_1(\dot{x}_1^2 + \dot{y}_1^2) + m_2(\dot{x}_2^2 + \dot{y}_2^2) + J_1\dot{\theta}_b^2 + J_2\dot{\theta}_f^2) \\
V &= m_1 g y_1 + m_2 g y_2 + \frac{k_{leg}}{2}(L - L_0)^2
\end{aligned}
$$

Where k_{leg} is the leg spring constant with natural length L_0, $\theta_b = \theta_1 + \theta_2$, $\theta_f = \theta_1 + \theta_2 + \theta_3$, and J_1 and J_2 represent combined link inertias. The resulting equations of motion can then be generated as:

$$
W(X, \tau_1, \tau_2) = \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \\ \ddot{L} \end{bmatrix} = M^{-1} \left(C + \begin{bmatrix} 0 \\ \tau_1 \\ \tau_2 \\ -b_k\dot{L} \end{bmatrix} \right) \tag{2.5}
$$

Where matrices M and C are both functions of X. Unlike SLIP-based hopping robots with body masses either on the hip or on a linkage above the hip joint, the variable tied to body stability is different for this system due to the large number of links. In this case,

the robot linkage must remain upright during operation, which means all three angles must be well behaved. However, similar to SLIP-based hoppers, we can generalize the control problem by constructing three auxiliary variables. These control variables are two CoM coordinates and a body angle, written as:

$$\begin{aligned} x_{cm} &= \frac{1}{m_1 + m_2}(m_1 x_1 + m_2 x_2) \\ y_{cm} &= \frac{1}{m_1 + m_2}(m_1 y_1 + m_2 y_2) \\ \theta_B &= tan^{-1}(\frac{y_2 - y_1}{x_2 - x_1}) \end{aligned} \tag{2.6}$$

Similar to SLIP, during flight the system's CoM follows ballistic dynamics as in (2.2). Control strategies for this system, and for regulating the CoM of hoppers in general, are presented in Sections III and VI. While an interesting system to study in terms of developing control strategies for more general compliant robots, since this system does not employ a thrust actuator the energy requirements for hopping gaits can be quite high, and ultimately unrealistic. Thus, the remaining hopping robot models presented in this chapter employ realistic means of series-elastic actuation.

2.3 Series-Elastic Actuated 1D Hoppers

The simplest type of hopping robot hardware one can build is, unsurprisingly, a vertical hopper. These robots must have an actuator mechanism to modify spring energy and allow for apex height regulation during vertical hopping. Although a simple problem in terms of how many control variables are involved (a "1D problem"), for real hardware implementations the coupling between spring compression and input motor current can be quite non-trivial, therefore we briefly discuss 1D hopper dynamics. It will also be shown later that for the simplified case of steady-state forward motion, control strategies derived

from 1D hopping systems can still be applicable. The robots considered in this work use SEA, and examples of such robots developed by the UCSB Robotics Lab group [32] can be seen in Fig. 2.3, along with a model schematic. Two vertical hopping prototypes were constructed in order to both test leg mechanisms and series-elastic actuators. As is the case with all hopping robots, the dynamics are divided into a stance and flight phase, with the flight phase dynamics being the trivial ballistic dynamics seen in (2.2), thus only the stance dynamics are discussed here.

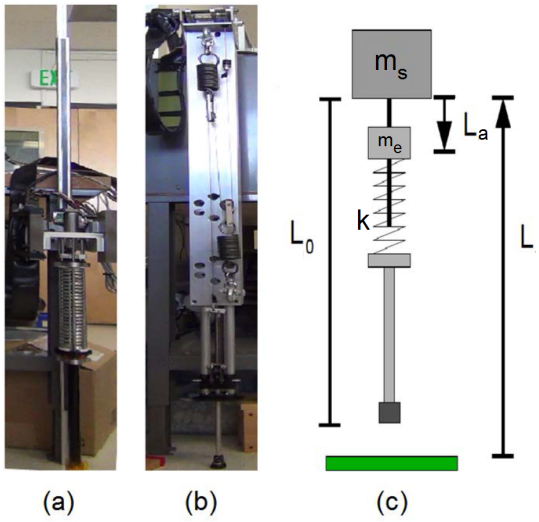


Figure 2.3: Above shows hopping robots (a) Hopper B, (b) Hopper C, and (c) schematic for 1D SEA hopper models, where L represents the length from the ground to the bottom of the leg with natural length L_0. The spring energy can be actively controlled via SEA position L_a, a state with real dynamics, that compresses the spring from the opposite end.

The first of these robots, the Hopper B, consists of a large spring coiled around the leg, precompressed via a metal plate driven by a ball screw which is attached to a geared motor. The Hopper B is considered a more or less traditional hopper design, where it is very easy to model how the robot is structured. The series-elastic actuator in this instance is a physical plate that moves to compress the spring, and therefore must be considered as a state of the system. We again apply the Lagrangian method in order

to derive the equations of motion for the system, with the target states L, the vertical position referenced from the hip of the robot, and L_a, the position of the SEA (metal plate). We begin by writing the kinetic co-energy and potential energy as:

$$\begin{aligned} T^* &= \frac{1}{2}(J_m + J_b)\dot{\theta}_m^2 + \frac{1}{2}m_p\dot{L}_a^2 + \frac{1}{2}m_B\dot{L}^2 \\ V &= m_B g L + \frac{k}{2}(L - L_a + L_0 + c)^2 \end{aligned} \tag{2.7}$$

Where θ_m, J_m, J_b, and m_p represent the motor angle, motor inertia, ball-screw inertia, and actuator plate mass respectively. We next eliminate θ_m as a variable by using the relationship of the ball screw velocity to linear plate velocity: $\dot{\theta}_m = -N\dot{L}_a$, and define the *effective mass* of the SEA as:

$$m_e = N^2(J_m + J_b) + m_p \tag{2.8}$$

Where N is the motor gear ratio. Using (2.7) and (2.8), the resulting dynamics of the system can be written as:

$$\begin{aligned} \ddot{L}_a &= \frac{1}{m_e}(K(L - L_a - L_0 - c) - b_1\dot{L}_a - \nu u_{leg}) - f_1 sign(\dot{L}_a) \\ \ddot{L} &= \frac{1}{m_B}(K(L_a - L + L_0 + c) - b_2\dot{L} - m_B g) - f_2 sign(\dot{L}) \\ \nu &= \frac{2\pi k_t}{l_b} \end{aligned} \tag{2.9}$$

The model consists of the variables $K, c, m_B, m_e, k_t, l_b, g,$ which represent respectively the spring constant, main spring pre-load, sprung mass, effective actuator mass, motor torque constant, ball-screw meters per revolution, and gravity constant. The coefficient ν converts the input current u_{leg} into input force to the SEA. Non-linear frictional effects are common in SEA systems, therefore we choose to include Coulomb friction terms f_1 and f_2 in addition to damping terms b_1 and b_2. Although simpler in construction, it

was experimentally determined that the ball-screw of the Hopper B was not well suited for the high forces experienced during decompression of the spring, and was prone to breaking.

The second vertical hopper constructed, the Hopper C, uses a network of static and sliding pulleys to actively change the system spring compression. This actuator and leg mechanism, schematic shown in Fig. 2.4, was ultimately determined to be superior to the Hopper B implementation due to being lighter, more robust to high velocities and forces resulting from rapid spring decompression, and most importantly having direct measurement of the robot leg state L via an encoder on the leg pulley. The SEA and leg mechanism from Hopper C was therefore chosen to be used on the 2D robot FRANK. Although more complex and quite mechanically different, the system dynamics can still

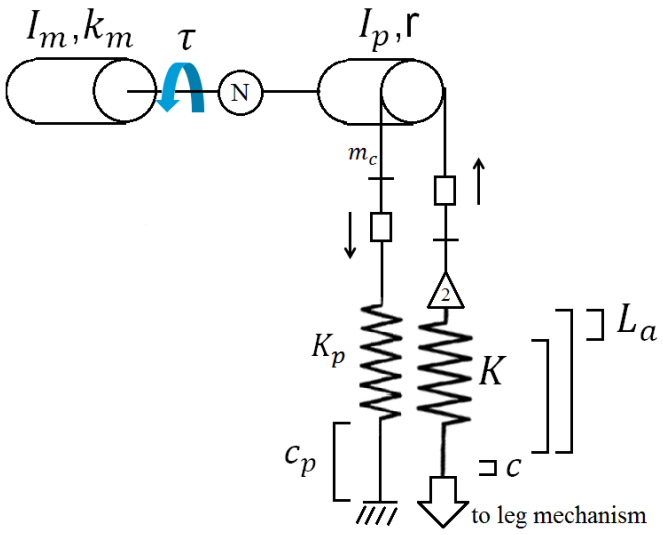


Figure 2.4: Schematic for the cable driven series-elastic actuation of the Hopper C. In addition to being overall lighter, the cable driven approach was experimentally determined to be more robust to the high spring forces, and less prone to being damaged as the Hopper B. This SEA implementation is the same used in the 2D robot FRANK.

be represented in nearly an identical form as those of the Hopper B. As was the case with the Hopper B, the states we use to model the system dynamics are L and L_a, and

the procedure for generating the equations of motion is exactly the same. In this case, the effective mass and motor force constant are written as:

$$m_e = 4m_c + \frac{4I_p + 4N^2 I_m}{r_p^2}$$
$$\nu = \frac{0.5Nk_t}{r}$$

Where I_p, I_m, r_p, m_c, k_t, and N are the pulley inertia, motor inertia, pulley radius, cable mass, motor torque constant, and gear ratio respectively. The dynamics for the Hopper C can be expressed as:

$$\begin{aligned}
\ddot{L} &= \frac{1}{m_B}(K(L_a - L + L_0 + c) - b_2\dot{L} - m_B g) - f_2 \text{sgn}(\dot{L}) \\
\ddot{L}_a &= \frac{1}{m_e}(K(L - L_a - L_0 + c) - b_1\dot{L}_a + \nu_S) - f_1 \text{sgn}(\dot{L}_a) \\
\nu_S &= k_p(2L_a - c_p) - \nu u_{leg}
\end{aligned} \tag{2.10}$$

In this case the model consists of two additional parameters: k_p and c_p, which represent the spring constant and pre-load of an additional, smaller spring necessary to tension the cables of the SEA. Using a cable driven hopper has the advantage of having an overall lighter design, not relying on a heavy ball-screw driven mechanism. However, the cost of using a complex network of cables and pulleys is increased frictional effects in both the leg and the actuator. However, system identification experiments have shown that the Hopper C (and FRANK) can still be reasonably represented using Coulombic friction. Details regarding system identification can be seen in Appendix A. Another important note here is that when realistic SEA dynamics are considered, any approximations that rely on the leg length being able to accelerate arbitrarily fast or even change instantaneously are not valid, as illustrated in Fig. 2.5. Here we see clearly that modeling the SEA as a constant impulse is not at all representative of hardware implementations.

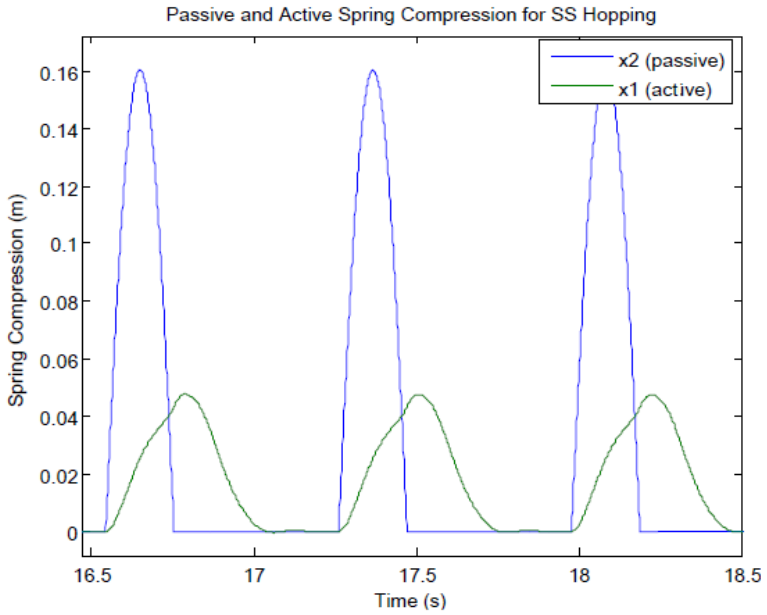


Figure 2.5: Compression cycles during stance phase for steady-state operation of the Hopper C hardware. In the data above, $x_1 = L_a$ and $x_2 = max((L_0 - L), 0)$, which represents spring compression. Note that actuator dynamics happen at a time scale similar to that of hopper dynamics and cannot be well-approximated as instantaneous.

2.4 Series-Elastic Actuated 2D Hoppers

This Section introduces a SEA 2D hopping robot. This robot has all the real features associated with the realistic hopping robot FRANK, with the exception of body angle dynamics. The SEA 2D hopper is essentially a more realistic version of the active SLIP, in that it can be realized in hardware by mechanically locking body rotation on the boom side of FRANK. Unlike SLIP however, the system exhibits SEA dynamics, leg mass and inertia, and potentially a torque control input at the hip. Note that this model does not exhibit body dynamics and therefore is still not quite sufficient for real-word operation, as these robots will always have a body on which electronics is typically mounted that must be stabilized, however the SEA 2D hopper is a useful system to study in order to understand how to extend SLIP-based methods to realistic SEA implementations. The

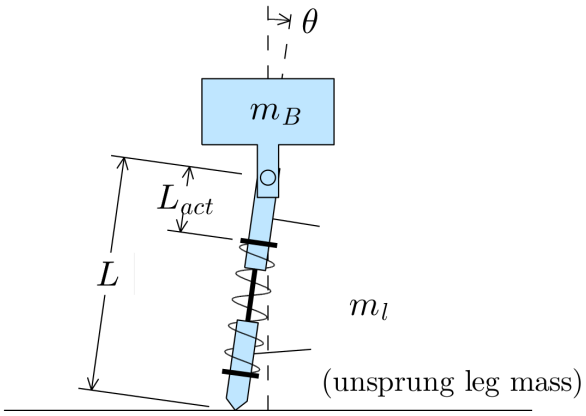


Figure 2.6: Schematic for the 2D SEA Hopper. This robot is essentially a more realistic version of the Active SLIP Model, and can be realized in hardware by mechanically locking the body of the robot FRANK.

state X of the system during stance is given as

$$X = [\theta, L, L_a, \dot{\theta}, \dot{L}, \dot{L}_a]^T$$

where θ represents the leg angle, L represents the leg length, and L_a represents the position of the SEA. The kinematic structure of the robot is given as:

$$\begin{aligned} x_{leg} &= x_{foot} + \frac{L}{2}\sin\theta \\ y_{leg} &= y_{foot} + \frac{L}{2}\cos\theta \\ x_{body} &= x_{foot} + L\sin\theta \\ y_{body} &= y_{foot} + L\cos\theta + l_1 \end{aligned} \tag{2.11}$$

where l_1 represents the distance between the hip joint and the body CoM, constant in the body-locked case, and x_{foot} and y_{foot} are constant during stance, and have ballistic dynamics seen in (2.2) during flight. Since the robot has leg mass, the CoM of the robot

is not exactly at the body mass CoM, and can be expressed as:

$$\begin{aligned} x_{CoM} &= \frac{1}{m_l + m_s}(m_l x_{leg} + m_s x_{body}) \\ y_{CoM} &= \frac{1}{m_l + m_s}(m_l y_{leg} + m_s y_{body}) \end{aligned} \tag{2.12}$$

where m_l represents the combined (unsprung) leg and foot mass, and x_{leg}, y_{leg} represent the position of the CoM of the leg. Similarly, x_{body}, y_{body} represent the position of the CoM of the (sprung) body mass m_B. In order to define the dynamics via the Lagrangian method, we write the kinetic co-energy and potential energy as:

$$\begin{aligned} T^* &= \frac{m_s}{2}(\dot{x}_{body}^2 + \dot{y}_{body}^2) + \frac{m_l}{2}(\dot{x}_{leg}^2 + \dot{y}_{leg}^2) + \frac{J_l}{2}\dot{\theta}^2 + \frac{m_e}{2}\dot{L}_a^2 \\ V &= g(m_s y_{body} + m_l y_{leg}) + \frac{K}{2}(L_0 - L + L_a + c)^2 + \frac{k_p}{2}(c_p - 2L_a)^2 \end{aligned} \tag{2.13}$$

Where the new parameters m_e, J_l, and m_l represent the effective SEA mass, leg inertia, and leg mass respectively. Using (2.13), the resulting dynamics of the stance phase are

$$\begin{bmatrix} \ddot{\theta} \\ \ddot{L} \\ \ddot{L}_a \end{bmatrix} = M^{-1} \left(C + \begin{bmatrix} -NK_t u_{hip} \\ -b_2\dot{L} - f_2 \text{sgn}(\dot{L}) \\ -\nu u_{leg} - b_1\dot{L}_a - f_1 \text{sgn}(\dot{L}_a) \end{bmatrix} \right) \tag{2.14}$$

where non-linear matrices M and C are both functions of X. The SEA actuator variable u_{leg} outputs current to drive L_a with motor torque constant K_t and gear ratio N.

During the discrete events touch-down and take-off, additional energy in the system is lost due to the unsprung leg mass. At touch-down, the foot of the robot impacts the ground causing the unsprung portion of the leg's velocity to be dissipated. The

instantaneous effect on the CoM velocity can be written as

$$\begin{aligned} \dot{x}^{+}_{CoM} &= \dot{x}^{-}_{CoM} - \frac{m_l}{2(m_B + m_l)} \dot{x}^{-}_{foot} \\ \dot{y}^{+}_{CoM} &= \dot{y}^{-}_{CoM} - \frac{m_l}{2(m_B + m_l)} \dot{y}^{-}_{foot}. \end{aligned} \tag{2.15}$$

At take-off, the moving frame of the body impacts the leg assembly causing the system to lift off the ground. As previously described for Raibert hoppers [38], the radial change in velocity can be found by conservation of linear momentum and results in additional loss on the CoM velocity as

$$\begin{aligned} \dot{r}^{+} &= \frac{m_B}{m_l + m_B} \dot{L}^{-} + \frac{m_l}{2(m_l + m_B)} L_0 \dot{\theta} \\ \dot{x}^{+}_{foot} &= \dot{r}^{+} \sin\theta, \quad \dot{y}^{+}_{foot} = \dot{r}^{+} \cos\theta \end{aligned} \tag{2.16}$$

As we see with the impact dynamics above, we lose energy at each step proportional to the amount of unsprung mass in the system, that any actuation policy we develop must account for. It will be shown in Section V that SLIP-based algorithms can be implemented on this robot, and precision step length control can be achieved.

2.5 The Hopping Robot FRANK

The hopping robot FRANK (**F**RANK: **R**obot **A**cronym **N**ot **K**nown), shown in Fig. 2.7, is a Series-Elastic Actuated 2D Hopper with an underactuated body, similar to the classic Raibert hoppers [38]. The robot is connected to a large carbon fiber boom, and an optional mechanical lock was developed on the boom side to limit body rotation. When FRANK's body is mechanically locked on the boom side, the model is exactly the Series-Elastic Actuated 2D Hopper from the previous Section, therefore this Section focuses solely on the body-unlocked implementation. FRANK has two actuators and

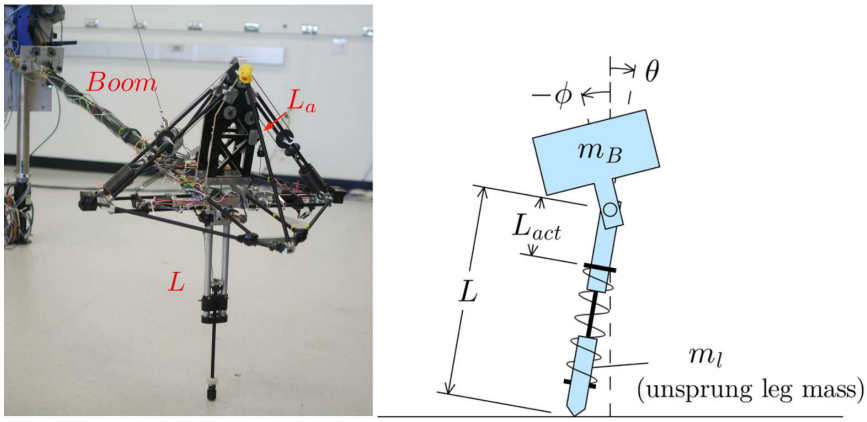


Figure 2.7: The robot FRANK is a low weight SLIP-inspired underactuated hopper. The robot consists of a series elastic actuator (SEA) to inject energy into the system, a planar leg angle actuator to control the forward touch-down angle (θ), and an underactuated body angle (ϕ) which rotates during operation and must be kept upright to maintain stable motions. The center of mass (CoM) of the body does not correspond to the hip joint, giving way to non-trivial coupling dynamics between ϕ and θ.

four dynamic states, and is therefore highly underactuated. Furthermore, the body CoM does not coincide with the robot's hip joint, giving rise to non-linear coupling dynamics between the leg and the body. Note that our angle and state conventions largely parallel Raiberts original work [38].

Similar to SLIP, this system can be described as having a flight phase and a stance phase. During the flight phase the system follows ballistic dynamics in (2.2). The states X of the system during stance are given as

$$X = [\theta, \phi, L, L_a, \dot{\theta}, \dot{\phi}, \dot{L}, \dot{L}_a]^T \tag{2.17}$$

where θ represents the leg angle, ϕ represents the underactuated body angle, L represents the leg length, and L_a represents the position of the SEA. The robot is kinematically structured identically to the SEA 2D Hopper introduced in the previous Section, with the only difference being the location of the body mass is now dependent on the body

angle as:

$$\begin{aligned} x_{leg} &= x_{foot} + \frac{L}{2}\sin\theta \\ y_{leg} &= y_{foot} + \frac{L}{2}\cos\theta \\ x_{body} &= x_{foot} + L\sin\theta + l_1\sin(\phi) \\ y_{body} &= y_{foot} + L\cos\theta + l_1\cos(\phi) \end{aligned} \tag{2.18}$$

The equation governing the location of system CoM can be seen in (2.12). To proceed and define the stance dynamics via the Lagrangian method, we write the kinetic co-energy and potential energy as:

$$\begin{aligned} T^* &= \frac{m_s}{2}(\dot{x}_{body}^2 + \dot{y}_{body}^2) + \frac{m_l}{2}(\dot{x}_{leg}^2 + \dot{y}_{leg}^2) + \frac{J_l}{2}\dot{\theta}^2 + \frac{m_e}{2}\dot{L}_a^2 + \frac{J}{2}\dot{\phi}^2 \\ V &= g(m_s y_{body} + m_l y_{leg}) + \frac{K}{2}(L_0 - L + L_a + c)^2 + \frac{k_p}{2}(c_p - 2L_a)^2 \end{aligned} \tag{2.19}$$

Where the new term J represents the system body inertia. The dynamics of the stance phase are finally generated as:

$$\begin{bmatrix} \ddot{\theta} \\ \ddot{\phi} \\ \ddot{L} \\ \ddot{L}_a \end{bmatrix} = M^{-1}\left(C + \begin{bmatrix} -NK_t u_{hip} \\ NK_t u_{hip} \\ -b_2\dot{L} - f_2 sgn(\dot{L}) \\ -\nu u_{leg} - b_1\dot{L}_a - f_1 sgn(\dot{L}_a) \end{bmatrix} \right) \tag{2.20}$$

where matrices M and C are both functions of X. Note that while in terms of the Lagrangian derivation, the only notable differences in the construction compared to the body locked case is the addition of the inertial term in the kinetic co-energy and the resulting counter-torque applied to the body by the hip actuator. While this may seem a simple addition, the resulting matrices $M(X)$ and $C(X)$ are *considerably* more complex

in this case, resulting in quite a difficult system to stabilize during nominal 2D hopping. The specifics of the body-leg coupling dynamics are discussed further in Chapter V.

The system impact dynamics are now also a function of the body angle ϕ, but when written in terms of the system CoM are exactly the same as those seen in Equations (2.16) and (2.15). All simulation results presented in this paper use realistic model parameters obtained via system identification of the robot FRANK, the details of which are discussed in Appendix A, and can be seen in Table 2.1. The robot is tethered to a large boom on which all hardware is mounted. The leg mechanism used by FRANK is exactly the leg used in the Hopper C, seen in Fig. 2.2. FRANK has motor encoders for θ and the motor that drives the SEA position L_a, along with boom encoders to measure ϕ, the vertical angle ψ, and the horizontal angle α_B, allowing real-time estimation of both forward and vertical position and velocity. As with the Hopper C, FRANK has an additional encoder on the leg pulley, allowing direct measurement of the leg state L in stance. Additionally, the SEA coupling between the motor and leg mechanism is the same as shown in Fig. 2.4. The specifics of how the leg itself is coupled to the system's two main springs, along with the electronics implemented for control are shown in Fig. 2.8.

All computation is done on board the hardware using a VersaLogic Tiger (VL-EPM-24) board, with communication between the host and target accomplished via Simulink Real Time. A Maxon motor controller is used in current control mode in order to apply the correct voltage to the motors given a desired current command by our controllers. Although the robot FRANK has an additional actuated angle state ($\theta_\perp$), we omit it from the 2D dynamics as it is only used to prevent the leg from slipping in the perpendicular direction during operation, the details of which are provided in our experimental results presented in Section IV. During the flight phase the robot FRANK in actual hardware still follows approximate ballistic dynamics, with some slight modifications due to the boom geometry. Specifics regarding these ballistic-like dynamics and corresponding system

identification data can be found in Appendix A.

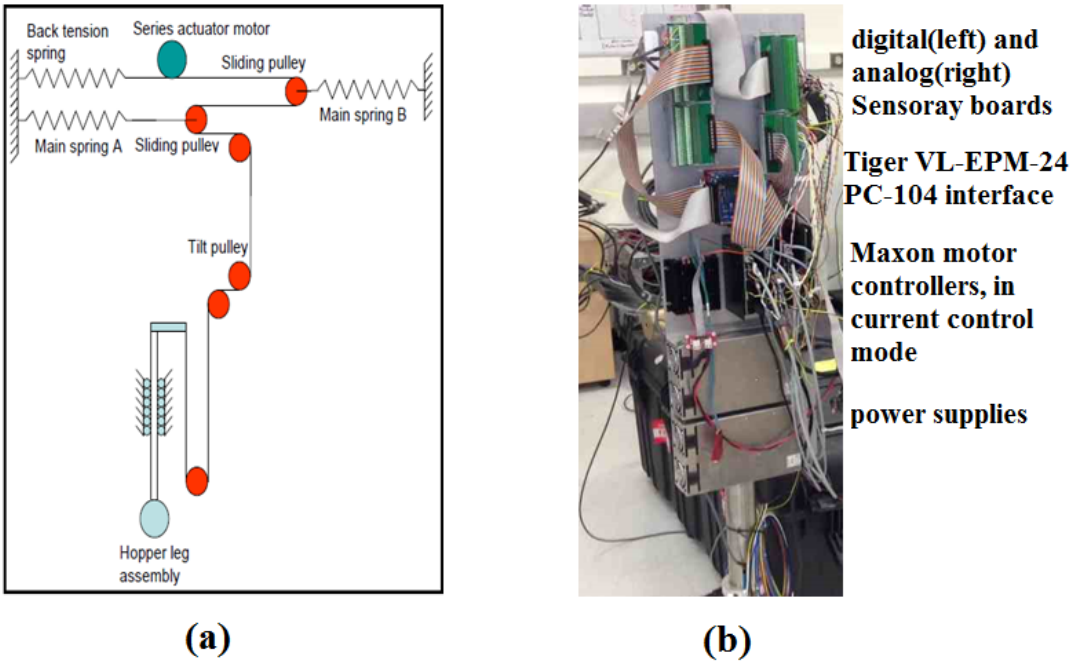


Figure 2.8: The robot FRANK uses a cable driven design for both the SEA and leg mechanism, seen above in (a). The leg mechanism used is exactly the same used in the 1D prototype Hopper C. All electronics (b) are mounted on the boom side, and communication is sent via an Ethernet connection from target to a host computer running Simulink.

It will be shown in Section V that the same SLIP-based algorithms implemented for the body-locked SEA 2D Hopper can also be implemented with the body unlocked, with some slight modifications and use of the torque input at the hip.

m_B	7.59 kg	Sprung (body) mass
m_l	0.548 kg	Unsprung (leg) mass
m_e	7.11 kg	Effective actuator mass
J_l	0.015 $kg\ m^2$	Leg inertia
J_B	0.227 $kg\ m^2$	Body inertia
l_0	54.3 cm	Natural leg length
l_1	16.2 cm	Body CoM dist. from hip
K	2,389 N/m	Main spring constant
c	0.002 m	Main spring pre-load
k_p	245.18 N/m	Tension spring constant
c	0.114 m	Tension spring pre-load
K_t	0.0369 Nm/A	Motor torque constant
N	66	Motor gear ratio
ν	59.1 N/A	SEA actuator constant
b_1	1.74 Ns/m	SEA linear friction
f_1	0.91 N	SEA coulomb friction
b_2	1.08 Ns/m	Leg linear friction
f_2	0.53 N	Leg coulomb friction
u_{max}	20 A	Maximum current input

Table 2.1: Model parameters for the robot FRANK

Chapter 3

High Order Partial Feedback Linearization on Hoppers

This chapter introduces PFL-based control laws applicable to hopping robots during the stance phase. There are two core reasons to develop such methods. The first is simply to have an accurate method of implementing feedback control continuously on the system while in contact with the ground. Fig. 3.1 shows simulation results for implementing PID-based apex height control on a model of FRANK. Even for this often considered "simple" case of regulating apex heights during vertical hopping, simplistic controllers that do not incorporate accurate dynamical models often have very sub-par performance, as they typically rely on discrete energy-based estimations and/or feedback of apex state measurements only. The control strategy used in Fig. 3.1 follows classical techniques [8] where the SEA input is set as a constant thrust at every hop calculated using feedback from the previous apex height and a reference command. The same strategy is then attempted in forward motion with the touch-down angle command also set using a discrete PID controller as in [8]. Such control methods may provide acceptable results for steady-state solutions, but lack the ability to track fast changing commands

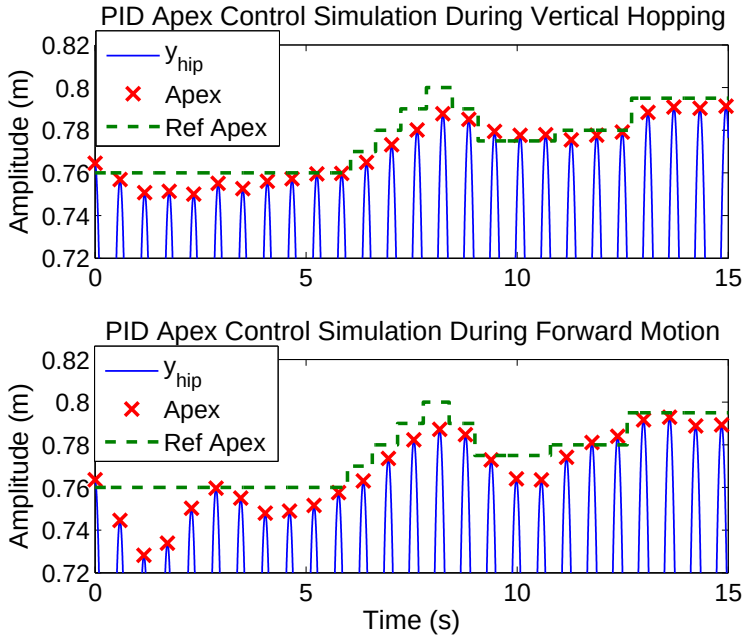


Figure 3.1: This figure shows that even for the simple case of regulating apex heights (the "1D problem") on a realistic hopper model, using simplistic control methods will not result in good performance. The top plot shows that even in vertical hopping the performance leaves much to be desired, and the bottom plot illustrates how performance will degrade further if such a strategy is attempted during forward motion.

required for foothold selection. Even in the case of analytically solving the 1D system for feed-forward commands, a controller solely based on feed-forward information will often result in significant errors from model inaccuracies. More details and comparative results on these control methods are provided in Section IV.

The second core motivation follows from the fact that analytical solutions for more general 2D hopping systems are difficult to calculate in the presence of complex actuator dynamics. Therefore, generating an accurate feed-forward command is quite difficult (or impossible) in the presence of analytically unsolvable governing stance phase equations. We would like to use u_{leg} to directly control L (not L_a) or other positional auxiliary variables, and since our system has potentially both nonlinear frictional effects and leg angle dynamics we would like to negate, we propose to use feedback linearization to

accomplish both goals. This chapter will review the fundamentals of PFL for mechanical and compliant systems, and provide the relevant PFL control laws to be used on our hopping robot models. We acknowledge that practical application of PFL on real systems typically requires an excellent model to accuracy cancel the non-linearities of the system. However, the stance phase dynamics of hoppers have the advantage of mostly being dominated by spring forces, therefore while it is critical to accurately determine the spring parameters, later results will suggest that a perfect model for some of the more difficult to identify model properties such as frictional coefficients is not necessarily required for reasonable results.

This chapter is focused solely on constructing the feedback laws. Subsequent chapters will provide implementation examples, trajectory generation, and algorithms we use to achieve various goals such as apex regulation and step length control.

3.1 Review of PFL for Mechanical Systems

In this Section the fundamentals of partial feedback linearization are reviewed. Controlling a system with PFL is essentially applying feedback linearization on some of the states, or auxiliary variables of the states, of an underactuated system. It was shown by Spong [58, 59, 60] that for an n degree of freedom system with m actuated states, for most mechanical systems in fact m of the equations of motions can be linearized whether or not they are directly attached to the actuators. As was shown by Spong, for a mechanical system with state vector $x \in \mathcal{R}^n$, we partition the state-space in terms of m actuated states x_2 and k underactuated states x_1 as:

$$M_{11}\ddot{x}_1 + M_{12}\ddot{x}_2 + \epsilon_1 = 0 \tag{3.1}$$

$$M_{21}\ddot{x}_1 + M_{22}\ddot{x}_2 + \epsilon_2 = \tau \tag{3.2}$$

where components M_{ij} make up some larger system matrix M, and along with ϵ_1 and ϵ_2 are nonlinear functions of the state x. The key idea here is that since m of our states are actuated, we should be able to control m out of n variables. If these control variables are the actuated states of the system, this is called *collocated linearization*, and the control law is written as:

$$\begin{aligned} \tau &= \bar{M}_{22} v + \bar{\epsilon_2} \\ \bar{\epsilon_2} &= \epsilon_2 - M_{21} M_{11}^{-1} \epsilon_1 \\ \bar{M}_{22} &= T^T M T \end{aligned} \tag{3.3}$$

with

$$T = \begin{bmatrix} -M_{11}^{-1} M_{12} \\ I_{mxm} \end{bmatrix}$$

The described controller cancels the natural dynamics of the system, and supplants them with those given by v, which are supplied by the control designer and are typically feedback controllers to drive the error of the controlled portion of the linearized system to zero.

Similarly, the case when the control variables are not the actuated states is deemed *non-collocated linearization*. This type of linearization is only possible for a set of output variables z when the mapping from z to the actuated states x_2 is *strongly inertially coupled*, which is a controllability condition. Essentially, if one substitutes the system dynamics in (3.2) into the dynamics of the desired auxiliary variables, the input term τ must explicitly appear in each component of the resulting auxiliary dynamics. The reader is refereed to [59] for more details and references regarding this condition. For the case of applying non-collocated linearization on the underactuated states x_2, the control

law is written as:

$$\begin{aligned} \tau &= \tilde{M}_{21} v + \tilde{\epsilon}_2 \\ \tilde{\epsilon}_2 &= \epsilon_2 - M_{22} \hat{M}_{12}^{-1} \epsilon_1 \\ \tilde{M}_{21} &= M_{21} - M_{22} \hat{M}_{12}^{-1} M_{11} \\ \hat{M}_{12} &= M_{12}^T (M_{12} M_{12}^T)^{-1} \end{aligned} \tag{3.4}$$

where again the natural dynamics are supplanted with a function v. Typically, we generate the dynamics for the desired error function as:

$$v = K_p(r_z - z) + K_d(\dot{r}_z - \dot{z})$$

where r_z and $\dot{r}_z$ are reference trajectories and z represents the PFL control variables, which were x_2 and x_1 in the provided examples of (3.3) and (3.4) respectively. Note that K_p and K_d determine the pole locations for the linearized portion of the system dynamics, and must be chosen properly to ensure the error dynamics decay exponentially to zero. The unlinearized portion of the system's dynamics, deemed the *zero dynamics* when this exponential convergence for the error of the linearized portion has occurred, must also remain well behaved. Designing trajectories for the PFL control variables (i.e., r_z, $\dot{r}_z$) such that these dynamics remain well behaved is the most challenging aspect of using PFL to control a system.

In regards to hopping robots, we typically have two actuators and three to four states, depending on the hopper model used. Thus, when applying PFL to the system we expect to be able to control at least two out of the three/four states (or auxiliary variables) of the system. In this dissertation we investigate two methods: applying PFL on the leg state on the robot, and applying PFL directly on the CoM x and y locations as auxiliary

variables.

3.2 PFL for Compliant Systems

Implementing feedback linearization for hoppers, and compliant systems in general, has the additional challenge of requiring all feedback loops to be generated at a higher order than the typical mechanical system. In general, for a system of the form:

$$\dot{x} = f(x) + g(x)u$$
$$y = h(x)$$

In order to use u to cancel nonlinearities in y and supplant the dynamics with some desired error function, we must consider the mapping from u to y. For a system with *relative degree* n, taking n time derivatives of the output yields:

$$\begin{aligned} y &= h(x) \\ \dot{y} &= \frac{dh(x)}{dx} f(x) \\ &\vdots \\ \frac{d^{n-1}y}{dt^{n-1}} &= \frac{d^{n-1}h(x)}{dx^{n-1}} f^{n-1}(x) \\ \frac{d^n y}{dt^n} &= \frac{d^n h(x)}{dx^n} f^n(x) + \frac{d(\frac{d^{n-1}h(x)}{dx^{n-1}} f^{n-1}(x))}{dx} g(x)u \end{aligned}$$

In other words, the *relative degree* for any input-output pair is the number of times the output function must be differentiated for the input term to appear and have instantaneous effect. For many mechanical systems, the relative degree is simply two, as is the

case for the classical PFL examples on the Acrobot and Pendubot presented by Spong [60]. For hopping robots, if we attempt to use our two actuators, a SEA and hip torque,

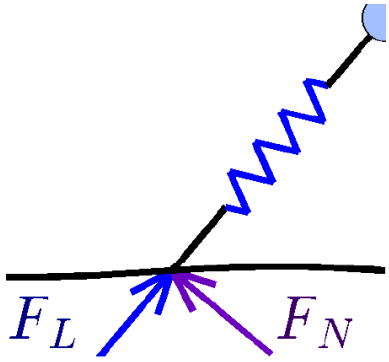


Figure 3.2: Using PFL to control a hopper typically means we seek to regulate both components of the ground contact force (F_L and F_N) independently, to in turn control the center of mass or some other axillary variable. However, because F_L is instantaneously set by the spring force, we must build the PFL loop about relative degree 2 or higher to implement this approach.

or in the case of the 3-link hopper two link torques, it is important to note that for a standard acceleration-based PFL (i.e. relative degree 2), the construction will fail if the system has a compliant leg, as all hoppers do, as shown in Fig. 3.2. This is because along the direction of the leg, the radial force is always set instantaneously by the spring or spring damper element as $F_L = -k_{leg}\Delta L - b_k\dot{L}$, such that we can only instantaneously effect one component of the system acceleration, not both. Thus, for hopping robots, we always expect the relative degree of any successful feedback linearization we use to necessarily be higher than 2, and will therefore be what we define as *high order partial feedback linearization* (HOPFL).

3.3 HOPFL on SEA Hopper Leg State

This Section introduces a feedback method for enforcing trajectories directly on the leg length L of SEA Hoppers. This is a powerful tool as it will later be shown that by

ensuring the trajectory of the leg length follows some nominal reference, approximations can be made to predict the underactuated leg angle dynamics, as the instantaneous acceleration of the leg angle depends only on the leg state L and not L_a. The leg length L is a state of the robot FRANK, however, as we can clearly see in the dynamics for the leg and SEA coupling in (2.10), the actuator input u_{leg} does not instantaneously effect the acceleration of the leg state. In this case, we must take two additional derivatives for our control variable to directly affect L, and therefore the relative degree of our system with L as the chosen output is 4.

We begin by considering the dynamics of the robot FRANK in (2.20). We will develop the HOPFL control law specifically for this model, but note that the same construction can be used for the SEA 2D Hopper (i.e. FRANK with locked body) in (2.14) with $\phi = 0$, and also for the 1D SEA Hopper C in (2.10) with both $\phi = 0$ and $\theta = 0$. We approximate the non-differentiable Coulombic terms with a scaled arctan function, which is quite accurate for scaling constant μ sufficiently small, and generate the approximate dynamics as

$$W = M^{-1}\left(C + \begin{bmatrix} -NK_t u_{hip} \\ NK_t u_{hip} \\ -b_2\dot{L} - \frac{2f_2}{pi} atan(\frac{\dot{L}}{\mu}) \\ -\nu u_{leg} - b_1\dot{L}_a - \frac{2f_1}{pi} atan(\frac{\dot{L}_a}{\mu}) \end{bmatrix}\right) \tag{3.5}$$

We proceed by taking two derivatives of the state acceleration equations in (3.5) to obtain the dynamics for the jounce of the system as

$$\Lambda(X, u_{leg}, u_{hip}, \dot{u}_{hip}, \ddot{u}_{hip}) = \frac{d^2}{dt^2} W \tag{3.6}$$

where we must also allow for the control input at the hip u_{hip} to have time-varying characteristics. We lastly substitute (3.6) in (3.5) and extract the component of Λ corresponding

to the leg-length state to form our control variable as

$$\dddot{L} = \gamma_L u_{leg} + \beta_L u_{hip} + \eta_L \dot{u}_{hip} + \alpha_L \ddot{u}_{hip} + \epsilon_L \tag{3.7}$$

where all coefficients are functions of the state X. In order to fully define the control law for L, we must calculate u_{hip}, $\dot{u}_{hip}$, and $\ddot{u}_{hip}$, which are functions of the hip controller used, and will be defined for each control implementation we present later in chapter V. When we take two derivatives of u_{hip}, however, state feedback terms present will typically cause the term $\ddot{L}_{act}$ to appear and therefore the component of $\ddot{u}_{hip}$ due to u_{leg} must be separated and used to generate new effective coefficients as

$$\begin{aligned} \ddot{u}_{hip} &= \delta_1 + \delta_2 u_{leg} \\ \tilde{\gamma}_L &= \gamma_L + \alpha_L \delta_2 \end{aligned} \tag{3.8}$$

Finally, using equations (3.7) and (3.8) we define our control law as

$$u_{leg} = \frac{1}{\tilde{\gamma}_L}(-\epsilon_L - \beta_L u_{hip} - \eta_L \dot{u}_{hip} - \alpha_L \delta_1 + v_L) \tag{3.9}$$

In other words, u_{leg} cancels the natural dynamics and forces any errors in L to decay via linear, fourth-order dynamics that we set through v_L. Specifically, we choose

$$\begin{aligned} v_L &= v_1 + v_2 \\ v_1 &= K_p(L_{ref} - L) + K_d(\dot{L}_{ref} - \dot{L}) \\ v_2 &= K_{dd}(\ddot{L}_{ref} - \ddot{L}) + K_{ddd}(\dddot{L}_{ref} - \dddot{L}) \end{aligned} \tag{3.10}$$

which requires four total poles in the closed-loop dynamics of $L(t)$. One option for setting the controller gains is to first select a dominant pole-pair with natural frequency ω_n and damping ratio ζ, and then set a significantly faster decay rate for the two remaining,

real-valued poles, p_3 and p_4. For a chosen set of w_n, ζ, p_3, p_4, the gains are:

$$\begin{aligned} K_p &= p_3 p_4 w_n^2 \\ K_d &= (p_3 + p_4) w_n^2 + 2 p_3 p_4 \zeta w_n \\ K_{dd} &= w_n^2 + 2\zeta(p_3 + p_4) w_n + p_3 p_4 \\ K_{ddd} &= p_3 + p_4 + 2\zeta w_n \end{aligned} \tag{3.11}$$

Additionally, we require both references and estimates of the acceleration and jerk of the system, which can be calculated using analytical computations once u_{hip} has been defined. Since the leg state is a position on the actual robot, this control construction can be loosely considered a type of collocated linearization, and is relatively easy to realize in hardware assuming a direct measurement for the leg length exists, as is the case for the robot FRANK. It will be shown in chapter V that the resulting trajectories of the underactuated variables ϕ and θ can be determined using SLIP-based techniques along with this HOPFL construction.

Note that in general, it is also possible to construct an equivalent PFL controller to regulate L_a directly. However, in this case we would not have feedback directly on L, and therefore cannot make approximations that depend on $L(t)$ converging to some nominal reference trajectory, which is critical for implementation of SLIP-based approximations and will be shown in chapter V.

3.4 HOPFL on Hopper CoM

For some hopping systems, it is advantageous to control the system CoM directly, as opposed to regulating the leg state and developing suitable reachability maps. Building the control law directly on the CoM has the advantage of directly regulating the take-off

velocities, however, comes at the additional challenge of being non-collocated and more difficult to implement. This HOPFL construction is also general enough to work on a variety of hopper models, and is therefore shown for both FRANK and the Compliant 3-link model. It will be shown later, however, that there is little reason to implement such a method on FRANK, where using HOPFL on the leg state is much better suited towards applying SLIP-based techniques. Applying this control framework towards step length control is discussed in chapter VI.

3.4.1 PFL Construction for FRANK

The control variables in this case are the CoM x and y locations of FRANK, seen in (2.12). Due to the system's series elastic element, constructing the acceleration of our control variables only yields one available input term. This can be verified by observing Equation (2.20) and noting the acceleration of L, θ, and ϕ are not instantaneously affected by u_{leg}. Therefore, when we take derivatives to define $\ddot{x}_{cm}$, $\ddot{y}_{cm}$ and substitute Equation (2.20) for the state acceleration variables we obtain

$$\begin{aligned} \ddot{x}_{cm} &= \beta_x(X)u_{hip} + \epsilon_x(X) \\ \ddot{y}_{cm} &= \beta_y(X)u_{hip} + \epsilon_y(X), \end{aligned} \tag{3.12}$$

where β, and ϵ are functions of the state X. The fourth state variable L_a, the active spring compression, appears in the dynamics for the acceleration of the remaining three state variables. Therefore, two derivatives must be taken for u_{leg} to instantaneously affect the CoM, thus as before we must build the linearization with relative degree 4. To cope mathematically with the Coulombic friction terms in the model, we again approximate

these terms via a scaled arctangent function as

$$W = M^{-1}\left(C + \begin{bmatrix} -NK_t u_{hip} \\ NK_t u_{hip} \\ -b_2\dot{L} - \frac{2f_2}{pi} atan(\frac{\dot{L}}{\mu}) \\ -\nu u_{leg} - b_1\dot{L}_a - \frac{2f_1}{pi} atan(\frac{\dot{L}_a}{\mu}) \end{bmatrix}\right) \tag{3.13}$$

which is a reasonable approximation for μ sufficiently small. We proceed by taking two derivatives of the state acceleration equations in (3.13) to obtain dynamics for the jounce of the system as

$$J(X, u_{leg}, u_{hip}, \dot{u}_{hip}, \ddot{u}_{hip}) \approx \frac{d}{dt}\frac{d}{dt}W \tag{3.14}$$

Next, we take additional time derivatives of Equation (2.12) to define $\dddot{y}_{cm}$, and by substituting in Equations (2.20) and (3.14) we form our control variables as

$$\begin{aligned} \ddot{x}_{cm} &= \beta_x u_{hip} + \epsilon_x \\ \dddot{y}_{cm} &= \gamma_y u_{leg} + \beta_y u_{hip} + \eta_y \dot{u}_{hip} + \alpha_y \ddot{u}_{hip} + \epsilon_y \end{aligned} \tag{3.15}$$

where the equation for x_{cm} is unchanged and γ, η, and α are higher order terms that are also a function of only the state X. We define the control law for x_{cm}, which is trivial as only one actuator has effect on the acceleration, as

$$u_{hip} = \frac{1}{\beta_x}(-\epsilon_x + v_x) \tag{3.16}$$

where v_x is calculated via PD feedback, to drive the system to desired references as

$$v_x = K_p(x_{ref} - x_{cm}) + K_d(\dot{x}_{ref} - \dot{x}_{cm}) \tag{3.17}$$

In order to define the control law for y_{cm} we must determine the two input derivative terms, which can be calculated as follows. We construct the first derivative of u_{hip} as

$$\dot{u}_{hip} = \frac{1}{\beta_x}(-\dot{\epsilon}_x + \dot{v}_x) - \frac{\dot{\beta}_x}{\beta_x^2}(-\epsilon_x + v_x) \tag{3.18}$$

Next, we make use of the fact that

$$\dot{v}_x = K_p(\dot{x}_{ref} - \dot{x}_{cm}) + K_d(\ddot{x}_{ref} - v_x) \tag{3.19}$$

Since the coefficients $\dot{\epsilon}_x$ and $\dot{\beta}_x$ are functions of X and state accelerations excluding L_a, using Equation (2.20) with our control law in Equation (3.16) allows these terms to be calculated. Next, we construct the second derivative of u_{hip} in exactly the same manner, and similarly make use of the fact that

$$\ddot{v}_x = K_p(\ddot{x}_{ref} - v_x) + K_d(\dddot{x}_{ref} - \dot{v}_x) \tag{3.20}$$

New coefficients $\ddot{\epsilon}_x$ and $\ddot{\beta}_x$ appear and are functions of X, state jerk excluding L_a, and state accelerations, which we can calculate using Equations (2.20) and (3.15) for the state dynamics with Equations (3.16) and (3.18) for the input terms. Similar to the HOPFL construction for the leg state, the term $\ddot{L}_{act}$ appears in $\ddot{\epsilon}_x$, therefore the component of $\ddot{u}_{hip}$ due to u_{leg} must be separated and used to generate new effective coefficients as

$$\begin{aligned} \ddot{u}_{hip} &= \delta_1 + \frac{\delta_2}{\beta_x} u_{leg} \\ \tilde{\gamma}_y &= \gamma_y + \alpha_y \frac{\delta_2}{\beta_x} \end{aligned} \tag{3.21}$$

Finally, using equations (3.16), (3.18), and (3.8) we define our control law for y_{cm} as

$$u_{leg} = \frac{1}{\tilde{\gamma}_y}(-\epsilon_y - \beta_y u_{hip} - \eta_y \dot{u}_{hip} - \alpha_y \delta_1 + v_y) \tag{3.22}$$

where v_y is calculated via feedback, to drive the system to desired references as

$$\begin{aligned} v_y &= v_1 + v_2 \\ v_1 &= K_1(y_{ref} - y_{cm}) + K_2(\dot{y}_{ref} - \dot{y}_{cm}) \\ v_2 &= K_3(\ddot{y}_{ref} - \ddot{y}_{cm}) + K_4(\dddot{y}_{ref} - \dddot{y}_{cm}) \end{aligned} \tag{3.23}$$

Since the relative degree of the feedback linearization is four, we require four total poles in the closed-loop dynamics of y_{cm}. One option for setting the controller gains is to first select a dominant pole-pair with natural frequency ω_n and damping ratio ζ, and then set a significantly faster decay rate for the two remaining, real-valued poles, z_3 and z_4. For a chosen set of w_n, ζ, z_3, z_4, the controller gains are:

$$\begin{aligned} K_1 &= z_3 z_4 w_n^2 \\ K_2 &= (z_3 + z_4)w_n^2 + 2z_3 z_4 \zeta w_n \\ K_3 &= w_n^2 + 2\zeta(z_3 + z_4)w_n + z_3 z_4 \\ K_4 &= z_3 + z_4 + 2\zeta w_n \end{aligned} \tag{3.24}$$

Additionally, we require both references and estimates of the acceleration and jerk for y_{cm}, however, since u_{hip} and its derivative are known the acceleration and jerk can be analytically calculated. The unactuated body angle ϕ must remain bounded in this case for stable operation, and Section VI will illustrate how to construct CoM trajectories so this can be accomplished.

3.4.2 PFL Construction for Compliant 3-link Model

In this Section we provide a similar HOPFL construction about the CoM for the Compliant 3-link model. The control variables are the x and y locations of the CoM for the 3-link robot in (2.6). In this case the body stability of the system is a function of multiple robot links, and is therefore described by the auxiliary variable θ_B, also shown in (2.6). Using PFL, we can control at most two of these three, because we have only two actuators. In the FRANK implementation, it was only necessary to extend one of the control variables to a higher order, allowing us to control x_{cm} using a construction with relative degree 2. Although we could implement a similar approach here, we intentionally construct both components of the CoM at a higher order, to illustrate the feasibility of implementing such an approach where perhaps both control variables require relative degree greater than 2.

We start by taking derivatives to define $\ddot{x}_{cm}$, $\ddot{y}_{cm}$ and $\ddot{\theta}_B$, and by substituting Eq. 2.5 for the state acceleration variables, we rewrite the acceleration of our control variables as follows:

$$\begin{aligned} \ddot{x}_{cm} &= \gamma_x(X)\tau_1 + \epsilon_x(X) \\ \ddot{y}_{cm} &= \gamma_y(X)\tau_1 + \epsilon_y(X) \\ \ddot{\theta}_B &= \gamma_\theta(X)\tau_1 + \epsilon_\theta(X), \end{aligned} \tag{3.25}$$

where γ, and ϵ are functions of the state X. As with all previous HOPFL constructions on our hopping robots, it is important to note that for a standard acceleration-based PFL, i.e., with relative degree 2, the construction will fail due to the compliant leg. As we see in (2.13) this manifested in all β coefficients corresponding to τ_2 evaluating to zero for all X. In this case we need only take one additional time derivative, and thus the relative degree is only 3. We proceed by defining the jerk EoM by taking the time

derivative of Eq. 2.5, noting that the additional time dependent variables τ_1 and τ_2 must be included as:

$$J(X, \tau_1, \tau_2, \dot{\tau}_1, \dot{\tau}_2) = \frac{d}{dt} W(X, \tau_1, \tau_2) \tag{3.26}$$

Next, we take additional time derivatives of Eq. 2.12 to define $\dddot{x}_{cm}$, $\dddot{y}_{cm}$ and $\dddot{\theta}_B$, and by substituting in Eq. 2.5 and Eq. 3.26 we rewrite the jerk control variable equations in a form similar to that of the acceleration equations seen in Eq. 3.25:

$$\begin{aligned}
\dddot{x}_{cm} &= \beta'_x(X)\tau_2 + \gamma'_x(X)\tau_1 + \eta_x(X)\dot{\tau}_1 + \epsilon'_x(X) \\
\dddot{y}_{cm} &= \beta'_y(X)\tau_2 + \gamma'_y(X)\tau_1 + \eta_y(X)\dot{\tau}_1 + \epsilon'_y(X) \\
\dddot{\theta}_B &= \beta'_\theta(X)\tau_2 + \gamma'_\theta(X)\tau_1 + \eta_\theta(X)\dot{\tau}_1 + \epsilon'_\theta(X),
\end{aligned} \tag{3.27}$$

In the FRANK implementation, we used analytical calculations to generate the derivatives of u_{hip}, however, this may not always be possible for some control implementations (even though here it actually is). Therefore, in this case we approximate the derivative of the torque input coefficients by using a first order difference approximation, denoting a new variable $\tau_1[t-T]$ as the stored previous controller output value, at some controller sampling time T. We assume the sampling time T is fast enough to give a reasonable estimation. We can then group the coefficients together to create new effective coefficients as:

$$\begin{aligned}
\dot{\tau}_1 &\approx \frac{\tau_1 - \tau_1[t-T]}{T} \\
\tilde{\epsilon} &= \epsilon' - \frac{\eta \tau_1[t-T]}{T} \\
\tilde{\gamma} &= \gamma' + \frac{\eta}{T}
\end{aligned} \tag{3.28}$$

If we wish to control x_{cm} and y_{cm}, the control law is, after some algebra:

$$\tau_2 = \frac{1}{\beta'_y - \beta'_x \frac{\tilde{\gamma}_y}{\tilde{\gamma}_x}} (\tilde{\epsilon}_x \frac{\tilde{\gamma}_y}{\tilde{\gamma}_x} - \tilde{\epsilon}_y - \hat{v}_1 \frac{\tilde{\gamma}_y}{\tilde{\gamma}_x} + \hat{v}_2) \tag{3.29}$$

$$\tau_1 = \frac{1}{\tilde{\gamma}_x}(-\tilde{\epsilon}_x - \beta'_x \tau_2 + \hat{v}_1), \tag{3.30}$$

where $\hat{v}_1$ and $\hat{v}_2$ are calculated via PD feedback to drive the system to desired references:

$$\begin{aligned} \hat{v}_1 &= K_p(x_{ref} - x_{cm}) + K_d(\dot{x}_{ref} - \dot{x}_{cm}) + K_{dd}(\ddot{x}_{ref} - \ddot{x}_{cm}) \\ \hat{v}_2 &= K_p(y_{ref} - y_{cm}) + K_d(\dot{y}_{ref} - \dot{y}_{cm}) + K_{dd}(\ddot{y}_{ref} - \ddot{y}_{cm}). \end{aligned} \tag{3.31}$$

Since the relative degree is 3, we must estimate the acceleration of our control variables. This can be accomplished by simply using the most-recently stored torque commands, $\tau_1[t-T]$ and $\tau_2[t-T]$, in the place of τ_1 and τ_2 in Eq. 3.27 to give a reasonable numerical estimate. We can again set the pole locations for the supplanted error dynamics by selecting a dominant pole-pair with natural frequency ω_n and damping ratio ζ, and then set a significantly faster decay rate for the remaining, real-valued pole, p_3. For a chosen set of ω_n, ζ, p_3, the gains are:

$$\begin{aligned} K_p &= p_3 \omega_n^2 \\ K_d &= \omega_n^2 + 2p_3 \zeta \omega_n \\ K_{dd} &= p_3 + 2\zeta\omega_n \end{aligned} \tag{3.32}$$

Lastly, by utilizing a PFL-based technique we have the flexibility to choose *any two* of our three variables of interest to control, and therefore if desired, we could for example, instead control y_{cm} and θ_B. The significant challenge for implementing this approach is

again the design of feasible trajectories for our two regulated variables that ensures the unregulated zero dynamics are well behaved. This is discussed in chapter VI.

It is also of note to mention that in general, coefficients for HOPFL controllers can be quite complex and potentially difficult to generate. Practical considerations when calculating these coefficients are therefore disused in Appendix B.

Chapter 4

Apex Control of SEA Hoppers

We consider the problem of apex height regulation, considered by many to be a "trivial 1D problem", to illustrate a few key points. First, in the presence of SEA dynamics, even the simple case of apex height regulation may not be necessarily straightforward, as using simple methods do not typically result in good performance as we saw in Fig. 3.1. Second, this is the simplest type of control to facilitate testing our HOPFL control-based method. We draw a parallel between more simple methods and point out the obvious improvements our method has to offer. Third, this problem is feasible for deployment on the robot FRANK, and will verify that HOPFL is something we can actually implement on a real robot that inevitably has model mismatches, sensor noise, etc.

4.1 PID-based 1D Height Regulation

This Section presents the simplest type of apex height regulation one can consider using to control the apex height for 1D hoppers, and will show that these simple techniques do not perform well for real hopping robots. These controllers are applicable to both the Hopper B and Hopper C models in (2.9) and (2.10) respectively. The first method

one may be temped to try using is feedback of apex state information to generate feed-forward commands, and command a **constant current step** during the stance phase. This requires taking discrete measurements $h[n]$ of the robot's height at each apex state $\mathcal{A}$, and attempting to command reference heights $r[n]$, where n represents the current hop number out of a sequence of N_{hops} hops. Thus, at each apex $\mathcal{A}$ we calculate

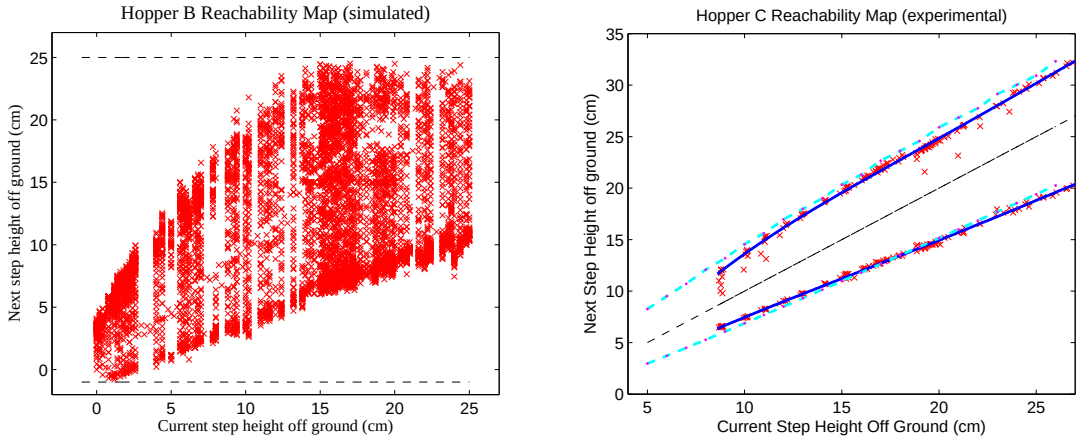
$$u_{PID} = K_P(r_n - h_n) + K_D(r_n - h_n - r_{n-1} + h_{n-1}) + K_I \sum_{n=1}^{N_{hops}} (r_n - h_n)$$

and in the subsequent stance phase we set $u_{leg} = u_{PID}$ for the entire duration of stance. Simulation results for this method applied to the Hopper C were already shown in the top image from Fig. 3.1, and it is quite clear this method does not yield acceptable performance. This control method lacks both a dynamics-based feed-forward term and feedback correction, and is ill-suited for implementation.

One can attempt to implement a smarter but still "simple" approach by making use of reachability maps and drawing inspiration from Raibert's thrust controller [8]. Raibert's method involved applying thrust action at mid-stance, i.e., when the system spring is expanding. Applying thrust actions during expansion for the purpose of increasing vertical energy of Active SLIP has also been implemented in the literature [16]. Therefore, we implement the following control law during the stance phase:

$$u_{leg} = \begin{cases} MAP(h, r) & \text{during compression} \\ K_p(L_{ref} - L) + K_d(\dot{L}_{ref} - \dot{L}) & \text{during expansion} \end{cases}$$

where L_{ref} and $\dot{L}_{ref}$ are smooth trajectories of length N with initial values aligned to



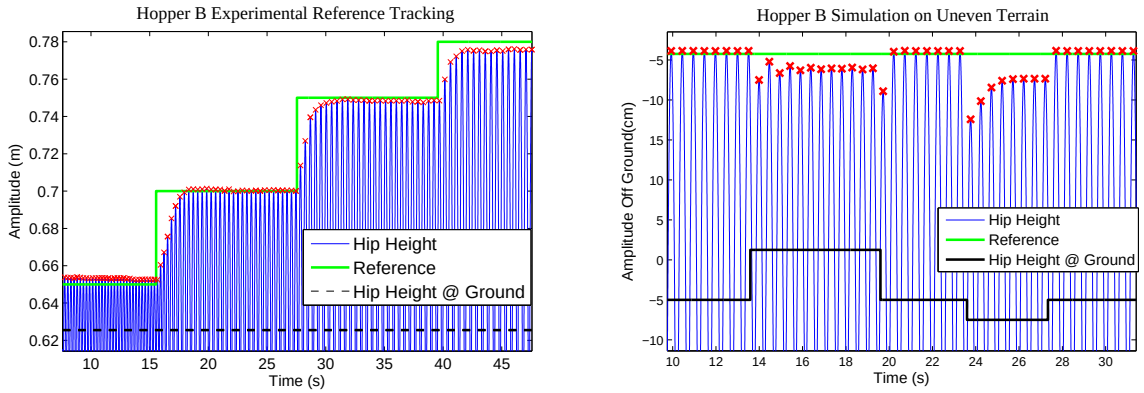
(a) (b)

Figure 4.1: Reachability maps for the 1D Hopper B (a) and Hopper C (b), defined as the set of possible vertical hops from each apex height. In the experimental data shown in (b), the red x's and blue lines represent measured hardware data, and the cyan dotted line represents expected simulation results. Experimentally mapping the reachable space can be quite time consuming, and is ill-suited for the more complex 2D hoppers.

the mid-stance initial conditions, and terminating values set as:

$$L_{ref}(N) = L_0$$
$$\dot{L}_{ref}(N) = (2g(r - L_0))^{1/2}$$

Thus, the feedback component during expansion attempts to drive the leg to the desired take-off velocity. During compression the feed-forward term $MAP(h, r)$ is constructed using a reachability map. For the 1D case, the reachability map is simply the set of next possible apex heights as a function of the current apex height, and thus it is tractable to construct a table of feed-forward commands as a function of current and desired apex states. Example reachability maps for the Hopper B and Hopper C are shown in Fig. 4.1. Note that this method does not extend well to the 2D case as the additional apex states (i.e. θ and ϕ) make both measuring and storing the data somewhat intractable.



(a) (b)

Figure 4.2: Results implementing the "improved" simple control method for regulating apex heights. As we see in the experimental performance in (a), results still leave much to be desired. Additionally, this method has little chance of functioning on rough terrain, as the simulation results in (b) show. These results suggest that even the "simple" problem of 1D height regulation is not quite as easy to implement on a real hopper as many might believe.

This control method was implemented on the Hopper B hardware, and as we can see in Fig. 4.2 (a), although an improvement over previous results, performance is still not good, due to two core reasons. First, the reachability map will inevitably be somewhat inaccurate to the real system, and certainly will not perform well on terrain not exactly as the data was measured. This is illustrated in Fig. 4.2 (b), where the controller clearly has no hope of rejecting disturbances. Second, and more importantly, simply "tacking on" a feedback controller to a highly compliant system is very pedestrian, as it is impossible to instantaneously set the acceleration of the system as we saw in Chapter III. Thus, we should consider applying our HOPFL controllers to better control the system.

4.2 HOPFL-based 1D Height Regulation

In this Section we provide a more precise apex height regulation algorithm for vertical 1D hopping, by constructing an analytical solution and using it as a reference for

a HOPFL controller on the leg state of the robot from (3.22). For the 1D case an analytical solution is possible to determine exactly if we assume the actuation of the SEA is a constant current step with magnitude u. Due to the Coulombic friction terms, this calculation is performed piecewise linearly, and the $||$ operator represents the concatenation of each piecewise analytical solution. The detailed analytical calculations for each piece using Inverse Laplace Transform Techniques can be seen in Appendix C. Given touch-down initial conditions $L_{0,0}$, $\dot{L}_{0,0}$,$L_{a0,0}$, $\dot{L}_{a0,0}$, and u, we can solve the dynamics in (2.9) (and/or (2.10)) to generate:

$$\begin{aligned} L_a(t) &= \mathop{\Big|\Big|}_{i=1}^{N_{pieces}} x_1(i, L_{0,i-1}, \dot{L}_{0,i-1}, L_{a0,i-1}, \dot{L}_{a0,i-1}, u) \\ L(t) &= \mathop{\Big|\Big|}_{i=1}^{N_{pieces}} x_2(i, L_{0,i-1}, \dot{L}_{0,i-1}, L_{a0,i-1}, \dot{L}_{a0,i-1}, u) \end{aligned} \tag{4.1}$$

The term N_{pieces} represents the number of piecewise components the solution must be broken up into, which is typically only 2, with the two pieces being compression and expansion, separated with the leg velocity changing sign at mid-stance. To construct trajectories for regulating apex heights, we consider the total energy of the spring-mass system during stance as

$$U_L = \int (k(-L(t) + L_0 + c) - m_B g)\dot{L}(t)\,\mathrm{d}t + U_\delta \tag{4.2}$$

where

$$U_\delta = -f_2(L_0 - L(t)) - \int b_2 \dot{L}(t)^2\,\mathrm{d}t + \int k L_a(t)\dot{L}(t)\,\mathrm{d}t - \delta_{TD} \tag{4.3}$$

Using the analytical solutions in (4.1), U_δ, deemed the Analytical Energy Delta (AED), can be computed exactly over all stance time; it represents the sum of all energy loss terms during stance, along with the energy added by the actuator. The term δ_{TD} represents any

instantaneous energy loss at touch-down, which can be calculated using initial conditions and the corresponding impact dynamics, for example, those of FRANK in Equations (2.16) and (2.15). With these equations, we can calculate exactly what apex heights the system will reach over all time, given initial conditions entering the first stance phase. Both the friction terms and the unsprung mass at the foot account for significant energy loss at each successive hop, and thus actuation is needed to introduce additional energy into the system as shown in Fig. 4.3, for either stochastic, height-varying terrain or for steady-state hopping or flat ground. To achieve consistent steady-state hopping, we can, of course, use our equations for U_δ to find the magnitude of current needed to achieve this.

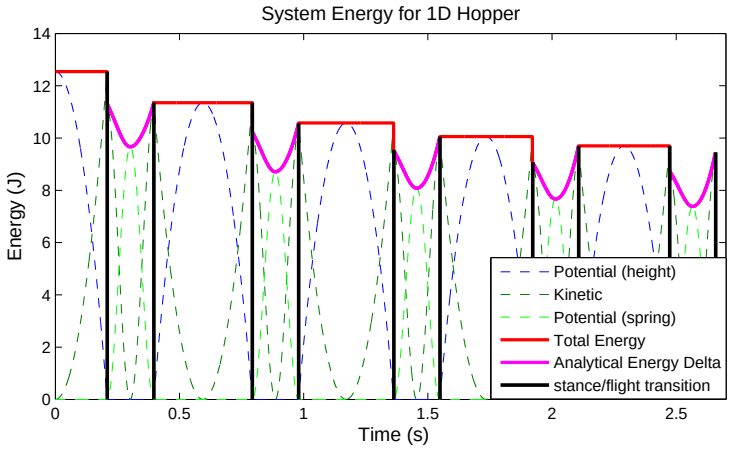


Figure 4.3: The system energy in (4.2) for 1D hoppers can be easily visualized during an actuated simulation. Energy is added to the system via the U-shaped dips between the flat plateaus of the ballistic phase. The AED is show in magenta and can be analytically calculated using (4.3) as a function of each set of initial conditions and input magnitude.

The algorithm to compute a set of HOPFL leg trajectories given initial conditions proceeds as follows. During flight, we forward solve the ballistic dynamics in (2.2), and any impact dynamics at touch-down. At the instant after touch-down, the energy level

of the system is computed by

$$E_{TD} = \frac{1}{2} m_B \dot{L}^2 \tag{4.4}$$

If a ground height disturbance, y_h, is present, the disturbance energy can be computed as

$$E_{dist} = \frac{1}{2} m_B g |y_h - L| \tag{4.5}$$

U_δ is then computed for the end of the stance phase as $E_\delta(u_{FF})$, where u_{FF} is a constant feed-forward current term. The controller selects current step magnitude u_{FF} to achieve the correct next-apex energy level as

$$u_{FF} = \arg\min |E_\delta(u_{FF}) - (m_B g h_{des} - E_{TD} - E_{dist})|, \tag{4.6}$$

where h_{des} is the desired next apex height. The above minimization function may not be analytically solvable, but finding the correct value of u_{FF} can be accomplished by simply calculating $E_\delta(u_{FF})$ over a window of values and determining the minimum value for some resolution. Although this returns a feed-forward current step magnitude u_{FF}, our purpose in performing this calculation is to use the resulting analytical solutions for the leg length L as references for the HOPFL Leg controller in (3.22). In this case, since we are operating a vertical hopper, we evaluate Eq. (3.22) with u_{leg}, θ, and ϕ all equal to zero. Simulation results during a stance phase, implementing a set of leg state solutions as trajectories for the HOPFL leg controller, are shiwn in Fig 4.4, with current consumption by the SEA shown in Fig. 4.5. Apex tracking simulation results implementing this approach are shown in Fig. 4.6 (a). As we can clearly see, these methods yield significant improvement over those of 3.1 and 4.2, and can function on uneven terrain.

It is of interest to note that since this method does indeed produce feedforward

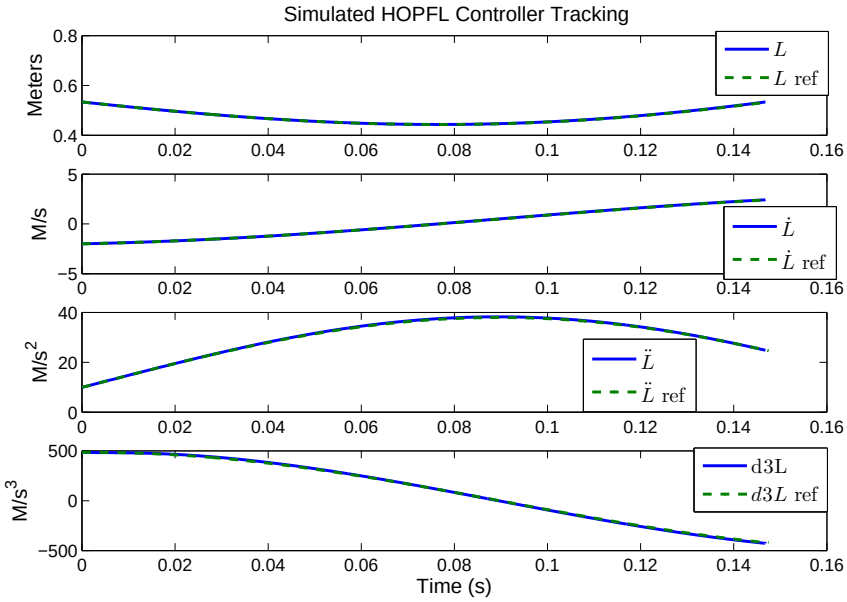


Figure 4.4: Simulation results for the described HOPFL leg controller, showing the controller's ability to track references directly on the leg state L. Trajectories were generated using our analytical SEA 1D solutions with an input current step of 5.8 A.

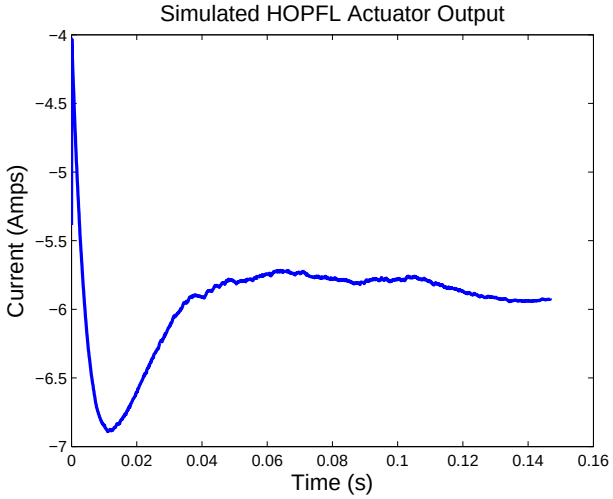
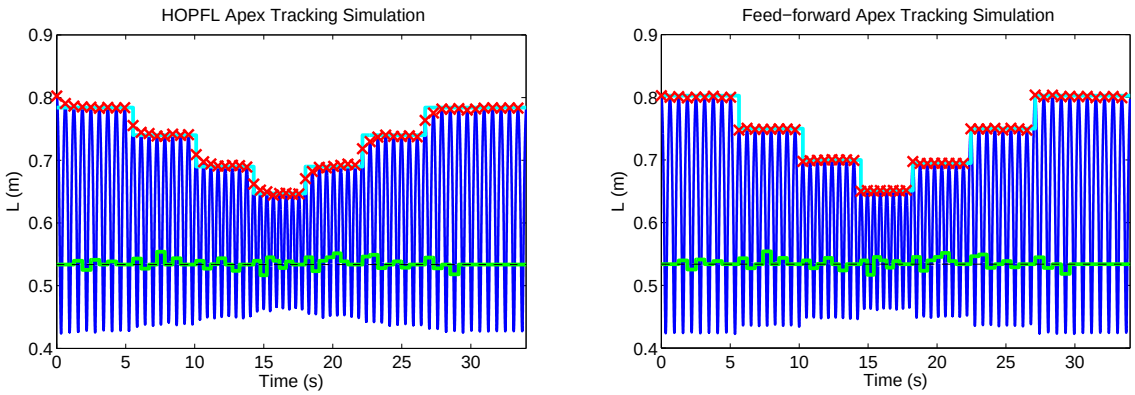


Figure 4.5: Actuation requirement in Amps for the described HOPFL leg controller during a simulated stance phase. Compared to the actuator effort of the FF strategy for identical trajectories, which for this particular example is a constant 5.8 A, the average current applying the HOPFL controller is only slightly higher at 5.94 A.



(a) (b)

Figure 4.6: Simulation results implementing 1D height regulation on the Hopper C are shown for (a) using the HOPFL leg state controller and (b) simply using the constant feed-forward command $u_{leg} = u_{FF}$. It is not surprising that the feed-forward strategy performs better given a perfect model with no noise, however, it is shown later however that using only feed-forward commands will perform *significantly* worse for both model mismatch and during 2D forward motion.

commands u_{FF}, we can simply construct an apex height regulating controller that sets $u_{leg} = u_{FF}$ during the entire stance phase. Figure 4.6 (b) illustrates that this does indeed provide improved performance over the pedestrian techniques from the previous Section, and at first appears to even outperform our HOPFL controller. We will later provide comparative results in both simulation and hardware showing that this method is quite inferior compared to instead using HOPFL to regulate a corresponding stance phase analytical solution as a trajectory, as it has no feedback correction.

4.3 PFL Coefficient Parameter mis-match

This Section considers applying our apex control methods in the presence of imperfect model information, which is often considered a limitation of PFL in general. It is, of course, necessary to obtain some level of accurate model information to have good performance. For SEA hopping robots, the spring constant, system mass, and actuator

coupling terms must be precisely known to have hope of achieving good performance. Luckily, these parameters can be determined quite accurately using system identification techniques, as can be seen in Appendix A.

Often when performing system identification of real systems, the frictional terms can be the most challenging to determine precisely, since they can vary both over time and the exact system state. This has proven to be a significant practical challenge for the SEA in our laboratory hardware. Therefore, it is of particular interest to study control results when the SEA frictional parameters used by the controller, e.g. f_1 and b_1 in Eq. C.1, do not match those of the real system. In this study, we assume the true dynamics of the system are those of FRANK from Table 2.1, and vary the frictional parameters used by the controller in order to study how these errors affect performance of our apex tracking controllers presented in the previous Section.

We use a base trajectory of apex heights we would like the system to follow for these studies. The apex heights were selected to span a reasonable amount of the reachable states seen in Fig. 4.1, and the ground level is also randomly varied on a step-to-step basis in order to simulate minor terrain variations. Simulation results for both control methods with perfect model information are shown in Figures 4.6. With correct parameters, the purely feed-forward method has negligible error, while the HOPFL method has some small tracking error. To conduct the simulation study, we define a range of controller frictional values of the series elastic actuator to iterate over as $[f_{1,min}, f_{1,max}] = [0, 2]$ and $[b_{1,min}, b_{1,max}] = [0, 30]$, which are the values the controller will use on the real system. The controllers are commanded to track the base apex height trajectory for each parameter combination in this range, and the sum of squared error (SSE), normalized by the square of the reference to be unit-less, is recorded by summing the ratio of squared apex height clearance off ground errors to every desired apex in each trial. For reference, the SSE of the purely feed-forward implementation using correct parameters is 0.12, and

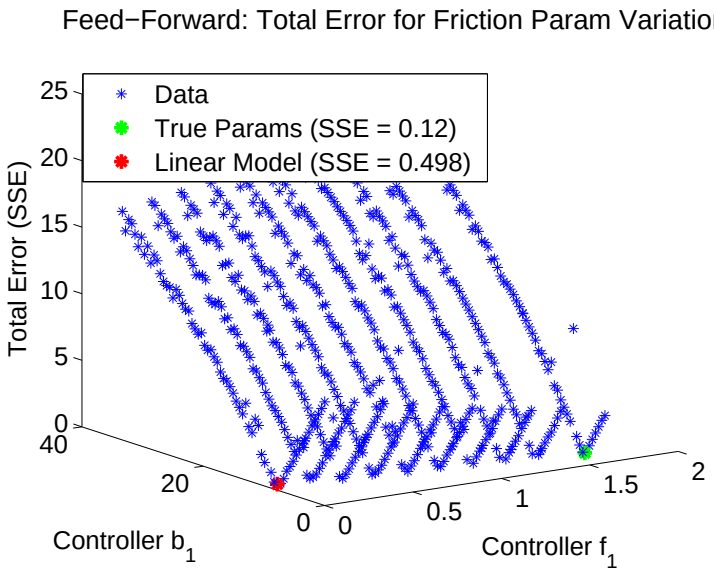


Figure 4.7: Results for variation of the SEA model's frictional terms used by the purely feed-forward controller. Each blue point represents a simulation with one set of incorrectly used frictional parameters. The feed-forward method is quite sensitive to variance of these model parameters, and is likely not robust enough to achieve good performance on hardware implementations.

the average apex error (AE) was 0.7%. In contrast, the HOPFL controller has SSE 0.33 and AE 1.5% with correct parameters.

Results for this simulation study are shown in Figures 4.7 and 4.8. As expected, the feed-forward controller performs poorly when the controller parameters do not match the system dynamics. While the feed-forward controller has very good error using the true parameters, other points with reasonably small error exist and represent cases when the friction and damping terms are identified incorrectly, but the total sum of their effect on energy loss is approximately the same. One example of this is highlighted as a red dot on Fig. 4.7, in which case the controller uses $f_1 = 0$ and $b_1 = 7.346$, meaning the model is assumed by the controller to have only linear damping terms. This error is still roughly five times larger than the ideal parameters.

The HOPFL implementation performs significantly better with model inaccuracies, as shown in Fig. 4.8. The maximum error, i.e. when the controller uses horribly inaccurate

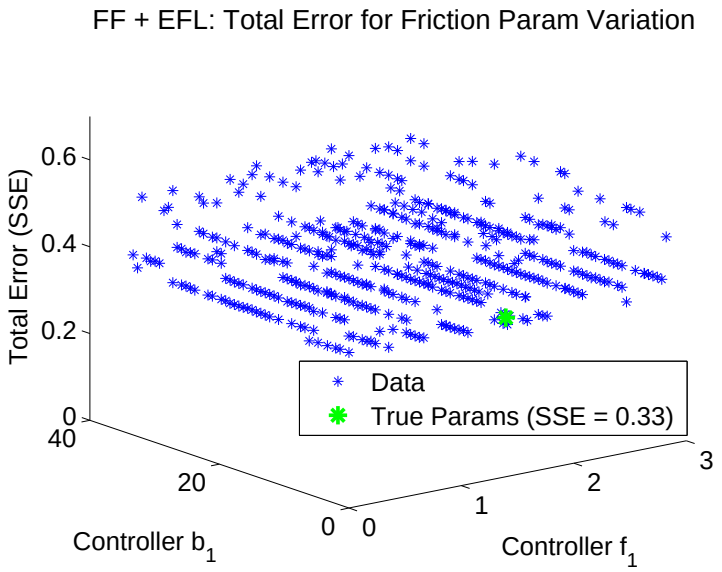


Figure 4.8: Results for variation of the SEA model's frictional terms for the HOPFL-based method. Noting the differences in the y-axis scale compared to the results in Fig. 4.7, this method provides significantly improved performance for the case of imperfect modeling.

frictional parameters, is *significantly* less compared to simply using the feed-forward current commands. This is perhaps not a surprising result, and reinforces the fact that feedback correction is necessary to achieve good performance for hopping systems.

4.4 Apex Height Regulation in Steady-State Forward Motion

Thus far we have considered vertical regulation of 1D hoppers only. However, it is possible in some cases to use these methods on 2D hopping robots. Note that in general, precise apex height regulation alone is *not* sufficient to achieve good step length results for irregularly spaced footholds, as this is a significantly more complex problem and is investigated in the next chapter. In this Section we consider the problem of regulating apex heights while the SEA 2D Hopper, i.e. FRANK with the body mechanically locked

with dynamics in (2.14), is hopping in steady-state forward motion.

Although the system has a hip actuator u_{hip} in addition to the SEA, we only allow this actuator to position the leg in a desired touch-down angle during the flight phase. Only the SEA outputs power during stance. Thus, during stance u_{hip} is zero, and during the flight phase, u_{hip} uses a simple controller to position the leg to set the touch-down angle θ_{TD} to achieve forward hopping. For simplicity we use a very simple controller, used by Raibert [38] in historic work with hoppers, to set the touch-down angle, given as

$$\theta_{TD} = \theta_0 + K(\dot{x}_r - \dot{x}_{hip}) \tag{4.7}$$

where θ_0 is a constant touch-down angle, and $\dot{x}_{hip}$ is the forward velocity of the hip joint, with forward velocity reference $\dot{x}_r$ and gain K.

Given this forward gait, we first naively attempt to directly apply out 1D HOPFL control method to track a set of apex trajectories during forward movement. Simulation results with θ_0 and $\dot{x}_r$ of -4.5 degrees and $0.66\frac{m}{s}$ respectively are show in Fig. 4.9, where it is clear significant apex error is present for all time. However, due to the HOPFL controller component accuracy providing feedback on leg compression trajectories, we see that the controller error is very consistent and appears to simply be an energy offset. This energy difference E_θ can in fact be calculated, and is represented as

$$E_\theta = \frac{1}{2} m_B (\dot{y}_{TO})^2 - m_B g (h_{des} - L_0)$$
$$\dot{y}_{TO} = (1-\rho)\dot{L}_{TO}\cos(\theta_{TO}) - \dot{\theta}_{TO} L_0 \sin(\theta_{TO})$$

Where $\dot{y}_{TO}$ is the take-off hip vertical velocity, h_{des} is the desired next apex height, and $\dot{L}_{TO}$, θ_{TO}, $\dot{\theta}_{TO}$ are the future velocity states of the system at the end of the stance phase, right before take-off. We must of course account for the impact dynamics from Eq. 2.16 with $\rho = \frac{m_l}{m_B + m_l}$. Therefore at touch-down, the next take-off state must be

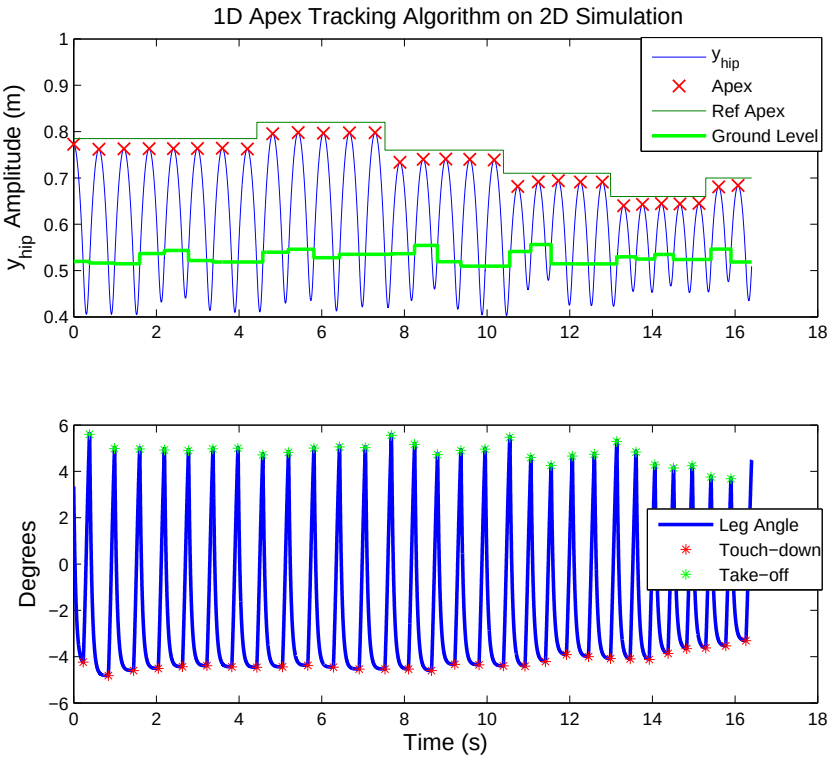


Figure 4.9: Attempting to use the HOPFL-based 1D apex regulation control strategy for our 2D hopper during steady-state forward motion results in significant apex error. However, the touch-down and take-off angles, while not exactly symmetric, are reasonably close to one another such that a simple energy adjustment to the control algorithm is possible.

partly estimated in order to correct for the energy offset. We note however that since our HOPFL controller component is regulating L and $\dot{L}$, the take-off value $\dot{L}_{TO}$ will simply be the terminating value of the trajectory as

$$(1-\rho)\dot{L}_{TO} = \sqrt{2g(h_{des} - L_0)} \tag{4.8}$$

We also note from Fig. 4.9 that during stance the angle of the leg swings to within a degree of the touch-down angle. Since this method is used only in steady-state forward motion, we use the following simple but adequate approximation for the take-off angular

values:

$$\begin{aligned} \theta_{TO} &\approx -\theta_{TD} \\ \dot{\theta}_{TO} &\approx \dot{\theta}_{TD} \end{aligned} \tag{4.9}$$

We can then use Eqs. 4.8 and 4.9 in Eq. 4.4 to approximate the energy difference as $\tilde{E}_\theta$, and the only change to our apex regulation algorithm is in the energy cost function as

$$J_{2D} = |E_\delta(u_{FF}) - (m_B g h_{des} - E_{td} - E_{dist} + \tilde{E}_\theta)| \tag{4.10}$$

These assumptions allow us to estimate the energy offset, as compared to 1D hopping, that occurs during steady-state 2D motion and command a slightly higher take-off velocity. After applying this minor energy adjustment we implement the controller exactly as in the 1D case. Simulation results for this algorithm are show in Fig. 4.10, and indeed the energy offset has been corrected and apex tracking performance is virtually identical to the 1D simulations. The average current consumption for this simulation is 2.63 Amps, which is well within continuous current limits of hardware. All apex height regulation techniques presented thus far have been successfully implemented in hardware on the robot FRANK, as it will be shown next.

4.5 Implementation on FRANK Hardware

In this Section we present motivating hardware results implementing our HOPFL strategy on the robot FRANK with the body mechanically locked, regulating apex heights during steady-state forward motion and rejecting ground disturbances during vertical hopping. We acknowledge these are simpler problems than foothold placement, which we address in simulation in the next two chapters. However, these results demonstrate

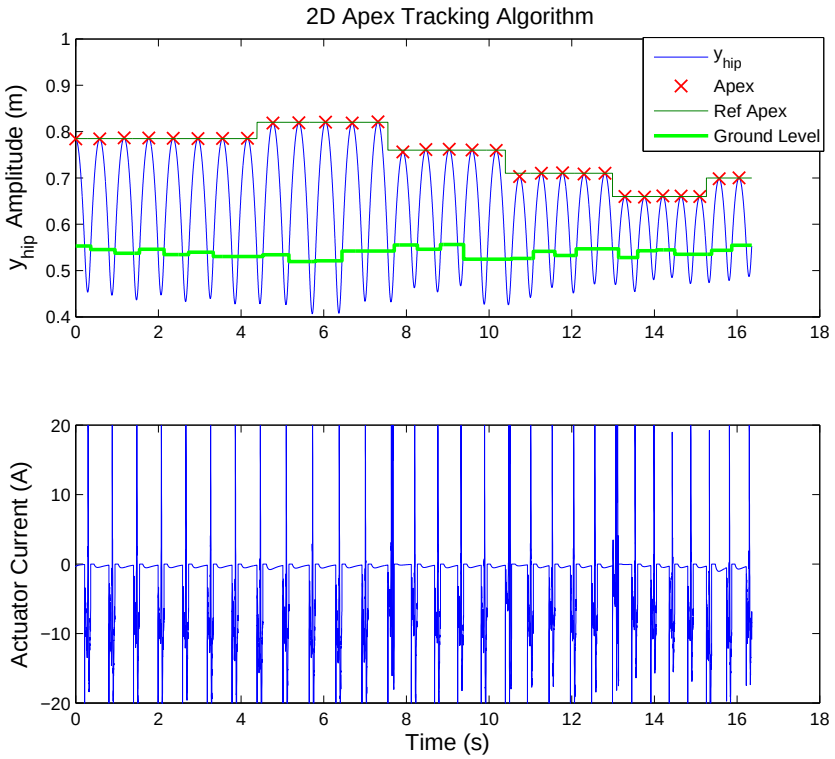


Figure 4.10: Estimating and correcting the energy offset results in good performance even in the presence of rough terrain, similar to the 1D results. The actuator output, while exhibiting some initial saturation at touch-down, is reasonably well behaved.

that using high-order PFL on the leg state is feasible and performs well on real hardware, which is inevitably affected by model inaccuracies, touch-down angle control difficulties (see Appendix D), and imperfect sensing, currently preventing us from performing more elaborate control experiments.

In order to adopt an apex height control strategy to the robot FRANK, some hardware details must be accounted for. Although a mechanical lock prevents the boom side of the robot from rotating, torsional forces on the carbon fiber boom result in the body of the robot flexing by a few degrees during operation. The effect on the stance phase dynamics is largely negligible. However, when controlling a specific touch-down angle θ_{TD} the relative encoder reading at the hip for θ must be added to the body angle ϕ

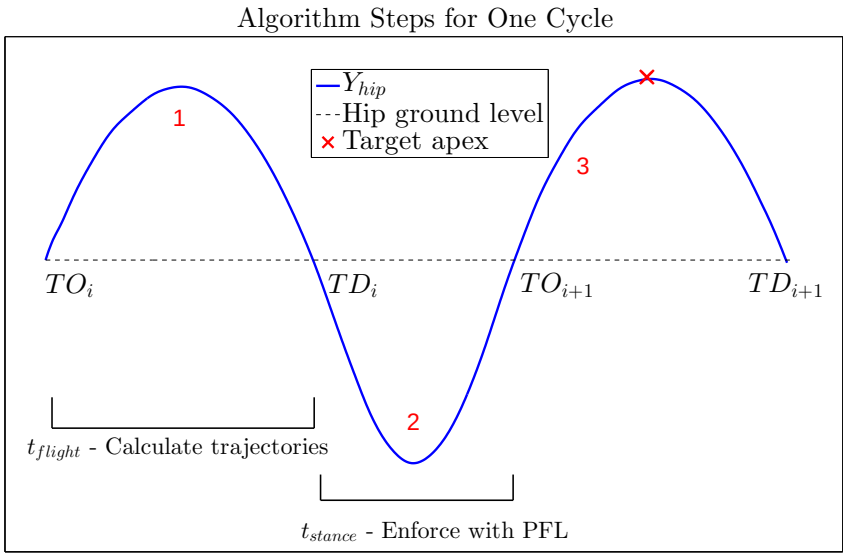


Figure 4.11: To regulate an apex state on the hardware, all control must be accomplished in the preceding stance phase. Furthermore, the feed-forward construction must complete all required calculations before the touch-down event, with a typical flight phase lasting 300ms-400ms, so the reference trajectories are ready at the start of the stance phase.

to correctly position the global leg angle. A pair of HC-SR04 acoustic sensors were mounted to each side of the robot in order to estimate the body angle ϕ so this could be accomplished. Since the robot hardware is tethered to a boom, during the flight phase the boom angle ψ, i.e., the angle that the boom forms vertically with respect to the horizon, follows non-linear dynamics that must be accounted for when determining proper take-off velocities. During hopping $|\psi| \leq -15$ [deg], therefore we use a small angle approximation to analytically solve and convert information from the leg states L and $\dot{L}$ to boom flight phase states ψ and $\dot{\psi}$. Detailed information regarding derivation and system identification of the flight phase ballistic-like boom dynamics can be seen in Appendix A. Lastly, all computation is running locally on the robot hardware, which has a maximum sampling rate of 1 KHz. Determining the correct feed-forward solution requires some calculation time, and since a typical stance phase can be as short as 150 ms, all calculations must be accomplished in the preceding flight phase in an algorithmic way,

as explained by Fig. 4.11, with algorithm steps {1},{2}, and {3} listed in red.

Given take-off initial conditions for the ith flight phase, either at TO_i or the corresponding apex, the algorithm forward solves impact and flight dynamics {1} to generate stance phase initial conditions TD_i, and then calculates the feed-forward solution in real-time during the current flight phase for the next stance phase. The robot determines the parametrizing current u_{leg} by solving 4.10 using a binary search algorithm, which takes roughly 0.1 sec to compute and is well within flight time requirements. At each iteration, the algorithm loops through steps $\{2-3\}$, using the analytical stance phase equations in (4.1) to solve for the stance phase trajectory, the impact dynamics in (2.16) with boom geometry, and flight phase analytical equations to calculate a resulting apex state. Upon finding the correct apex state for a given reference, the algorithm saves the stance trajectory $L(t)$ and the first three derivatives for use in the HOPFL leg state controller in (3.9) during the subsequent stance phase. Experimental reference tracking performance by the HOPFL controller for a set of trajectories is shown in Fig. 4.12, with actuator usage u_{leg} for a single experimental stance phase shown in Fig. 4.13. The SEA position L_a is also shown to illustrate that it remains quite well behaved. These results verify the feasibility of using our 4th-order PFL control on real hopper hardware.

4.5.1 Apex Height Regulation during Vertical Hopping

Experimental apex tracking performance during vertical hopping was implemented on the robot FRANK. To provide horizontal stabilization, we simply use our touch-down angle control law in (4.7) with θ_0 and $\dot{x}_r$ both set to zero. Hardware results using HOPFL can be seen in Fig. 4.14, and hardware results using only feed-forward commands (i.e., $u_{leg} = u_{FF}$) can be seen in Fig. 4.15. Clearly, the HOPFL-based method performs significantly better, as it has feedback correction to help with model inaccuracies, sensor

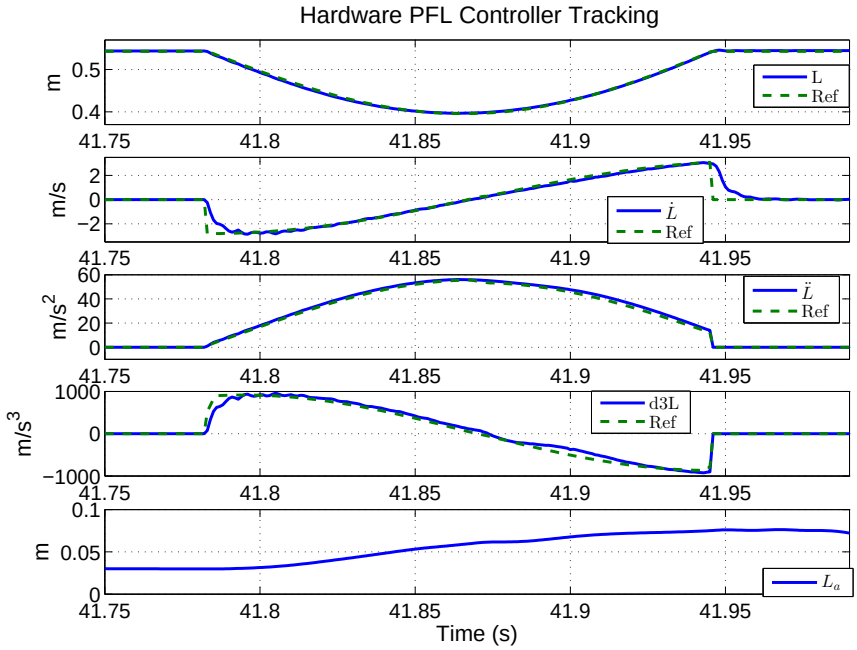


Figure 4.12: The robot FRANK was successfully able to track all references during the stance phase with reasonable results. The state measurement for $\dot{L}$ is estimated using a filter, and the state measurements for $\ddot{L}$ and $\dddot{L}$ are calculated in real-time using analytic derivatives of the dynamics in (2.14).

noise, etc. It has been well shown thus far that simply using a feed-forward command for the SEA actuator does not result in good performance, therefore all subsequent chapters and Sections will focus solely on using the HOPFL-based methods.

4.5.2 Apex Height Regulation in Forward Motion

Hardware results for apex tracking during approximate steady-state motion, with a commanded forward speed of $\dot{x}_r = 0.42$ ms^{-1}, are shown in Fig. 4.16. As the results show, the robot is able to track apex references that quickly change on a step-to-step basis even in the presence of noisy forward velocity signals. This is a considerable improvement over results generated via simplistic PID methods shown in Fig. 3.1. By using feed-forward based trajectory generation, we are able to track step-to-step apex changes and also gain

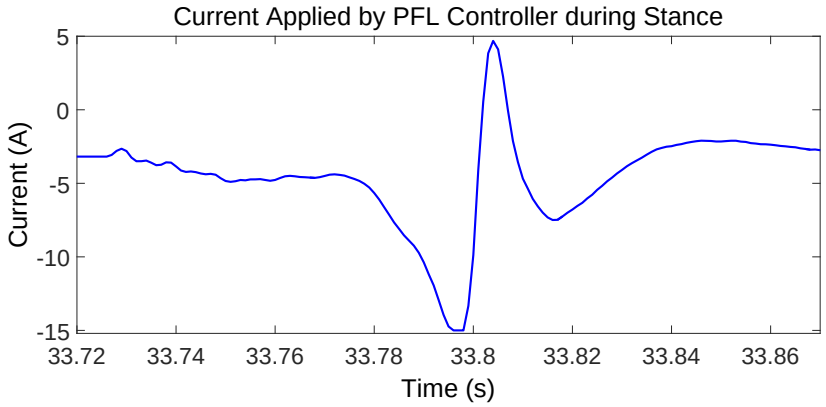


Figure 4.13: Above shows the current applied by the SEA hardware during a stance phase. This stance trajectory corresponds to a solution generated with $|u_{FF}| = 4.23A$, and the average absolute current usage here is $4.83A$, therefore we estimate usage of the HOPFL controller on our hardware costs approximately 14% more power when compared to a purely feed-forward method.

increased tracking performance of feedback control by using our high order PFL controller directly on the leg state. Accuracy can likely be improved in future work by using the leg angle actuator, which is currently unused in stance, to better regulate forward speed, but comparing to Fig. 3.1 we clearly show hardware results that are an improvement over simulation results using conventional methods. We also note that both accuracy of forward speed and apex height can likely be improved by more elaborate touch-down angle controllers, usage of the hip actuator in stance, and in particular incorporating model information of SLIP-like angular dynamics, which is explored in the next chapter in simulation.

4.5.3 Ground Disturbance Rejection

Another desirable attribute of control systems for legged robots is the ability to reject ground disturbances well enough to allow for operation on rough terrain. Although the acoustic sensor based body angle measurements currently prevent FRANK from traversing terrain boards, the control algorithm can be easily augmented to handle non-

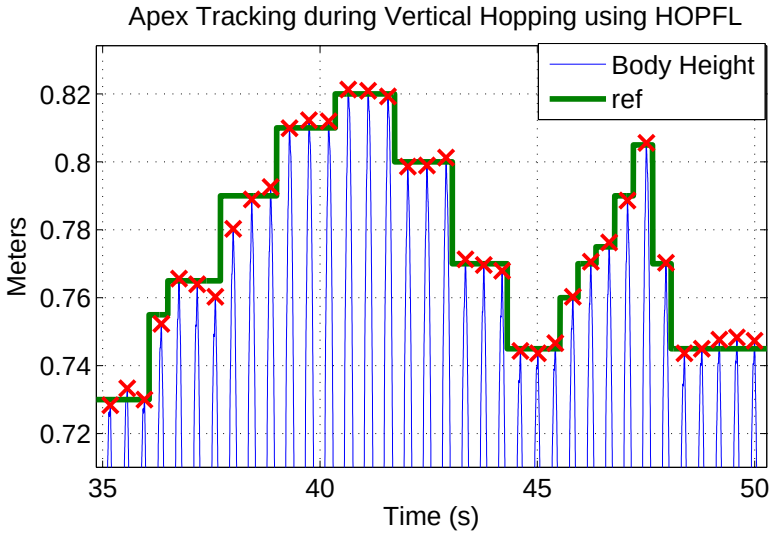


Figure 4.14: Using the HOPFL controller to track apex heights on the hardware FRANK results in quite good performance, and are quite similar to the simulation results we saw in Fig. 4.6.

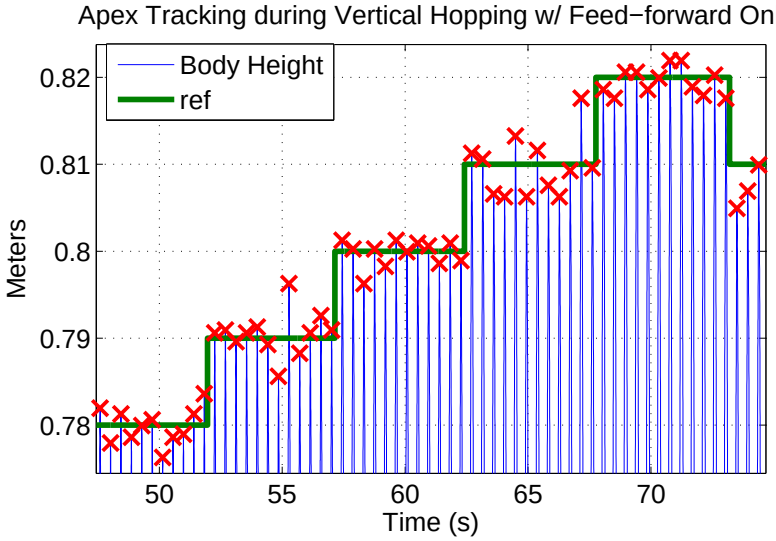


Figure 4.15: Simply using feed-forward commands does not result in particularly good performance on the hardware. This is an expected result, as when we simulated model mis-matches in only the frictional parameters we saw tracking errors in general did not remain small. Therefore, we conclude, as expected, stance phase feedback correction is necessary to obtain good results for hardware deployment of hopping robotic systems.

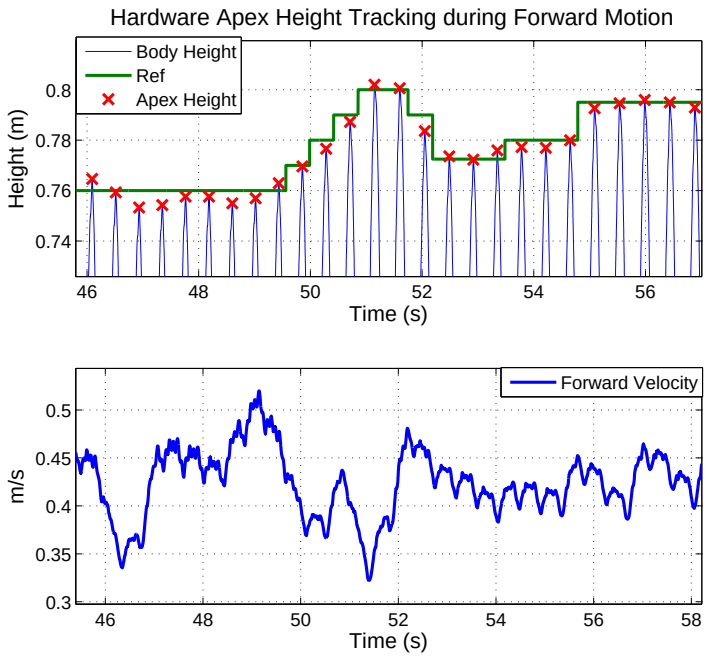


Figure 4.16: The controller is able to track apex heights on the hardware, even during forward motion. The average apex error recorded for this data is $1.6\% \pm 1.3\%$ of the commanded apex height clearance off the ground.

zero ground levels. This is accomplished by simply computing in parallel multiple solution sets over a range of ground levels in algorithm steps 2-3 from Fig. 4.11, and at touch-down selecting the solution corresponding to the proper ground height. This makes use of parallel computing and requires the ability to estimate the level of the ground at touch-down, which is a capability of the robot FRANK. During experimental vertical hopping large boards of approximate thickness 1.6cm and 3.8cm were injected under the robot during flight. Figure 4.17 displays the robot's ability to easily reject these ground disturbances, allowing potential operation on rough terrain while retaining reasonable apex accuracy.

These hardware results motivate further investigation of using high order PFL in more complicated hopping gaits. Specifically, in the next Sections we investigate in simulation generating trajectories for the 2D problem of step length control in such a way that we

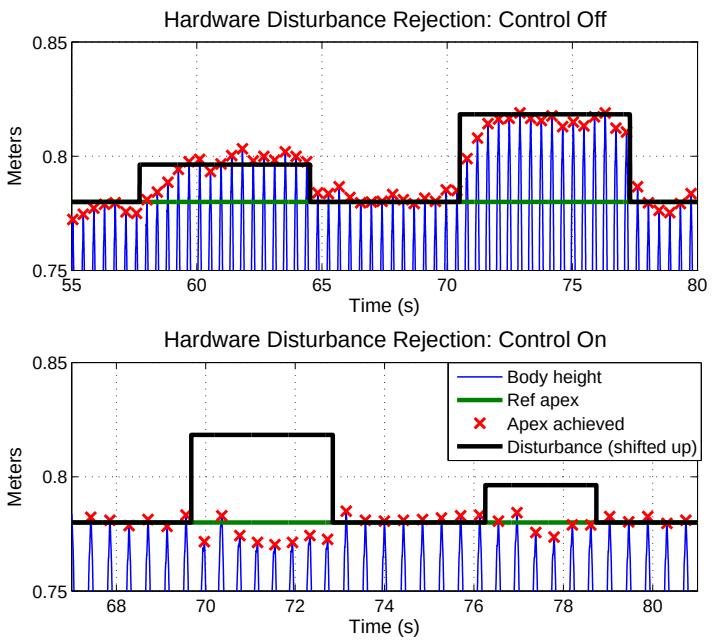


Figure 4.17: The control algorithm can easily be modified to reject ground disturbances on the hardware. By calculating a set of solutions and determining the ground level at impact, the robot is able to reject these kind of disturbances in real-time.

can predict the underactuated angular dynamics, and enforce them with PFL to regulate footholds.

Chapter 5

SLIP-based Step Length Control of SEA Hoppers

This chapter provides control strategies and algorithms for implementing step length control on our SEA 2D Hopper model. We illustrate how to apply our HOPFL leg controller so that algorithms designed for the SLIP model already present in the literature can be accurately implemented as reference trajectories. The body of the robot remains locked until chapter 5.5, thus we are considering step length control for the SEA 2D Hopper with dynamics in (2.14). In chapter 5.5, we build upon these results and augment the system with control laws for the hip torque in order to both stabilize body motions and force the resulting leg angle trajectories to be analytically tractable from SLIP. This implementation is a key result of this dissertation, and provides precise foothold regulation even in the presence of non-steady leg and body motions.

5.1 Review of SLIP Analytical Approximations

Our method builds upon recent work conduced by Piovan and Byl [28, 16], where assumptions on the trajectory of the leg state of the robot allow the normally unsolvable angular dynamics to be approximated analytically to high accuracy. Work in [28] considers dynamics of the Active SLIP, seen in (2.4). It is shown in [28] that if we can control exactly the leg-length trajectory to be a composite of sinewaves of the form $L(t) = r + a\cos(\omega t + \beta) + vt$, we can compute an accurate approximation of the leg-angle dynamics throughout stance as

$$\theta_{SLIP}(t) \approx \frac{1}{L(t)}(u_0(t) + \epsilon u_1(t) + \epsilon^2 u_2(t) + ...), \tag{5.1}$$

where the various $u_i(t)$'s for $i = 0, 1, \ldots$ are solutions of

$$\begin{aligned} &\frac{d^2}{dt^2}u_0 - \lambda^2 u_0 = 0, \\ &\frac{d^2}{dt^2}u_i - \lambda^2 u_i = -\delta u_{i-1}\cos(\omega t + \beta), \quad i = 1, 2, \ldots \end{aligned}$$

and the initial conditions for the above differential equations and constants r, v, λ, β, δ, ω, and ϵ are determined from the robot state at touch-down.

To force the SLIP's leg-length trajectory to be a composite of sinewaves, [28] loosely approximates the actuator's motion to be the sum of a piecewise linear function $L_{lin}(t)$ paired with a nonlinear component, $L_{nl}(t)$, as

$$\begin{aligned} L_a(t) &= L_{lin}(t) + L_{nl}(t) \\ L_{nl}(t) &= \frac{m}{k}(gsin(\theta(t)) - L(t)\dot{\theta}^2(t)) \end{aligned} \tag{5.2}$$

where we see $L_{nl}(t)$ is a state feedback term with the purpose of cancelling the nonlinear

terms of the L-dynamics in SLIP in (2.4), and forming the closed loop dynamics as a spring-damper system in addition to the second term. This control component is clearly fully determined by the states of the system throughout stance, and the second term $L_{lin}(t)$ is a design variable.

Generating a trajectory to drive the system to a desired state becomes then a problem of finding the right piecewise linear function $L_{lin}(t)$. It is important to note that the SEA actuator dynamics of L_a are not captured by the linear parametrization that we use to generate a reference trajectory for L. However, as will be shown in the next subsection, when we use our HOPFL leg controller to regulate one of these analytical solutions as a trajectory this is not a problem as long as the SEA remains within physical limits. Thus, given initial conditions L_0, $\dot{L}_{TD}^{-}$, θ_{TD} $\dot{\theta}_{TD}$ along with some $L_{lin}(t)$ we can analytically produce a set of stance phase trajectories $L_{SLIP}(t)$ and $\theta_{SLIP}(t)$. This calculation is extremely fast; on a modern computer each solution only takes roughly $10\mu s$ to compute.

5.2 Parameterization of SEA Motion and Reachable Space

When the body of FRANK is mechanically locked and the dynamics of the system are those we saw of the 2D SEA Hopper in (2.14), the angular dynamics ($\ddot{\theta}$) during stance are *nearly identical* to those of SLIP. This is not the case of the leg state dynamics ($\ddot{L}$), as the rate of compression of the leg is highly coupled to the SEA dynamics, and a given L_{lin} command may not be a feasible solution for the real SEA. However, using HOPFL on the leg state has the advantage of redirecting the problem to the dynamics of L, essentially ignoring the accuracy of the parametrization used to generate the trajectory for the leg. Thus, when using our HOPFL leg controller in (3.9) to regulate one of these SLIP

approximations as a reference trajectory, we expect $L(t) = L_{SLIP}(t)$ and $\theta(t) \approx \theta_{SLIP}(t)$, with $\theta_{SLIP}(t)$ calculated using (5.1).

If we parametrize L_{lin} as a function of a small numbers of parameters, the problem of searching for an actuation policy to yield some nominal $L(t)$ is equivalent to performing a search over all feasible values of such parameters. Starting from a fixed initial condition, i.e., apex state $\mathcal{A} = [y, \dot{x}]$ and θ_{TD} pair, one can compute the reachable space $\mathcal{R}$ of the system as the set of all apex states reachable in one step for any feasible motion of the SEA throughout stance. This is accomplished using our stance phase analytical approximations and forward solving the subsequent ballistic phase. As the reachable space is a function of the SEA motions, the particular parametrization of L_a, and consequently L_{lin}, used to solve the system's dynamics affects the shape and size of the computed space. Therefore, the parametrization used can most effectively be employed in control strategies when it does not excessively contract the reachable space. As shown in [28] and [16], to maximize the reachable space of the SLIP model, the actuator needs to be controlled throughout the entire stance phase. In [28], the actuator motion was parameterized using two switching time variables: times at which the actuator would instantaneously change velocity and actuate in a different direction (or stop completely). In order to populate the reachable space in an easier to visualize manner, we take a slightly different approach here.

We assume that the actuator moves at its maximum constant velocity $\dot{L}_{a,MAX}$ to reach a desired constant value, L_{a0}, and at some time t_s the actuator moves with maximum velocity $\dot{L}_{a,MAX}$ towards its upper or lower limit until the end of the stance phase. In general $\dot{L}_{a,MAX}$ should be set to approximate the mean saturation velocity of the actuator given a step change. In terms of the reachable space, increasing and decreasing $\dot{L}_{a,MAX}$ causes the total area to expand and contract, and must be set such that absolute bounds are not violated. For this work we used $\dot{L}_{a,MAX} = 0.5m/s$ to paramaterize

the SEA motion. Therefore, searching for an actuation policy to yield some nominal $L(t)$ is equivalent to performing a search over all feasible values of L_{a0} and t_s. As previously mentioned, the linear parametrization of L_a does not fully capture its nonlinear dynamics, as shown in Fig. 5.1. The parameters used for the approximated function are $L_{a0} = 0.03$ m, and $t_s = 0.12$ s. The 4th-order PFL has the added benefit of ignoring discrepancies in the actuator's dynamics versus its parametrization while driving the leg length dynamics, L, to the reference trajectory. The resulting trajectory of the SEA must, however, remain within physical limits of the robot. Note that FRANK (and any robot

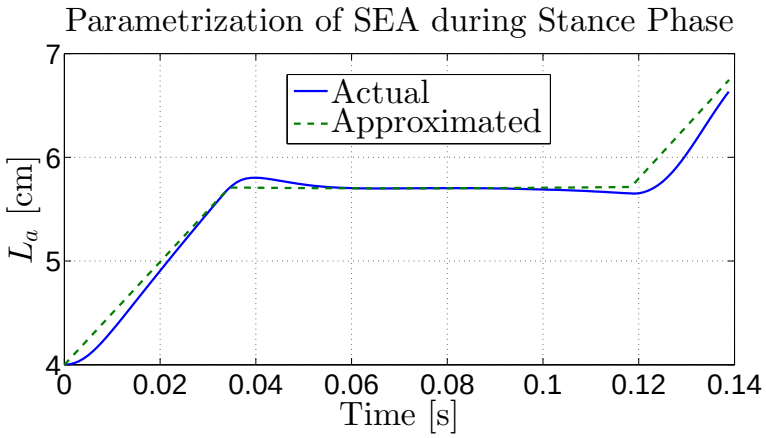


Figure 5.1: This figure shows an example of the approximated $L_a(t)$ computed as in (5.2) (dotted green line), used to generate the leg-length reference trajectory, versus the actual L_a computed from the system's equations of motion (2.14) (solid blue line). Note that in general we do not ever control the SEA directly, rather we use the approximate SEA trajectory to generate a coordinating leg length trajectory for use by the HOPFL controller.

in general) is subject to energy loss at impact due to the unsprung mass. The SLIP, being a simplified model with a massless leg, does not capture this aspect. It is then important to remember to artificially incorporate energy loss at the transition between phases, as described in Eq. 2.15 and 2.16. It is, however, quite trivial to incorporate these impact dynamics into our reachability calculations, as we simply must remove the appropriate amount of energy each time we construct a solution from our analytical calculations.

Fig. 5.2 shows the total reachable space from a set initial conditions, computed numerically by solving the system's dynamics (2.14) and with the parametrization of L_a discussed above. As we can see, the analytically calculated reachable space estimate captures a large amount of the real system's reachable space. It is worth mentioning that numerically solving the system's dynamics, especially when frictional and damping terms are present, is computationally taxing and cannot be carried out in real-time. In particular to Fig. 5.2, the real reachable space shown consists of approximately 5,000 points that are calculated from a brute force search of all possible L_a commands using an ODE solver, which takes several hours to simulate. In contrast, the analytical grid shown consists of a much smaller set of only about 600 points and takes less than one second to calculate. Hence, it is not possible to compute the actual reachable space of the system during real-time hopping, but it is possible to perform such computations when using the stance-phase approximation and the SEA parametrization. Although the approximation does not capture parts of the boundary of the true reachable space, our method retains approximately 61% of the reachable area and is several orders of magnitude faster to calculate. Since each apex state is simply a function of two decoupled parameters, grid calculations can make great use of parallel processing. Several parameters can be tuned to allow for a faster computation of the reachable space. In particular, the grid of L_{a0} and t_s can be computed at different resolutions to increase or decrease the accuracy versus the computation time.

The reachable space expressed as a function of forward velocity and apex height does not in itself fully characterize one dimension: the forward position, or, more interesting in terms of motion planning, the step length S. As defined in Fig. 2.1, $S = S_{TO}+S_{flight}+S_{TD}$ cannot be computed without a priori knowledge of the future touch-down angle $\theta_{TD,i+1}$. For any fixed pair $\theta_{TD,i}$ and $\theta_{TD,i+1}$, the step-length achievable in one step is not unique, but is a function of the actuator's parametrization $\{L_{a0},\, t_s\}_i$. Once a desired S is chosen

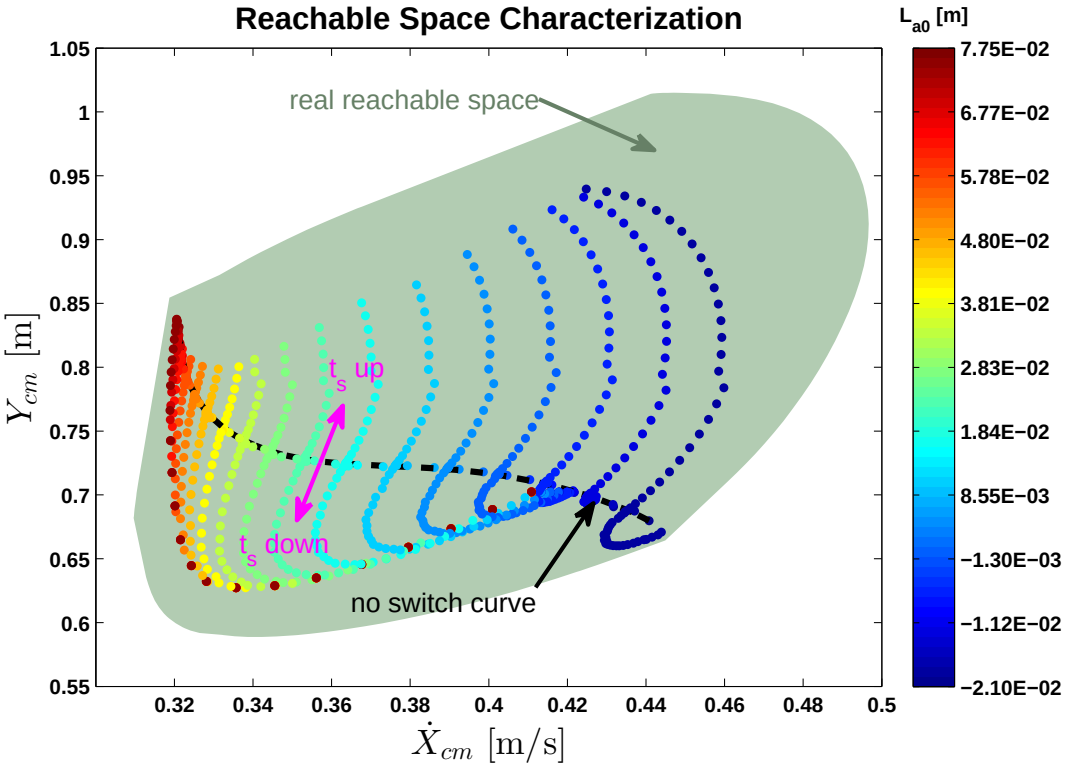


Figure 5.2: Above shows how the reachable space $\mathcal{R}$ can be parametrized by our SLIP-based technique, compared to the real reachable space of the system for an initial apex state of $(\dot{x}_{cm}, y_{cm}) = (0.4m/s, 0.75m)$. The colors denote the discrete values of L_{a0} used to compute the grid, and the black line denotes apex states resulting in commanding a constant L_{a0} without any switches, where the magenta arrows illustrate how the apex states change as t_s decreases to zero and actuates the SEA in both positive and negative directions.

within the set of achievable step-lengths $S \in [S_{min}, S_{max}]$, we can look at the curve of possible apex states within the reachable space that results in that step-length, as shown in Fig. 5.3. In general, we can choose the desired state based on particular considerations. For example, we may want to maintain the robot's gait as close as possible to a certain reference in terms of forward speed and/or apex height. The convex hull is also shown in black and can be used to very quickly calculate the approximate centroid of the reachable space $\mathcal{C}$, shown as the black X. Calculation of the approximate centroid location can be accomplished with an extremely small amount of grid points.

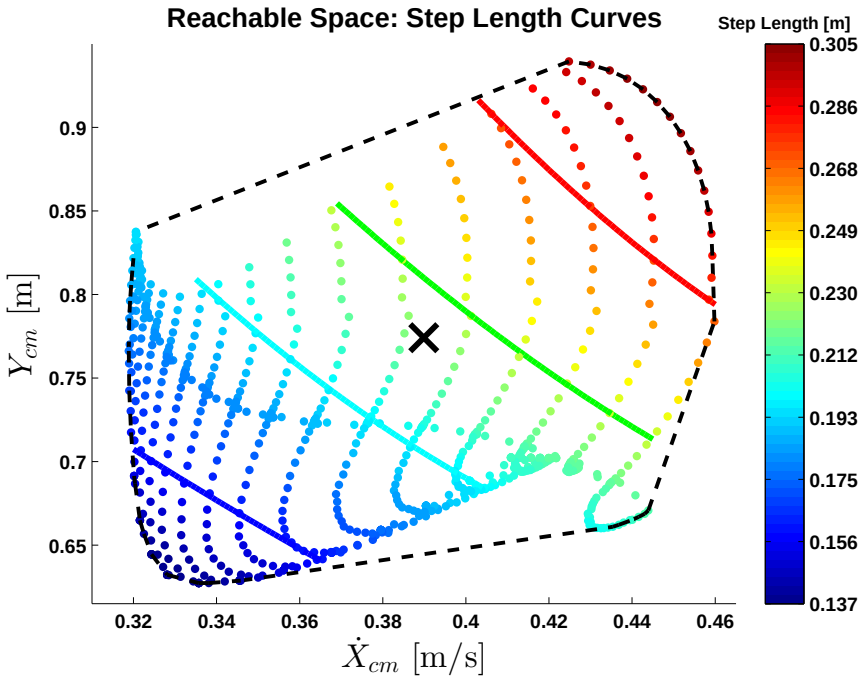


Figure 5.3: Above shows the parametrization of the reachable space $\mathcal{R}$ for an initial condition of $(\dot{x}_{cm}, y_{cm}) = (0.4m/s, 0.75m)$ and next touch-down angle $\theta_{TD} = -3\,\text{deg}$ with the step length S shown in color. Here we see that for each step length within the reachable space several specific apex state solutions exist and evolve on curves, with a few examples shown specifically above.

5.3 Touch-down Angle Control Methods

During each flight phase, given initial conditions $\mathcal{A}_{i-1}$ and $\theta_{TD,i}$ and a desired step-length S, one needs to compute the actuation's parametrization $\{L_{a0},\, t_s\}_i$ *and* the future touch-down angle $\theta_{TD,i+1}$ that results in the desired step-length S_i (and therefore also apex state $\mathcal{A}_i$) to be within some neighborhood of a reference state. This means all touch-down angle reference updates must happen at the same time as the trajectory generation for the HOPFL leg state occurs, before TD_i. The simplest method we consider for setting touch-down angles is to implement a slightly more accurate version of the Raibert-inspired method from (4.7) as

$$\theta_{TD} = \theta_0 + K_P(\dot{x}_r - \dot{x}_n) + K_D(-\dot{x}_n + \dot{x}_{n-1}) + K_I \sum_{n=1}^{N_{hops}} (\dot{x}_r - \dot{x}_n) \tag{5.3}$$

where θ_0 is a constant touch-down angle with forward velocity reference $\dot{x}_r$. The term $\dot{x}_n$ represents the discrete measurement of the forward velocity taken at apex $\mathcal{A}_n$. This control method is essentially the same as (4.7), but with the added derivative and integral terms. This method has been used historically due to it's simplicity to implement. The usage of simplistic touch-down angle control such as (5.3) limits the reachable space at each step by forcing all possible solutions of a given step-length to be within some neighborhood of a forward speed reference $\dot{x}_r$. Additionally, such simple methods require knowledge of compatible $(\theta_{TD,i+1}, \dot{x}_r)$ pairs that typically must be experimentally measured and tuned along with feedback gains. It will be shown later that we *can* still obtain reasonable results using this method, but the system can be destabilized if too aggressively changing step references are chosen.

We next present a reachability-based method for choosing touch-down angles and actuation's parametrization that allow for any specific step-length solution within the

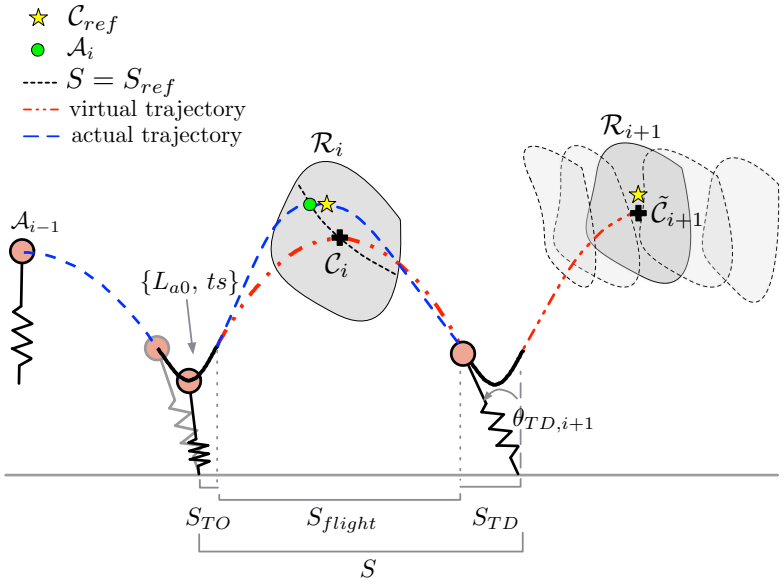


Figure 5.4: The touch-down angle is calculated by performing a binary search over $\mathcal{R}_{i+1}$ (translucent polygons) in order to align the future centroid $\tilde{C}_{i+1}$ with centroid reference $\mathcal{C}_{ref}$ (yellow star). The actual apex state visited $\mathcal{A}_i$ (green circle) will always be along the corresponding step length curve to S (dashed line) and is chosen to be the point horizontally closest to $\mathcal{C}_{ref}$.

current reachable space to be chosen (even randomly) at each step, and provide an algorithm to regulate the desired step lengths. The key property this method makes use of is the fact that varying $\theta_{TD,i+1}$ biases the velocity range of the next reachable space, thus causing $\mathcal{R}_{i+1}$ to translate along the velocity axis based on the chosen touch-down angle. More details on how varying the touch-down angles affects the reachable space can be seen in Appendix D. At the i-th flight phase, starting from $\mathcal{A}_{i-1}$ and $\theta_{TD,i}$, the choice of the future $\theta_{TD,i+1}$ to achieve the desired step-length S_i is performed as follows (refer to Fig. 5.4).

In order to regulate step length S_i, the next touch-down angle $\theta_{TD,i+1}$ must be computed from apex state $\mathcal{A}_{i-1}$, as $\theta_{TD,i}$ is defined from the previous step and $\mathcal{R}_i$ is completely deterministic from $\mathcal{A}_{i-1}$. Thus, the position of $\mathcal{A}_i$ depends on the future actuator's parametrization that is computed as a function of $\theta_{TD,i+1}$, and therefore cannot be computed directly from $\mathcal{A}_{i-1}$. However, the future apex state is an element of the reachable

space of the current apex state and touch-down angle: $\mathcal{A}_i \in \mathcal{R}_i$. As previously discussed, $\mathcal{R}_i$ can be estimated using the closed-form approximation of the stance phase dynamics and a coarse grid $\{L_{a0},\, t_s\}$. In particular, we can compute the centroid $\mathcal{C}_i$ of $\mathcal{R}_i$, as seen in Fig. 5.3, using a further reduced grid to only define the convex hull. We can conservatively assume the expected future apex to be $\tilde{\mathcal{A}}_i = \mathcal{C}_i$. From $\tilde{\mathcal{A}}_i$ we can calculate the future centroid $\tilde{C}_{i+1}$ once we set a touch-down angle, thus we perform a binary search over a broad range of feasible touch-down angles. The binary search chooses $\theta_{TD,i+1}$ computing the reachable space from $\tilde{\mathcal{A}}_i$, $\mathcal{R}_{i+1}$, whose centroid minimizes the distance with respect to a target centroid $\mathcal{C}_{ref}$:

$$\theta_{TD,i+1} = \arg\min_{\theta_{TD}} \parallel \tilde{C}_{i+1} - \mathcal{C}_{ref} \parallel_2 . \tag{5.4}$$

In other words, the touch-down angle is chosen to align the future centroid of the reachable space to a target reference centroid. The target centroid $\mathcal{C}_{ref}$ is typically chosen to be constant, but can be set to transition the system to different energy states. Note that generally $\mathcal{A}_i \neq \tilde{\mathcal{A}}_i$. The guess apex $\tilde{\mathcal{A}}_i$ is used to compute $\theta_{TD,i+1}$, while the actual apex $\mathcal{A}_i$ will be a function of L_{a0} and t_s that will be computed next. However, because both apex states belong to $\mathcal{R}_i$, their distance (i.e., the error on the guess of the future state) is bounded by the size of $\mathcal{R}_i$.

5.4 Step Length Algorithm with Body Locked

We now provide our complete algorithm used to precisely regulate footholds that can be chosen randomly from within the reachable space of the system at each step. Starting at each take-off, we perform a minimization algorithm during the flight phase to find the values of the SLIP-based trajectory parameters, L_{a0} and t_s, that minimize the cost

function

$$J = |S_i - S_{ref}|, \tag{5.5}$$

where S_i and S_{ref} are the achieved and desired step-lengths respectively, which are a function of both the ballistic path and angle values at take-off and touch-down. To test our algorithm, the foothold reference S_{ref} is chosen *randomly* from within the reachable space at each step.

The algorithm proceeds as follows (refer to Fig. 5.4).

i. From $\mathcal{A}_{i-1}$ or during flight, $\theta_{TD,i+1}$ is computed using the binary search on the centroid of the estimated future reachable space, $\tilde{\mathcal{C}}_{i+1}$ minimizing (5.4).

ii. The reachable space $\mathcal{R}_i$ is computed using the closed-form approximation of the stance-phase dynamics for a coarse grid of the parameters $\{L_{a0},\, t_s\}$.

iii. From TO_i we can backsolve the ballistic flight phase and system impact dynamics and find the set of states, forming a 1D curve, that give the reference step-length S_{ref}:

$$\mathcal{L}_S = \{\mathcal{A} \in \mathcal{R}_i \mid S_i = S_{ref}\},$$

where $\mathcal{L}_S$ represents the $S = S_{ref}$ curve in Fig. 5.4. Defining $y_{\mathcal{L}_S}$ and y_{ref} as the y-component of $\mathcal{L}_S$ and $\mathcal{C}_{ref}$ respectively, we choose the policy

$$\{L_{a0}^S, t_s^S\} = \underset{L_{a0},\, t_s}{\arg\min} |y_{\mathcal{L}_S} - y_{ref}|,$$

which represents the actuation pair in the $\{L_{a0}, t_s\}$-grid which results in the point in the $\mathcal{L}_S$ curve closest to our target state. Note that many solutions are possible here, but we specifically choose to be closest to the y-component of the target state since the $\dot{x}$-component is already aligned using the touch-down angle controller.

iv. The optimal policy that minimizes the cost function J in (5.5) is found using a Nelder-Mead constraint optimization algorithm as

$$\{L_{a0}^{opt}, t_s^{opt}\} = \arg\min_{L_{a0}, t_s} J,$$

initialized at $\{L_{a0}^S, t_s^S\}$, which is the actuation pair from the coarse-grid approximation computed in the previous step.

v. L_{a0}^{opt} and t_s^{opt} are used to compute the leg-length reference trajectory L_{ref}, and the 4th-order PFL (3.10) is implemented at touch-down.

Thanks to the existence of an accurate closed-form approximation for the stance phase dynamics, both the reachable space computation and the optimization search can be performed with low cost in terms of computation time. While generally the cost function J is not convex over the entire $\{L_{a0}, t_s\}$-space, the global minimum will reasonably be expected to be in a neighborhood of $\{L_{a0}^S, t_s^S\}$. Thus, initializing the optimization function at $\{L_{a0}^S, t_s^S\}$ guarantees, within accuracy of the coarse grid used, that the solution found will be the global minimum. Additionally, $\{L_{a0}^S, t_s^S\}$ converges to $\{L_{a0}^{opt}, t_s^{opt}\}$ as the number of points in the grid used to compute the reachable space at step (ii) tends to infinity. Therefore, if the $\{L_{a0}, t_s\}$-grid is fine enough with respect to a set tolerance for the error of the solution found, step (iv) can be omitted. We next discuss accuracy of foothold placement, feasibility of computation time, and stability of the reachable space.

5.4.1 Simulated Step Length Regulation Results

The control strategies discussed were implemented in simulation using an ODE solver to numerically compute the dynamics for the SEA 2D Hopper (i.e., FRANK with a locked body) in (2.14), along with system parameters from Table 2.1. The simulations were per-

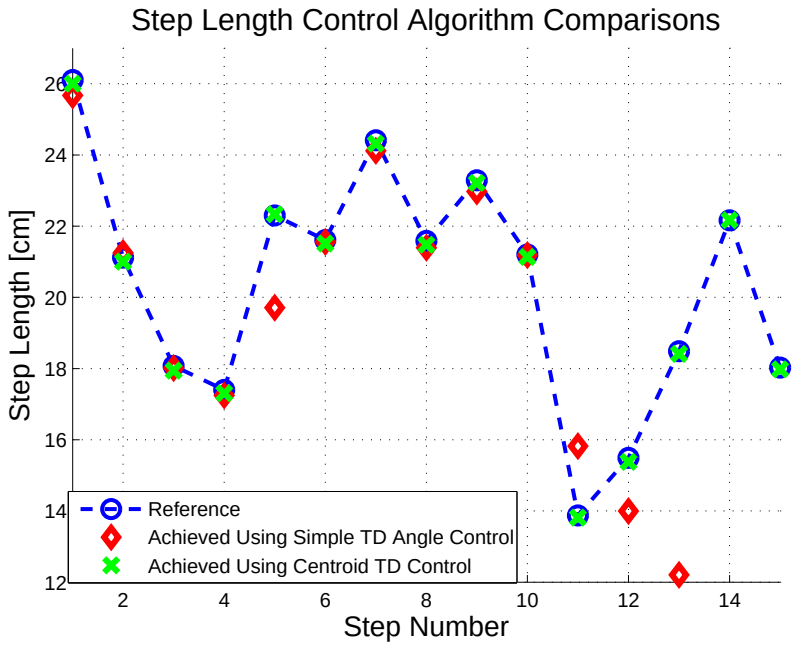


Figure 5.5: Above are simulation results for the control algorithm using the simple feedback based touch-down angle controller (red diamond) from (4.7) and the centroid-based TD angle controller (green X). Here in particular at step numbers 5 and 11 we see fast changing reference commands outside of the simple controller's reachable space, eventually resulting in failure. In contrast, controlling the touch-down angle using our centroid-based method results in both a higher reachable space at each step and the ability to track fast step-to-step varying footholds accurately.

formed using an Intel i7-2600 CPU, running all 8 logical cores to make use of parallel processing available in our algorithms. We can use our step-length control algorithm to track footholds that change quickly on a step-to-step basis. This is contrasted with simpler, classical techniques where the touch-down angle is traditionally set by using feedback of the forward speed measurement, as in (4.7). Data in Fig. 5.5, compare several simulated steps for both control methods, where the foothold reference is chosen randomly at each step from within the reachable space. It is clear the centroid-based touchdown controller performs drastically better, as the simpler touch-down angle controller is typically suited for regulating steady-state motions and does not have the advantage of having a large reachable space at each step.

5.4.2 Centroid Shifting

For the majority of the simulation results presented in this work, we consider simply commanding the target centroid location $\mathcal{C}_{ref}$ to remain constant throughout all step commands. Specifically, we command $\mathcal{C}_{ref} = (0.75m, 0.4m/s)$ as this was an apex state we were able to achieve during hardware experiments with FRANK, without bottoming out the spring or exceeding the actuator's limits. Commanding a reference centroid means we expect the actual apex states $\mathcal{A}$ visited by the robot at each step to be within some neighborhood of $\mathcal{C}_{ref}$, as the particular solution picked is randomly chosen from within the reachable step-lengths. It is,however, possible to shift the location of the centroid, which we illustrate in Fig. 5.6. Here we see the actual apex states $\mathcal{A}$ chosen for particular step length commands can be biased by choosing different reference centroids $\mathcal{C}_{ref}$. While generally this does not change the total number of possible step commands for a given apex state, it does allow for changes in the mean step-lengths by transitioning the system to lower or higher energy states. Thus, a higher planner could consider this as an additional control variable for regulating set step sequences. In fact even if a simpler method is used to generate the touch-down angles, we can still bias the overall energy level via choice of y_{ref} in algorithm step iii.

5.4.3 Effect of Reachability Grid Size

Both the accuracy of the achieved footholds and the controller computation time are functions of the total number of points analytically calculated, N_{grid}, i.e., the sum of the points used in the $\{L_{a0}, t_s\}$-grid and the maximum number of points used in the binary search-based touch-down angle update calculation. The grid size (resolution) must be set low enough to achieve a feasible computation time, with the ballistic time varying depending on the step length chosen at each hop. Typical ballistic durations for our

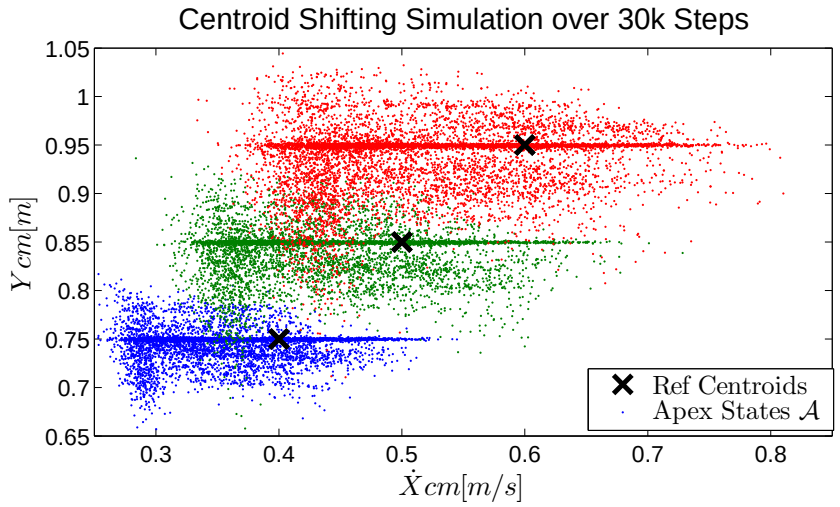


Figure 5.6: Shifting the target centroid allows us to bias the overall energy level of the reachable space. Three different centroid references $\mathcal{C}_{ref}$ are commanded during a simulation of 30k steps (X's), and as expected the y_{cm} component of the particular step length apex state $\mathcal{A}$, the actual apex state visited by the robot (S-curve soln), is within some neighbourhood of the corresponding y_{ref} component of the commanded centroids $\mathcal{C}_{ref}$. It is important to note that the variance in the apex states here do not affect step length accuracy as long as the step command is within the reachable space at each step.

system vary between 300ms-500ms. We define the *normalized computation time* to be

$$\rho_{fl} = 100 \frac{\text{computation time}}{\text{flight time}}$$

To be computationally feasible all computations must be accomplished during flight, thus we require $\rho_{fl} < 100\%$. Several simulation sets shown in Fig. 5.7 were performed consisting of 5,000 randomly chosen step commands, where the reachability grid size N_{grid} was varied, and the distributions of the normalized computation time and of the step-length error were measured. Here we see that we must select N_{grid} appropriately small and expect some small step-length error at each step. Particular to our computational hardware, we must choose $N_{grid} < 120$, and expect mean step-length errors of around 0.6%.

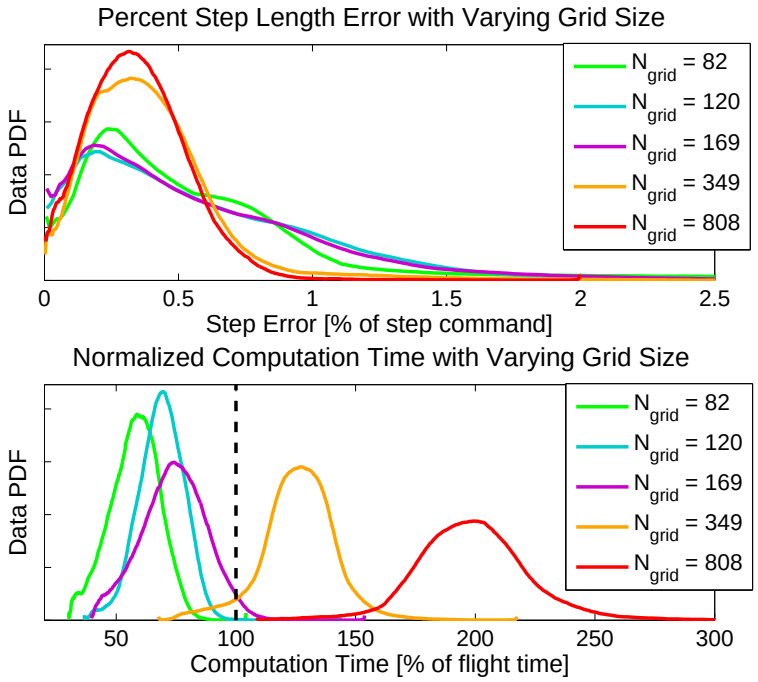


Figure 5.7: Each set of data above corresponds to a simulation of 5k steps, and here we see the distribution of step length error (top) and normalized computation time (bottom) as a function of varying grid size. We see that while the computation time decreases dramatically as we reduce the grid size, the variance of the step error only increases by a fairly small amount. We must always choose a grid size such that the normalized computation time is less than the flight phase duration (black dotted line).

5.4.4 Stability

The proposed control action drives the system to a non-steady-state motion, where the step-length is changed at each hop. In particular, our control policy varies the step-length by choosing a random value; if said value were not to be achievable in one step, the closest achievable is chosen. It is not generally possible to globally assert stability of systems whose trajectory does not converge to a constant state nor to a limit cycle, although successful simulation results for 30,000 steps from Fig. 5.6 with randomized choice of the desired step-lengths at each hop statistically suggest a level of stability for our system. Therefore, the following stability test is performed. From a set of 300 initial conditions

and 100 consecutive hops, we command a step length as a fixed percentage of the step-length range achievable at each step. For example, at each hop we determine the range of step-lengths achievable $[S_{min}, S_{max}]$ and we command S as $S = \alpha(S_{max} - S_{min}) + S_{min}$, where α is a fixed percentage in [0, 1]. Simulation results show that for higher values of α, the systems converges to a limit cycle. For lower values of α, however, the system oscillates between several low energy apex states. This is due to the fact that for low step-lengths, the solution is for the system to reach a low energy state, whose reachable space has no intersection with the desired apex height given by $\mathcal{C}_{ref}$ (forward velocity is controlled via choice of θ_{TD}). As a result, the solution tends to periodically oscillate between low energy apex states. Despite the oscillation, the solution remains bounded. A consideration stemming from this stability experiment is that for any initial condition chosen in the initial ball in Fig. 5.8, every future hop will stay within the ball for several values of α. At each hop, the original state-space shrinks to a convergence set.

5.5 Approximation of Body Angle Dynamics

We now consider extending our SLIP-based step length regulation algorithm to the robot FRANK with the body unlocked, with dynamics in (2.20). We have thus far intentionally left the hip angle actuator u_{hip} completely unused (i.e., passive) during the stance phase. We now consider the problem of designing SLIP-based trajectories for the leg state and augmenting them to be implemented concurrently with non-zero hip actuator control laws, which are required to provide body stability. As was shown originally by Raibert in [38], the body dynamics can be condensely expressed as

$$J\ddot{\phi} = F_t(t)l_1 sin(\phi - \theta) - F_n(t)l_1 cos(\phi - \theta) + \tau_{hip} \tag{5.6}$$

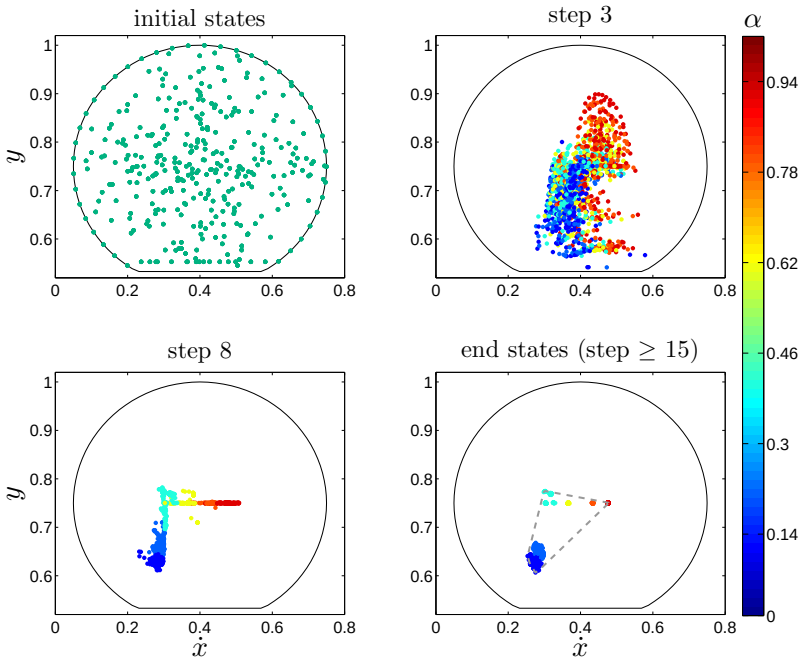


Figure 5.8: This figure shows the progression of the reached apex state after 3, 8, and over 15 hops (end states), starting from a pool of initial apex states randomly picked inside and on the edge of a closed set. After a relatively small number of hops the solution converges to a steady-state limit cycle or to a periodical oscillatory state. As we show here, starting from a closed set we converge to a smaller, closed set.

where l_1 is the body CoM displacement from the hip and $F_t(t)$ and $F_n(t)$ are forces acting at the hip between the leg and body, and are generally non-linear functions of the states of the robot. Here we see clearly when the body CoM is located at the hip (i.e., $l_1 = 0$), the dynamics become trivial, as was the case in [28]. In order to develop a suitable control method, we must first employ some approximations.

Our body stabilization methods largely rely on these approximations being accurate. The model parameters published for the 2D Raibert hopper, seen in [38], are quite clearly not the actual parameters of their robot, as the values are typically only represented to one significant figure. Such accuracy only captures the approximate order of magnitude of the model parameters, which was sufficient for their methods (which were therefore

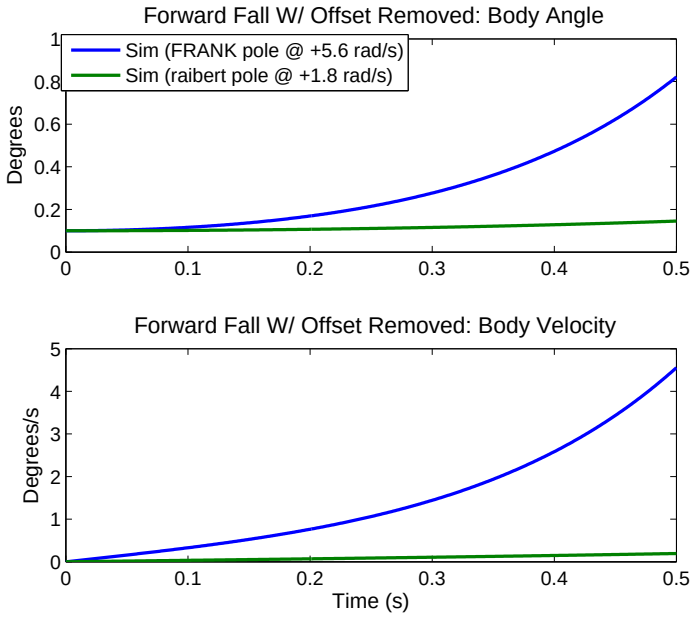


Figure 5.9: Simulation results from a non-zero initial condition of the linearized body dynamics of FRANK vs the Raibert hoppers. As shown, FRANK's linearized system has a significantly more unstable pole. This simulation is only used to note that FRANK is more difficult to stabilize, and the difference between accurate and heuristic model parameters result in different levels of stability; we do not linearize the dynamics for our control strategies.

not very accurate). In contrast, our model parameters for the body and leg dynamics are obtained from system identification, seen in Table 2.1, the details of which can be seen in Appendix A. It is of note to mention that FRANK is a significantly more difficult system to stabilize than the Raibert hopper using the published parameters. This can be easily visualized by locking the leg in simulation and linearizing the dynamics of the body about the vertical. As we see in Fig. 5.9, the body of FRANK destabilizes much faster than the Raibert hopper.

5.5.1 Approximation of Body Angle Dynamics: Flight

During the flight phase, all spring forces are zero, simplifying the dynamics considerably. However, due to the non-zero leg mass and inertia, when we move the leg during

flight to position it in some nominal touch-down angle it will inevitably impart torques on the body. This means the dynamics of ϕ during flight are a function of ϕ, $\dot{\phi}$, θ, $\dot{\theta}$ and u_{hip}. During the flight phase we use a simple PFL controller to re-position the leg to the next touch-down angle associated with our stance phase initial conditions. Thus, we can determine the future actuator input as

$$u_{hip,flight} = \frac{1}{\beta_\theta}(-\epsilon_\theta + \ddot{\theta}_{fl}) \tag{5.7}$$

where the PFL coefficients β_θ, ϵ_θ are constructed from the flight phase dynamics of FRANK. The flight phase trajectory of the leg angle, $\theta_{fl}(t)$, is a design choice with the only requirement being that the terminating value is the next desired touch-down angle. Thus, if we assume the leg angle is controlled to follow some nominal trajectory $\theta_{fl}(t)$, we can write the flight phase body dynamics as

$$\ddot{\phi}_{fl} = f_{fl}(\phi, \dot{\phi}, \theta_{fl}, \dot{\theta}_{fl}, \ddot{\theta}_{fl}) \tag{5.8}$$

We next employ small angle approximations on all trigonometric terms which involve θ, ϕ,$\phi - \theta$, and $2\phi - 2\theta$. To force the dynamics to be analytic given θ_{fl}, we remove the three high order terms containing ϕ and $\dot{\phi}$, resulting in

$$\begin{aligned} \ddot{\phi}_{fl} &\approx \frac{c_1\ddot{\theta}_{fl} - 2l_1 m_B m_l l_0 \dot{\theta}_{fl}^2 \theta_{fl}}{c_2} \\ c_1 &= -4J_l m_B + 4J_l m_l + m_B m_l l_0^2 + 2l_1 m_B m_l l_0 \\ c_2 &= 4m_B m_l l_1^2 + 2m_B m_l l_0 l_1 + 4Jm_B + 4Jm_l \end{aligned} \tag{5.9}$$

where the terms omitted are $2l_1 m_B m_l l_0 \theta_{fl} \dot{\phi}^2$, $2l_0 l_1 m_B m_l \dot{\theta}_{fl}^2 \phi$, and $2l_0 l_1 m_B m_l \dot{\phi}^2 \phi$ which we expect to be quite small. Thus, we need only define a smooth trajectory for θ_{fl} and

its derivatives, and then given body initial conditions we can use (5.9) to compute the resulting body trajectory during flight with low computational cost. All flight phase trajectories $\theta_{fl}(t)$ used in this work are simply constructed to be smooth curves that terminate with a desired touch-down angle and zero angular velocity by the end of the flight phase.

5.5.2 Approximation of Body Angle Dynamics: Stance

The stance phase body dynamics are considerably more complex due to the time varying leg length. However, since we have developed a control law that acts directly on the leg state, we have the advantage of knowing the leg length trajectory a priori. Thus, we assume our HOPFL in (3.9) is regulating the leg state to a nominal SLIP-based trajectory, therefore $L(t) = L_{SLIP}(t)$ and is analytically available to us. The approximation procedure parallels the previous Section, where we use small angle approximations and remove higher order terms. The dynamics for the leg and body angle during the stance phase can be approximated as

$$\begin{aligned} \ddot{\theta} &\approx 2\frac{\dot{L}\dot{\theta}}{L} - \frac{g}{L}\sin\theta + \Omega_\theta(L)u_{hip} + \xi(L)(\phi - \theta) \\ &= \ddot{\theta}_{SLIP} + \Omega_\theta(L)u_{hip} + \xi(L)(\phi - \theta) \end{aligned} \tag{5.10}$$

$$\ddot{\phi} \approx \Omega_\phi(L)u_{hip} - \frac{L}{l_1}\xi(L)(\phi - \theta) \tag{5.11}$$

where Ω_ϕ, Ω_θ, ξ are non-linear differentiable functions of L and we immediately see part of the approximated leg angle dynamics are in fact those of SLIP from (2.4). The approximations for both stance and flight are reasonable as we do not expect our SLIP-inspired leg angle trajectories to be large during operation, and our control strategies will aim to keep ϕ small, though we do expect some small error. Fig. 5.10 shows our

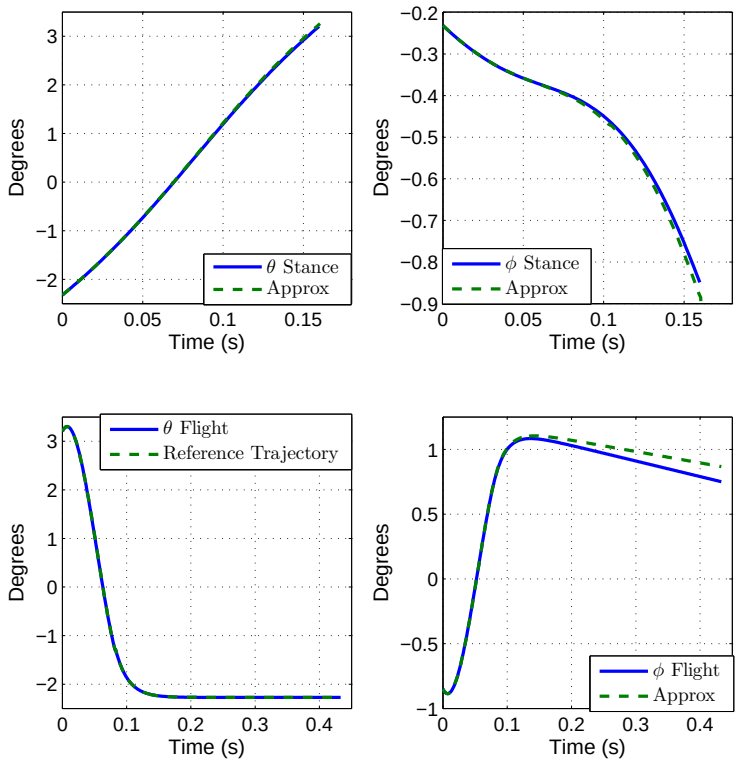


Figure 5.10: Above shows our analytical approximations (green dashed line) for the leg-body coupling dynamics plotted against full dynamical simulations (solid blue line), with stance phase approximations are shown on the top, and the flight phase approximations at the bottom. The reference trajectory for the leg angle in flight is simply a smooth curve terminating at a desired touch-down angle with zero angular velocity by the end of flight.

approximations for both stance and flight, where it is clear they perform quite well for the angles our systems operates at. In this particular example, the robot is moving forward at approximately $0.5m/s$.

5.6 Augmentation of Hip Torque Input

In the simplified but similar case in [28], a constant torque input was implemented to provide body stabilization, allowing all SLIP-based trajectories to be modified with

respect to only one variable that need be searched over and found. Although we cannot simply command a constant torque in this case, we can force the mapping from $\theta_{SLIP}(t)$ to $\theta(t)$ to also be a function of only one variable. We proceed by defining our control law for the hip input as

$$u_{hip} = \frac{1}{\Omega_\theta}(-\xi(\phi - \theta) + \tilde{u}) \tag{5.12}$$

Where $\tilde{u}$ is a constant input term. The purpose of this control law is to force the approximate dynamics of the leg angle in (5.10) to converge to $\ddot{\theta} = \ddot{\theta}_{SLIP} + \tilde{u}$, and since we also have $L(t) = L_{SLIP}(t)$ this results in

$$\theta(t) = \theta_{SLIP}(t) + \frac{1}{2}\tilde{u}t^2 \tag{5.13}$$

Where $\theta_{SLIP}(t)$ is calculated using our approximation in (5.1). Since all terms in (5.12) are functions of only state variables and constants, we can easily define the first two derivatives and implement it with our HOPFL leg controller in (3.9). Next we consider the resulting body dynamics by combining (5.12), (5.13), and (5.11) resulting in

$$\ddot{\phi} = \frac{\Omega_\phi}{\Omega_\theta}\tilde{u} - \xi(\frac{L}{l_1} + \frac{\Omega_\phi}{\Omega_\theta})(\phi - \theta_{SLIP}(t) - \frac{1}{2}\tilde{u}t^2) \tag{5.14}$$

Since we can generate solutions for $L_{SLIP}(t)$ and $\theta_{SLIP}(t)$, the only term preventing us from integrating (5.14) to generate our solution is the term ϕ itself. Therefore, we perform the following. We first evaluate (5.14) with the assumption $\phi(t) = \phi_0$, perform two integrations using stance phase initial conditions and generate a first order guess $\tilde{\phi}(t)$. Then, we recursively repeat the process by evaluating (5.14) with $\phi(t) = \tilde{\phi}(t)$. Fig 5.11 illustrates that this method is quite accurate after only one additional iteration, thus given $L_{SLIP}(t)$, $\theta_{SLIP}(t)$, and $\tilde{u}$ we can approximate $\phi(t)$ with two total evaluations of (5.14) along with four total integrations.

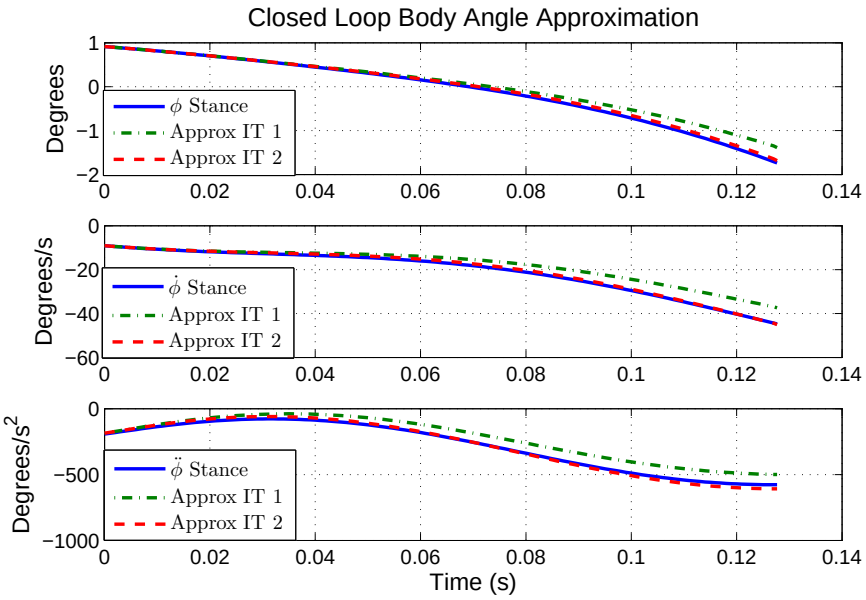


Figure 5.11: Using the control law in (5.12) allows us to approximate the stance body angle dynamics using only initial conditions, a corresponding SLIP trajectory, and $\tilde{u}$. The actual trajectories for the body (solid blue curve) are generated by using an ODE solver to simulate the full stance dynamics. In this example, the approximations shown are generated by using (5.13) to analytically compute $\theta(t)$ and integrate (5.14), which requires only two total iterations to achieve good accuracy.

The purpose of the term $\tilde{u}$ is to provide a constant input variable that can be algorithmically searched over to find solutions that provide body stabilization, similar to [28]. We need only search over L_{a0}, t_s and now also $\tilde{u}$ to find solutions that both result in desired step lengths and keep the body angle well behaved. To fully define the reachable space, both the body and body angular velocity must now be characterized. However, we are only concerned with ensuring the body angle remains small during operation. Fig. 5.12 shows an example of the reachable space as a function of the new input variable $\tilde{u}$ in the colorbar. Here we see for certain values of $\tilde{u}$, a slice of the reachable space results in small body angles at touch-down and a 2D planar set quite similar to the body locked dynamics. If we keep the body angle small enough, the body angular velocity will also remain small, as illustrated by Fig. 5.13.

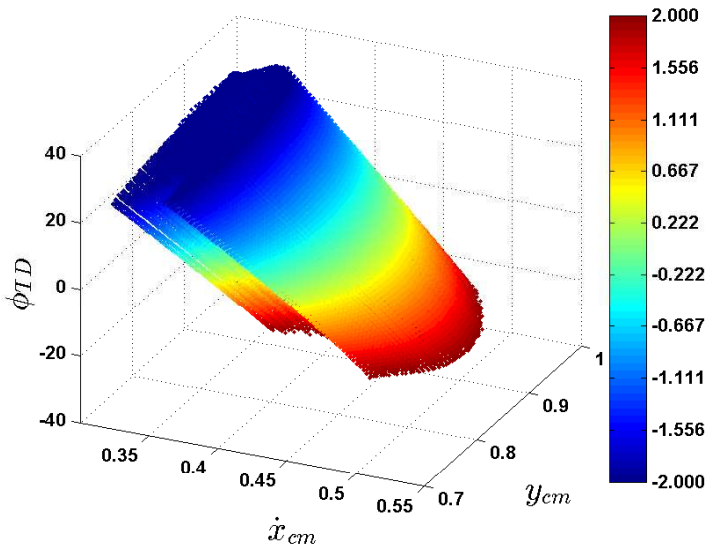


Figure 5.12: The reachable space is significantly more difficult to visualize when the body of the robot is unlocked. In this figure we show the next touch-down angle on the z-axis instead of the apex state as it is better coupled to step length. The colorbar represents iterated values of $\tilde{u}$, and we see a large cross-section of the reachable space exists where the body angle remains small at the next touch-down.

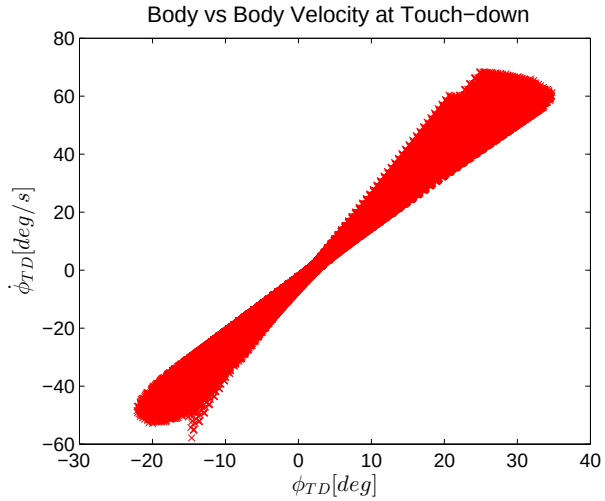


Figure 5.13: In order to fully define the reachable space, both the body angle *and* angular velocity must be defined. However, since our control strategies aim to keep the body angle small during stance, and the system is ballistic during flight, the relationship between the body and body velocity (and $\tilde{u}$) at touch-down can be approximately represented as linear. Note that while this means we do reduce our set of possible states, we can still satisfy the requirement of keeping the body angle and velocity small during operation.

5.7 Step Length Algorithm with Body Unlocked

For each set of initial conditions and input parameters L_{a0}, t_s, and $\tilde{u}$, we can analytically compute estimates of the stance and flight phases of the system using the control and approximation methods presented in this paper. We now provide an example application of our control strategy to generate and enforce trajectories on the leg state for 2D hopping. In particular, we control our system to hop following a reference trajectory of step-lengths, i.e., the distance between two consecutive footholds. At each take-off we use a minimization algorithm to find the values L_{a0}, t_s, and $\tilde{u}$ that minimize the same cost function as before:

$$J = |S_i - S_{ref}|, \tag{5.15}$$

where S_i and S_{ref} are the achieved and desired step-lengths respectively, which are a function of both the ballistic path and angle values at take-off and touch-down. The algorithm procedure mostly parallels the body locked case seen in chapter 5.4 and is as follows.

i. From initial conditions, the $(i+1)-$th touch-down angle is computed as in (4.7), or using the centroid-based method.

ii. The reachable space for the system at step (i), $\mathbf{R}_i$, is computed using the closed-form SLIP-based approximations along with body angle approximations for a coarse grid of the parameters L_{a0}, t_s, and $\tilde{u}$.

iii. From TO_i we can backsolve the flight phase using ballistic dynamics and (5.9). We remove all points from $\mathbf{R}_i$ that result in the next touch-down body angle being larger than a threshold to generate $\hat{\mathbf{R}}_i$:

$$\hat{\mathbf{R}}_i = \{\mathbf{R}_i \ \mid \ |\phi_{TD,i}| < \phi_{thresh}\},$$

iv. We next find the set of states that give the reference step-length S_r:

$$[y_S,\, \dot{x}_S] = \{[y,\, \dot{x}] \in \hat{\mathbf{R}}_i \mid S_i = S_r\},$$

For each step length, there exists a curve of possible solutions. We choose the policy

$$\{L_{a0}^S, t_s^S, \tilde{u}^S\} = \underset{L_{a0},\, t_s,\, \tilde{u}}{\arg\min} |y_S - y_r|,$$

which represents the point in the $\{L_{a0}, t_s, \tilde{u}\}$-grid closest to some target state y_r, which gives us some control of the overall energy level at apex.

v. Because $\{L_{a0}^S, t_s^S, \tilde{u}^S\}$ is computed from a coarse grid, the optimal policy that minimizes the cost function J in (5.5) is found using a Nelder-Mead constraint optimization algorithm as

$$\{L_{a0}^{opt}, t_s^{opt}, \tilde{u}^{opt}\} = \underset{L_{a0},\, t_s,\, \tilde{u}}{\arg\min} J,$$

initialized at $\{L_{a0}^S, t_s^S, \tilde{u}^S\}$ with constraint $|\phi_{TD,i}| < \phi_{thresh}$.

vi. L_{a0}^{opt}, t_s^{opt}, and $\tilde{u}^{opt}$ are used to compute the leg-length reference trajectory L_{ref}, and the HOPFL in (3.9) along with the hip controller in (5.12) is implemented at touch-down.

Fig. 5.14 shows simulation results of our control strategy for a set of desired step lengths. The touch-down angles are set using the simple control law in (4.7), with $\theta_0 = -3\,\text{deg}$ and $\dot{x}_r = 0.4m/s$. The step length algorithm parameters ϕ_{thresh} and y_r used in this example are $-1.5\,\text{deg}$ and $0.85m$ respectively. The reference step length is changed every 3 hops, and it is chosen from a set of feasible step lengths. These results do exhibit some error due to our approximations, slightly more than the body-unlocked case in Fig. 5.5, but the foothold errors are still quite reasonable. As we see in Fig. 5.14,

the body angle exhibits non steady-state behavior but is well behaved as any errors do to our approximations are essentially reset at each step when a new set of initial conditions are measured. Unfortunately, since we now must search over an additional variable $\tilde{u}$, the increased number of grid points increases the difficulty of obtaining computation time feasible for complete online deployment. Using more powerful current and next generation parallelized processors will alleviate this problem. In order to implement the centroid-based touch-down angle control method online for our hardware, a table based approach is likely needed for one out of the three search variables.

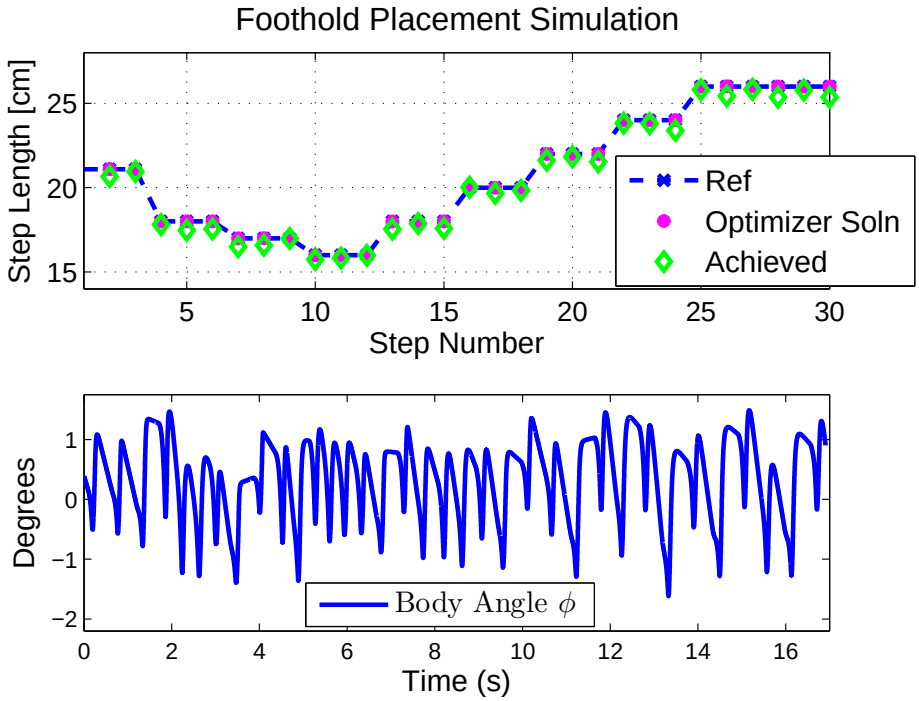


Figure 5.14: Above shows simulation results during step length regulation. The optimizer solution refers to the step length computed with the stance phase approximation, while the achieved state is computed solving numerically the system's equations of motion, implementing the control strategies outlined in this Section.The mean error of the step length results (top figure) is 1.4%, with the maximum error being 3.5%, which is quite reasonable. The resulting trajectory of the body during the simulation is also shown (bottom figure) to illustrate that while it does not exhibit steady-state behaviour it remains both well behaved and small in magnitude, by design.

Chapter 6

Step Length Regulation for More General Hopping Robots

In this chapter we investigate an alternate method for regulating the step length of hopping 2D robots. The method presented here involves a more complex design process, but can be implemented on a wide variety of hopper models. We generate a lower order model and analytically solve for the effective ground reaction force vector to indirectly enforce stable trajectories on the body angular acceleration, while maintaining precise ballistic take-off conditions on the CoM. To illustrate this approach, we provide implementations for both FRANK and the 3-link compliant robot. In this chapter we use the HOPFL construction about the CoM discussed in Chapter 3.4. For the SLIP-based approach, we were able to use *online trajectory generation* by making use of analytical approximations. In contrast, this Section uses *offline trajectory generation* and builds trajectories for direct use by the HOPFL CoM controller from Chapter 3.

6.1 Equivalent CoM Point-mass Model

We construct CoM trajectories for the system by building a lower dimensional model with states that are exactly our control variables x_{cm}, y_{cm} and the uncontrolled body angle ϕ. We generate trajectories for the CoM with the ground reaction force (GRF) pointing towards the CoM, allowing direct control of the position of the robot while keeping the body angle ϕ well behaved. We construct the reduced point mass model by abstracting the system to a single point mass model with GRF magnitude F_t and CoM locations x_{cm} and y_{cm}. The point-mass moves along a 4th order quadratic trajectory that can be either symmetric or asymmetric depending on design parameters. We must augment this point-mass model with an additional parameter, $\delta(t)$, shown in Figure 6.1, which represents an angular offset of the force vector that provides deliberate torques to the bodies when implemented on the robots. All trajectories constructed with this method are a function of stance phase CoM initial conditions x_0, y_0, $\dot{x}_0$, and $\dot{y}_0$.

The dynamics for the reduced system during the stance phase can be written as

$$\begin{aligned} \ddot{x}_{cm} &= \frac{1}{m} F_t \sin(atan(\frac{x_{cm}}{y_{cm}} + \delta(t))) = \frac{1}{m} F_t s_1 \\ \ddot{y}_{cm} &= \frac{1}{m} F_t \cos(atan(\frac{x_{cm}}{y_{cm}}) + \delta(t)) - g = \frac{1}{m} F_t c_1 - g \end{aligned} \tag{6.1}$$

Next we define y_{cm} as a function of x_{cm}. We desire trajectories that are symmetric in y_{cm}, and asymmetric in $\dot{x}_{cm}$ and $\dot{y}_{cm}$, which allows for correction of impact energy losses at take-off and touch-down, and also allows for trajectory switching into different gaits with different desired stride lengths. We also desire the ability to set not only the initial positions and velocities of x_{cm} and y_{cm}, but also the initial acceleration of y_{cm} to align with the expected initial spring force. The curve we choose to enforce is

$$y_{cm}(x_{cm}) = a + b(x_{cm} - p_1)^2 + c(x_{cm} - p_2)^4 \tag{6.2}$$

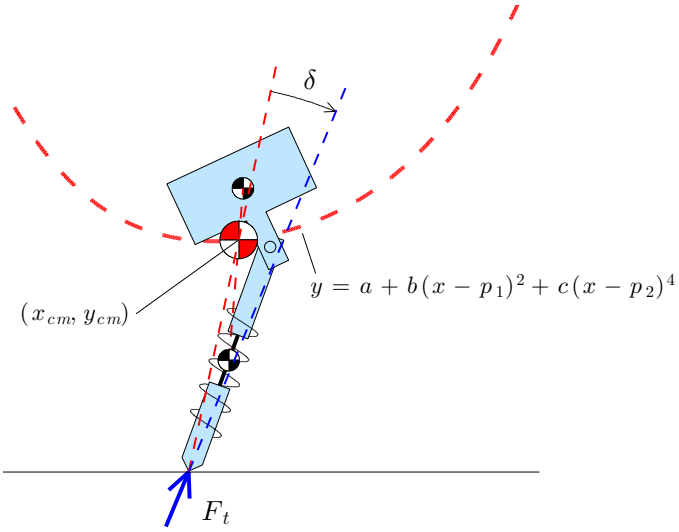


Figure 6.1: Reduced point-mass model overlayed with CoM of FRANK during a stance phase. In our planned trajectories, the ground reaction force can be constrained to point exactly toward a point offset from the center of mass of the system by angle $\delta(t)$. This offset angle can be used to enforce trajectories precisely on the body angle acceleration during the stance phase, while maintaining desired terminating CoM trajectory values.

We next write the acceleration of y_{cm} as:

$$\ddot{y}_{cm} = y''\dot{x}_{cm}^2 + y'\ddot{x}_{cm} \tag{6.3}$$

where the derivatives $y'' = \frac{d^2 y_{cm}}{dx_{cm}^2}$ and $y' = \frac{dy_{cm}}{dx_{cm}}$ can be computed analytically since we have defined $y_{cm}(x_{cm})$ in Eq. (6.2). Next, we combine Eq. (6.1) with Eq. (6.3) to solve for F_t as:

$$F_t = \frac{1}{c_1 - y's_1}(my'' - g), \tag{6.4}$$

and by substituting Eq. (6.4) into Eq. (6.1) we generate stance phase trajectories by selecting trajectory coefficients and solving the differential equation in an ODE solver. The coefficients a, b, and c are determined by initial conditions. Two equations are trivially generated by considering Equation (6.2) evaluated at initial conditions y_0 and x_0, and taking one derivative and evaluating with initial conditions $\dot{y}_0$ and $\dot{x}_0$. In order

to align the initial acceleration of y_{cm} to a desired value we must construct an additional equation by taking two derivatives of Equation (6.2) as

$$\ddot{y} = (2b + 12c(p_2 - x)^2)\dot{x}^2 - \ddot{x}(b(2p_1 - 2x) + 4c(p_2 - x)^3). \tag{6.5}$$

For the above trajectory to be valid it must be equal to the point-mass model's acceleration in Equation (6.1). Equating Equations (6.5) and (6.1) allows us to generate the last needed equation. Solving for the coefficients and substituting in their solution for $\ddot{y}_{cm}$ results in the condition

$$\ddot{x}_{cm} = \frac{\sin(\delta(t) + atan(\frac{x_{cm}}{y_{cm}}))(\ddot{y}_{cm} + g)}{\cos(\delta(t) + atan(\frac{x_{cm}}{y_{cm}}))} \tag{6.6}$$

and since $\ddot{x}_{cm}$ can typically be set instantaneously via our control as we saw in Chapter III, $\ddot{y}_0$ becomes a design parameter. Since our controller needs only CoM stance phase trajectories to operate, we can control the gait step length by choosing trajectories with different terminating velocities. Given two trajectories with different $\dot{x}_{cm}$ terminating values, we construct asymmetric trajectories that switch to or from the desired trajectory set. To simply illustrate the ability to switch between different gaits, we designed three CoM trajectory sets for "Small", "Medium", and "Large" strides, such that the difference in step length between each set is roughly ten percent. Figure 6.2 shows example trajectories for y_{cm} and $\dot{x}_{cm}$ used to achieve gait switching.

The required orientation of the GRF depends on the kinematics of the system being controlled. Thus, different robots will have slightly different stabilization strategies (and possibly auxiliary variables). The power of this control framework, however, is the fact that the trajectory construction procedure in (6.4) once $\delta(t)$ has been defined to stabilize a particular auxiliary variable is *exactly* the same for different robots.

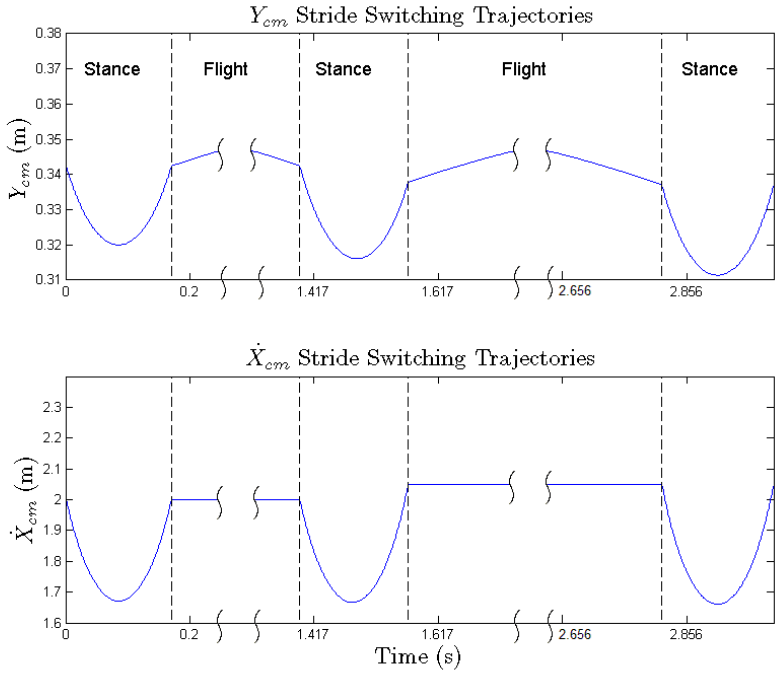


Figure 6.2: We use symmetric CoM trajectories to regulate a specified stride length based on the value of $\dot{x}_{cm}$ at take-off, and use asymmetric trajectories to switch between them. Above shows example trajectories used to switch from a "medium" stride length to a "large" stride length, in which the value of $\dot{x}_{cm}$ at take-off is increased. Also note that the flight phases in this figure are truncated to show the stance phase in greater detail.

6.2 Implementation on SEA 2D Hopper

In this Section we provide implementation details on FRANK. We ensure the initial values of our trajectories for $\ddot{y}_{cm}$ and $\dddot{y}_{cm}$ align with the real system by setting L_a and $\dot{L}_{act}$ to the appropriate touch-down values, which is easily accomplished by controlling them during the previous flight phase. For SEA 2D Hoppers, simply pointing the force vector F_t exactly at the CoM, i.e. $\delta(t) = 0$, will result in a small amount of injected angular momentum at the end of the stance phase. Additionally, during the next flight phase re-positioning the leg will inevitably incur additional angular rotation of the body. We therefore consider using $\delta(t)$ to enforce desired conditions on the acceleration of the

body.

We start by considering the dynamics of the body during the stance phase in a general form as:

$$\ddot{\phi}_{st} = f_{st}(\phi, \dot{\phi}, L, \dot{L}, \theta, \dot{\theta}, L_{act}, u_{hip}) \tag{6.7}$$

We proceed by re-writing f_s in terms of our control variables in the reduced model in (6.1): x_{cm} and y_{cm}. First we make note that we can eliminate $u(t)$ as a variable as

$$u_{hip,stance} = \frac{1}{\beta_x}(-\epsilon_x + \ddot{x}_{cm}) \tag{6.8}$$

where we have used Equations (3.16) and (3.17) and the assumption that the PFL controller is functioning. We use Equation (2.12) and its first derivatives to generate three equations for the state variables L, $\dot{L}$, θ, and $\dot{\theta}$, and generate the last required equation by using the second derivatives of (2.12) together with the system dynamics in (2.20). This is possible as L_a only appears in second derivative terms of the position states. Thus, we re-write the body angle acceleration using a change of variables as

$$\ddot{\phi}_{st} = f_{\tilde{st}}(\phi, \dot{\phi}, x_{cm}, y_{cm}, \dot{x}_{cm}, \dot{y}_{cm}, \delta(t)) \tag{6.9}$$

Here we note that at every time t_i, we can find the required $\delta(t_i)$ to achieve a desired angular acceleration $\ddot{\phi}(t_i)$. For the example presented in this dissertation, the solution was found by using a simple binary search algorithm while solving the differential equation in (6.1). Thus, if the initial body angle ϕ_0 and velocity $\dot{\phi}_0$ are known, such a method can be used to enforce trajectories on $\ddot{\phi}$ that result in desired take-off values ϕ_{TO} and $\dot{\phi}_{TO}$.

Similar to our approach in Chapter 5.5, during the flight phase we use a simple PFL controller to re-position the leg to the next touch-down angle associated with our stance

phase initial conditions. Thus, we can determine the future actuator input as

$$u_{hip,flight} = \frac{1}{\beta_\theta}(-\epsilon_\theta + \ddot{\theta}_{fl}) \tag{6.10}$$

where the PFL coefficients β_θ, ϵ_θ are constructed from the flight phase dynamics and we have used the assumption that the PFL controller is functioning. The flight phase trajectory of the leg angle, $\theta_{fl}(t)$, is a design choice with the only requirement being that terminating value is the next touch-down angle, given by

$$\theta_{TD} = -\cos^{-1}(\frac{y_0(m+m_l) - l_2 m\cos(\phi)}{L_0(m+\frac{m_l}{2})}) \tag{6.11}$$

Using this trajectory, we can then write the flight phase dynamics for ϕ similarly as we did in Chapter 5.5 as

$$\ddot{\phi}_{fl} = f_{fl}(\phi, \dot{\phi}, \theta_{fl}, \dot{\theta}_{fl}, \ddot{\theta}_{fl}). \tag{6.12}$$

For a given designed ϕ_{TO}, $\dot{\phi}_{TO}$, and $\theta_{fl}(t)$, we use Equation (6.12) to calculate the flight phase trajectory of the body and determine the subsequent touch-down values ϕ_{TD} and $\dot{\phi}_{TD}$. The next design decision is the selection of a desired stride length S_{step} and CoM initial conditions x_0, y_0, $\dot{x}_0$, and $\dot{y}_0$. The stride length is given as

$$S_{step} = x_{TO} - x_0 + \frac{2\dot{x}^+_{TO}\dot{y}^+_{TO}}{g}. \tag{6.13}$$

The initial conditions θ_0, $\dot{\theta}_0$, and ϕ_0 are uniquely determined, and the initial body angular rate $\dot{\phi}_0$ is a design parameter. For a stance time t_{st} and sampling rate T_s, the trajectory length N is given by t_{st}/T_s. For a system with no impact dynamics, the terminating trajectory velocity values $\dot{x}(N)$ and $\dot{y}(N)$ could simply be $\dot{x}_0$ and $-\dot{y}_0$, resulting in a symmetric trajectory. However, this is not the case for real systems. The stride length

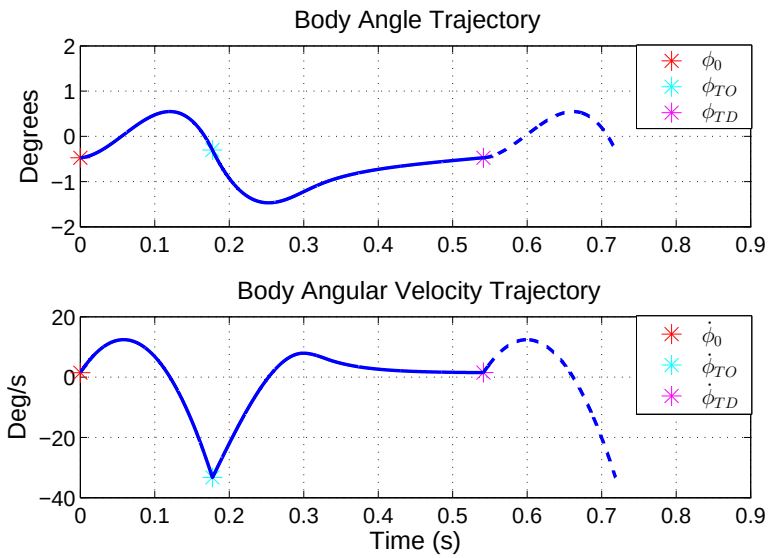


Figure 6.3: The trajectory which the uncontrolled body angle follows is carefully designed such that the touch-down values are equal to the initial conditions from the previous step. The stance phase occurs between the red and cyan markers. The flight phase ends at the magenta marker and returns to the initial value, by design.

is not determined by the terminating velocities of the trajectory $\dot{x}(N)$ and $\dot{y}(N)$, but rather the velocities after the take-off dynamics occur. By using the impact dynamics in Equations (2.16) and (2.15) with Equation (2.12) and its derivatives we can analytically compute the required terminating trajectory values to enforce the condition $\dot{x}_{TD}^{+} = \dot{x}_0$ and $\dot{y}_{TD}^{+} = \dot{y}_0$ as

$$\begin{aligned} \dot{x}_{TO}^{-} &= f_1(x_0, y_0, \dot{x}_0, \dot{y}_0, \phi_0, \dot{\phi}_0, \phi_{TO}, \dot{\phi}_{TO}) \\ \dot{y}_{TO}^{-} &= f_2(x_0, y_0, \dot{x}_0, \dot{y}_0, \phi_0, \dot{\phi}_0, \phi_{TO}, \dot{\phi}_{TO}). \end{aligned} \tag{6.14}$$

We use the asymmetry coefficients p_1 and p_2 to add asymmetry such that the terminating trajectory values are $\dot{x}_{TO}^{-}$ and $\dot{y}_{TO}^{-}$. Note that we can also easily construct trajectories to switch from one set of CoM trajectories to the other by simply finding the proper p_1 and p_2. Therefore, the design task simply becomes selecting $\dot{x}_0$, and $\dot{y}_0$ to achieve a desired

stride length and finding the appropriate p_1, p_2, $\dot{\phi}_0$, ϕ_{TO}, and $\dot{\phi}_{TO}$ such that

$$\begin{aligned} \dot{x}(N) &= \dot{x}_{TO}^{-} \\ \dot{y}(N) &= \dot{y}_{TO}^{-} \\ \dot{\phi}_{TD} &= \phi_0 \\ \dot{\phi}_{TD} &= \dot{\phi}_0 \end{aligned} \tag{6.15}$$

The trajectories are generated offline by solving the differential equation presented in Equations (6.1) and (6.4) for a set of parameters, and enforced in real time using the HOPFL CoM controller in (3.22). An example of such a trajectory for the body angle is shown in Fig. 6.3, where we design the body angle to remain relatively small, quite similar to the SLIP-based method. The trajectory presented here is designed for a stride length of 35 cm. Table 6.1 lists the model and design values used in this particular example. Simulation results over two steps are shown for the CoM control variables in Fig. 6.5, and the uncontrolled body angle in Fig. 6.6. The controller is able to successfully enforce the designed stance phase trajectories, and the body angle's trajectory follows the designed curve, by design. The actuator output for a single step is shown in Figure 6.4, which illustrates that the controller's output is both reasonable and within our hardware limits.

It is worth noting that for the case of FRANK, this method is quite inferior when compared to the SLIP-based technique we saw in Chapter V. The complete trajectory construction procedure is provided here to illustrate how general the method is, and also to note the difficulty of the design process compared to the SLIP-based methods from Chapter V. For robots that are SLIP-like in nature (like FRANK), there is little reason to not use tools readily available in the literature for SLIP. However, the general method presented here is necessary for multi-link hoppers, as we will see next, where the robot is kinematically structured sufficiently different from SLIP.

x_0	-0.051 m	x_{cm} traj initial point
$\dot{x}_0$	2.21 m/s	$\dot{x}_{cm}$ traj initial point
y_0	0.5805 m	y_{cm} traj initial point
$\dot{y}_0$	-1.7 m/s	$\dot{y}_{cm}$ traj initial point
$\ddot{y}_0$	18 m/s^2	$\ddot{y}_{cm}$ traj initial point
ϕ_0	-0.4753deg	Initial body angle
$\dot{\phi}_0$	1.47deg/s	Initial body velocity
ϕ_{TO}, $\dot{\phi}_{TO}$	-0.3deg, -33deg/s	Body design params
t_{st}	0.177 s	Stance trajectory time
$\dot{y}(N)$,$\dot{x}(N)$	0.6891 m/s, 1.8516 m/s	TO velocity design
p_1, p_2	1.4E-3, 2.5E-3	Asymmetry constants
ω_n, ζ, z_3, z_4	18, 0.8, 80, 90	y_{cm} PFL params
K_p, K_d	400, 30	x_{cm} PFL params

Table 6.1: Design Parameters for FRANK

6.3 Implementation on Compliant 3-link Robot

The compliant 3-link robot, seen in Fig. 2.2, is a system for which SLIP-based trajectories are not directly applicable. When we considered the body stabilization control problem in the context of controlling FRANK, a direct extension from SLIP model trajectories was tractable as we only needed to stabilize one joint angle: ϕ. We saw that even for this case the coupling dynamics between one leg angle and one body angle were highly complex. The compliant 3-link robot has too many angles to make this type of control feasible. Therefore, as we saw (2.6), we construct an auxiliary variable resulting from a geometric combination of all the angles in order to analyze body stability in terms of only one term. Contrary to the body angle of FRANK, θ_B cannot be stabilized around zero (vertically straight up), but must be biased at some nominal value. Since the other two control variables are the CoM locations of the robot, we consider implementing our point-mass model based method to control the system. In fact, for this system the stabilization problem is much easier compared to implementation of this method on FRANK.

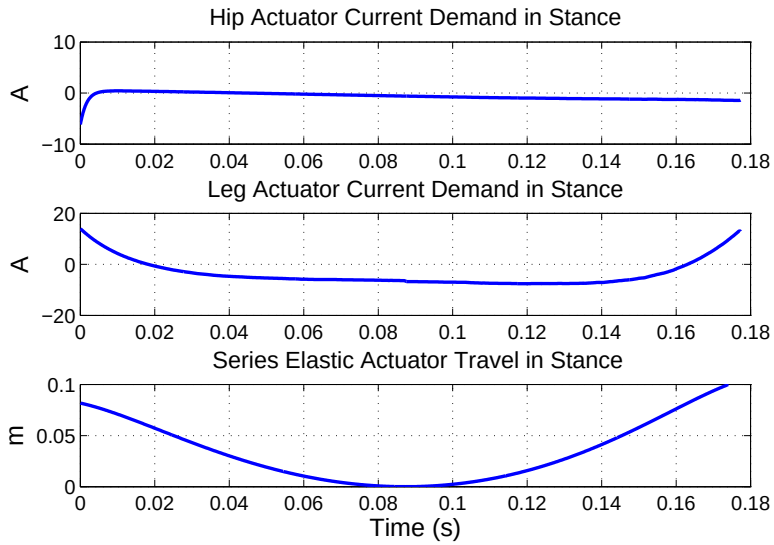


Figure 6.4: Both the hip actuator and the series elastic actuator of FRANK are active to achieve CoM tracking. The controller output stays within our system's maximum output capability of 20A, which equates to torque values of 0.738 Nm on the motor side and 48.7 Nm on the gear side. The position of the series elastic actuator is also shown to illustrate that it stays within 0.1m, the physical limits of our robot.

Trajectories of the CoM and body attitude over time depend upon the orientation of the GRF. In our approach, we correspondingly generate trajectories by planning allowable motion for a point-foot contact and for no rotation of the body; i.e., with the GRF planned to point directly toward the CoM. The trajectories we construct for the compliant 3-link robot are generated such that the total net torque w.r.t. the CoM is zero, thus under ideal conditions with no perturbations there would be no change in the angular momentum of the system, and the body angle would remain constant during both stance and the subsequent flight phase. Such trajectories can be constructed by using our reduced point-mass model construction and simply setting $\delta(t)$ in (6.1) to zero for all time, as shown in Fig. 6.7. This simplifies the design process considerably, as we no longer have to perform a binary search and calculate the body angle dynamics at each step. Given stance phase initial conditions, we generate trajectories by solving (6.1) as discussed in 6.1.

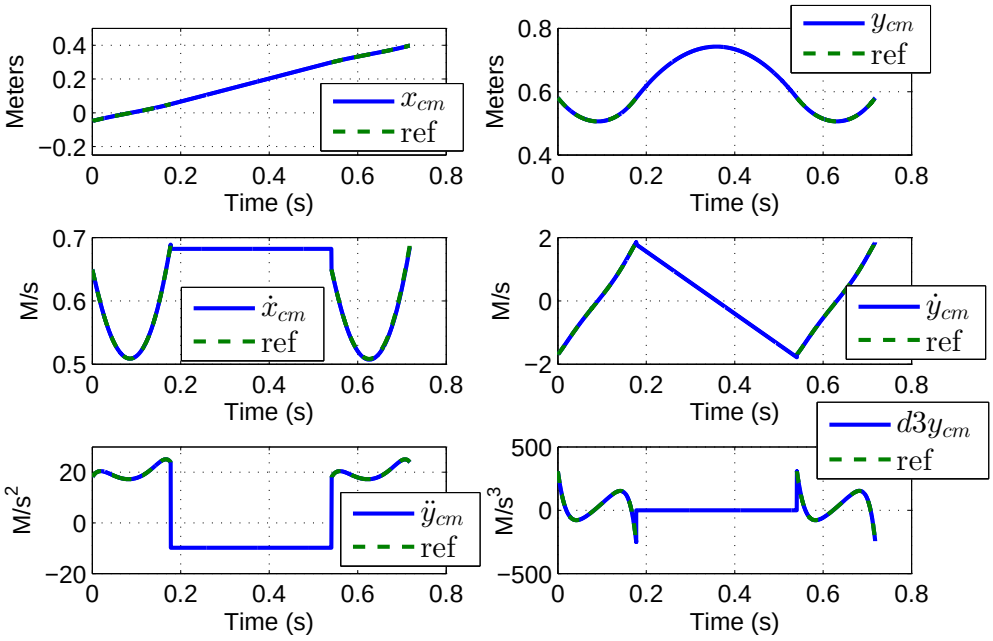


Figure 6.5: Simulation results on FRANK for over 2 steps enforcing CoM trajectories, with references shown during the stance phase in green. The initial conditions at the second stance phase are aligned with the trajectory, by design.

6.3.1 Operation on Rough Terrain

Thus far we have assumed that our system is able to stay on our designed CoM trajectories perfectly for all time. A real implementation of this control framework, however, must remain robust to non-zero perturbations of the states, which in turn will inject angular velocity into the system that must be removed. Additional disturbances to the system due to ground height offsets (i.e., rough terrain) may be also treated as misalignments of the trajectory initial conditions at impact. Since this system does not employ SEA, for a given stance-phase trajectory it is possible the spring will not be finished decompressing when the trajectory terminates. One solution to this problem is to extend the stance phase trajectory by forward solving the ballistic equation for the CoM. In other words, $\dot{x}_{cm}$ remains constant (at the planned take-off velocity) and

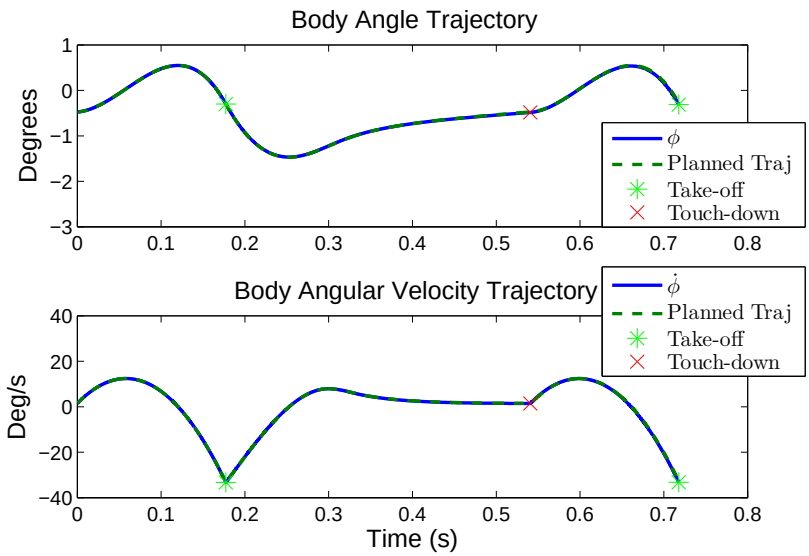


Figure 6.6: Simulation results on FRANK for over 2 steps for the body angle, with the predicted trajectory shown in green. As expected, the body angle is well behaved during the stance and flight phases, and is correctly aligned with the subsequent stance phase initial condition at the end of the flight phase.

$y_{cm}(x_{cm})$ keeps the total vertical energy ($mgy + \frac{1}{2}m\dot{y}^2$) also constant. This will result in injection of some amount of undesired angular velocity.

To address these problems, we divide the stance phase into two parts, *Correction* and *Main*, as shown in Fig. 6.8. The Main phase implements the HOPFL CoM controller in (3.22), while the Correction phase instead controls y_{cm} and θ_B using the same HOPFL construction, simply selecting two different auxiliary variables. This is done to remove any angular velocity introduced into the system, and return the body angle to a nominal value.

At the instant of ground impact, the body angle of the system may be above or below the desired system value. The Correction Phase trajectories for the body angle are planned on the fly to match the initial body angle, velocity, and acceleration conditions and then to converge to zero angular momentum by the end of the correction phase. We construct smooth trajectories from a combination of polynomial and exponential

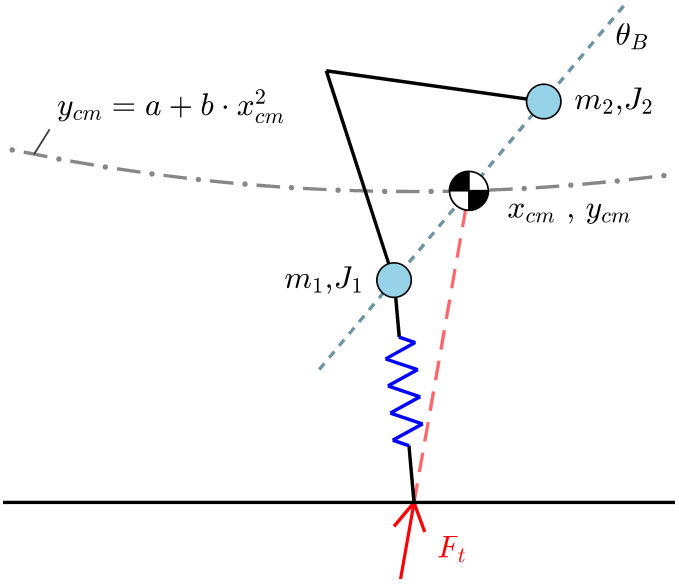


Figure 6.7: Reduced point-mass model overlayed on the 3-link compliant robot's CoM during a stance phase. In our planned trajectories, the ground reaction force is intentionally constrained to point exactly toward the center of mass of the system. If the system begins with zero initial angular velocity, as desired, then this GRF constraint automatically maintains zero angular velocity throughout stance, by design.

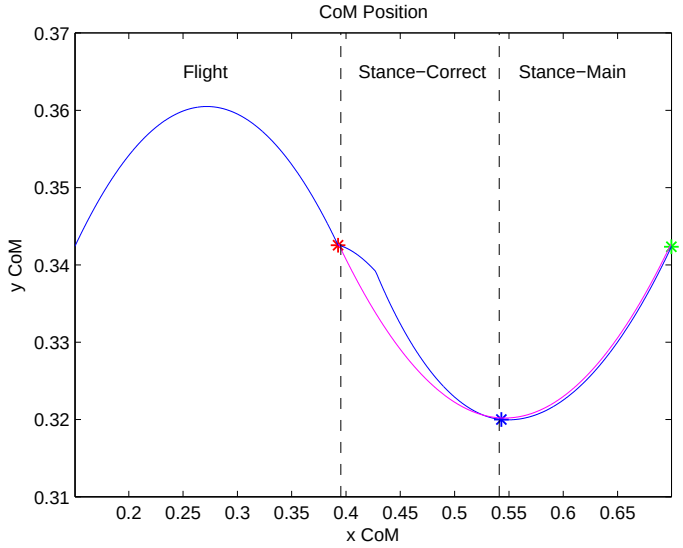


Figure 6.8: To control x_{cm}, y_{cm} and θ_B, the stance phase is divided into two phases. The Correction phase removes any present angular velocity and brings the system to the desired CoM (x, y) trajectory in space.

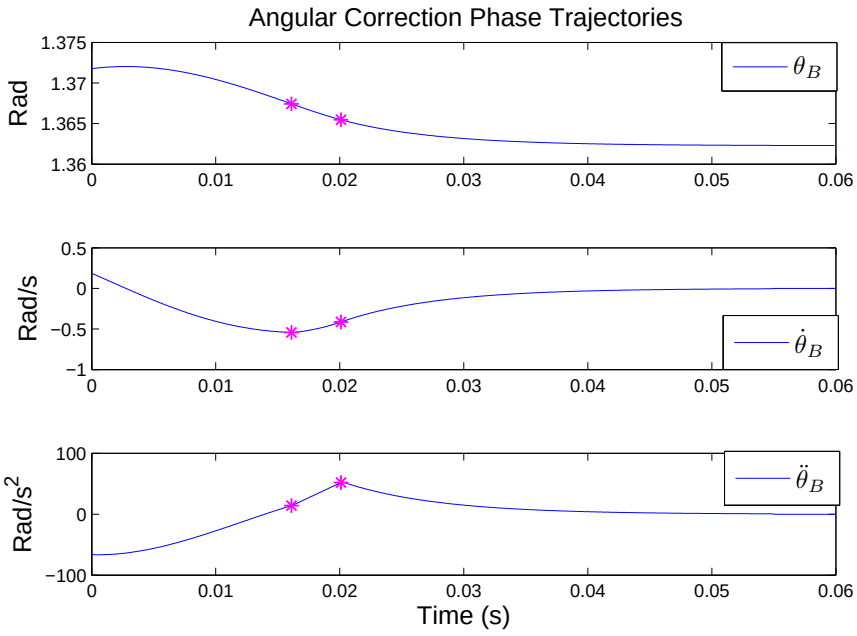


Figure 6.9: Example smooth trajectory generated for the body angle in the Correction Phase. The trajectory is generated by combining two polynomials and a decaying exponential function, and solving for the required coefficients such that the touchdown initial conditions are aligned with the trajectory initial values. The trajectories return the body angle to its nominal value and remove all angular velocity.

functions. Figure 6.9 illustrates a trajectory generated for negative body angle drift with positive angular velocity and negative body angular acceleration at touch-down. After the body angle trajectory is chosen, we construct a valid trajectory for y_{cm}. The trajectories for y_{cm} are generated by casting the y_{cm} trajectory as a function of our uncontrolled variable, x_{cm}, as we previously defined in (6.2). We therefore set the y_{cm} reference trajectories in the Correction phase as a function of the current x_{cm}, i.e.,

$$\begin{aligned} y_{ref} &= y_{traj}(x_{cm}) \\ \dot{y}_{ref} &= \dot{y}_{traj}(x_{cm}) \\ \ddot{y}_{ref} &= \ddot{y}_{traj}(x_{cm}) \end{aligned} \tag{6.16}$$

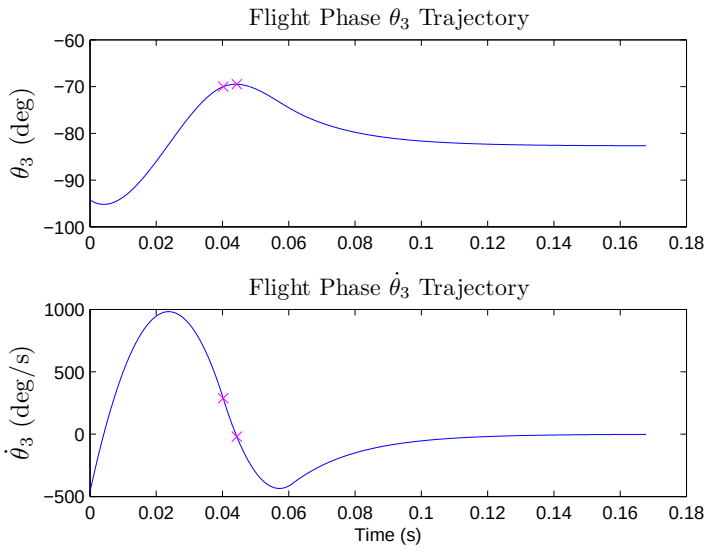


Figure 6.10: Example smooth trajectory generated for spine angle, θ_3, during the flight phase. The trajectory is generated in an identical way as the body angle trajectories previously seen in Fig. 6.9, by using a combination of polynomials and decaying exponential functions to ensure initial condition alignment and to provide convergence with zero velocity to initial spine angle.

During the flight phase, the mass stance link can be repositioned arbitrarily without affecting the rest of the dynamics, so we only have one available actuator τ_2, which we use to implement simple collocated PFL on the spine angle θ_3, in order to return the angle to the initial stance value. The trajectory is generated in a similar manner as the body angle trajectories, and an example is shown in Fig. 6.10.

The parameters used in this simulation study are shown in Table 6.2. Any unlisted coefficients are zero. Simulation results on minor rough terrain are shown in Fig. 6.11. We see that results are reasonably good, where the robot has the ability to switch into different gaits dynamically and regulate different stride lengths. Step length accuracy is well below 10% on average, and is not as precise as the SLIP-based methods we saw earlier on FRANK in Chapter 5. This is more or less due to limitations of the compliant 3-link robot as a hopper, as it does not employ SEA and thus aligning the initial acceleration of the system at touch-down to reference values is more difficult to implement. The design

m_1, m_2	2.0 kg	Hip masses
J_1, J_2	0.0052 $kg\ m^2$	Joint inertias
L_0	0.2540 m	Link natural lengths
k_{leg}	2,625 N/m	Spring constant
b_k	25.36 Ns/m	Spring damping
$\theta(0)_B$	1.37 rad	Initial body angle
$x_{small}(0)$, $\dot{x}_{small}(0)$	-0.103 m, 1.82 m/s	Small stride IC
$x_{med}(0)$, $\dot{x}_{med}(0)$	-0.141 m, 2.0 m/s	Med stride IC
$x_{large}(0)$, $\dot{x}_{large}(0)$	-0.175 m, 2.2 m/s	Large stride IC
a_{small}, b_{small}	0.341, 1.49	Small stride coeffs
a_{med}, b_{med}	0.327, 1.35	Med stride coeffs
a_{large}, b_{large}	0.295, 1	Large stride coeffs
ω_n, ζ, p_3	25, 0.9, 700	PFL gain parameters

Table 6.2: Simulation Parameters for Compliant 3-link Hopper

process is also significantly more complex when analytical calculations are not available. However, this method can be used to generate stable trajectories that can be stored in a database capable of regulating step lengths with reasonable accuracy for a wide variety of compliant hopping robots.

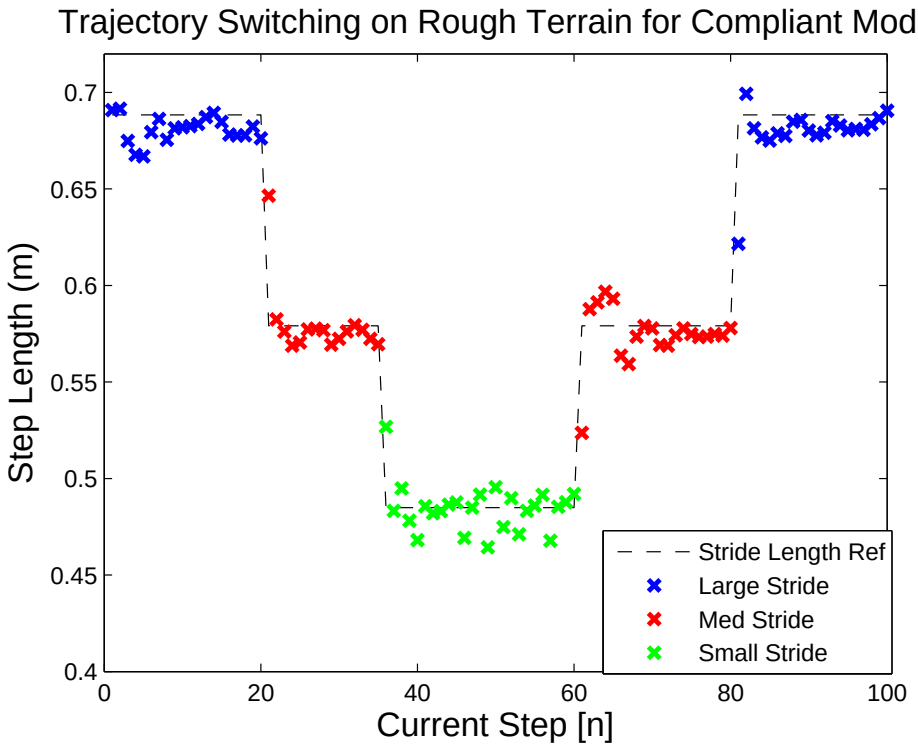


Figure 6.11: Step length results implementing HOPFL CoM control on the compliant 3-link robot. The ground level was also randomly varied between -0.5cm and 0.5cm at each step to simulate minor terrain variations. We see comparable results to the SLIP-based method in 5.14, however each trajectory needing to be designed manually makes implementation more difficult overall. This method does however have the advantage of being general enough to be applied to a variety of hopper models.

Chapter 7

Conclusions and Future Work

Hopping robots are highly dynamic systems that are useful to study if we wish to advance the state-of-the-art of precision foothold placement for compliant legged systems. We have investigated in detail control strategies towards implementation of precise controllers to regulate both apex states and step lengths during dynamic hopping gaits. We have presented motivating hardware results and algorithms for modeling and control techniques for a realistic series-elastic actuated hopping robot to achieve accurate state tracking. For practical hopping robots, actuators have real dynamics that must be modeled for state tracking to work well with feed-forward control methods. For closed-form approximations of step-to-step dynamics, we argue such models are essential for both higher-level planning and low-level feed-forward and feedback control. We have developed high order partial feedback linearization based control strategies specifically for the leg state of SEA hopping robots, and verified that such strategy is not only practical to implement on hardware but also yields quite accurate results on a real hardware implementation.

By using our high order partial feedback linearizing controller directly on the leg state of the robot, we are able to generate methods that use fast SLIP-based approximations,

allowing us to parametrize and calculate the reachable space in real time. By constructing a PFL controller for the hip torque input, we are able to provide stabilizing body motions and also retain the ability to analytically compute our trajectories from SLIP. The ability to perform analytical reachability computations at each step is powerful. This allows us to construct control frameworks online on a step-to-step basis that not only result in excellent performance but also allow for a large set of possible footholds at each step. We argue this is critical in improving both reliability (e.g., ability to recover from terrain perturbations) and agility (e.g., ability to accurately go to any of a family of reachable future states) of realistic spring-legged robots.

For robots that cannot be directly represented as an evolution of the SLIP model, we can construct trajectories for the systems CoM by using a very general point-mass model with an acting GRF vector. By constructing an equivalent point-mass model we provide a method that allows for direct control of the CoM such that accurate stride regulation and switching is possible, while also maintaining body stability. We provide a method that allows for the construction of trajectories with arbitrary initial conditions while maintaining desired take-off velocities. Future work includes constructing a database of trajectories with varying initial conditions to account for problems such as uneven terrain and long term angular drift. Operation on rough terrain results in unknown disturbances to the initial position and velocities of the robot, thus having a method for generating alternate trajectories for varying initial conditions is an extension planned for this work. Future work could consider sensing improvements or moving the hardware to a 3D platform, and providing hardware implementations of the 2D foothold selection strategies presented in this dissertation.

Future work also may consider further implementations on the robot FRANK hardware. As discussed in Appendix D, FRANK is currently incapable of precision touchdown angle control and/or traversing terrain boards due to state estimation hardware

challenges regarding the body angle and forward velocity. In order to achieve higher touch-down angle precision control on FRANK, several paths for future work are possible. First, a mechanically stiffer boom may alleviate some of the torsional forces, and the boom also likely must be more securely attached to the ceiling assembly. Completely decoupling the robot from the boom is also an option, however this requires development of algorithms for 3D hopping. Second, state estimation can likely be improved with higher quality IMU systems. Implementing IMU-based measurements on hoppers is quite a challenging problem, as any accelerometer measurement must withstand impactive forces and remain well behaved. Incorporating non-linear state estimation techniques is also a possibility for future work. Lastly, augmenting the hardware to allow for higher energy gaits is a viable option. While this certainly will not solve any of these problems completely, it will result in longer ballistic phases, and can likely be accomplished by making stiffer system springs and higher actuation power available to FRANK.

Appendix A

System Identification of FRANK

This Section briefly overviews system identification techniques used to determine the model parameters of the robot FRANK. These parameters were identified by decoupling the dynamics of state variables via temporary modification of the robot in each case. Even though we use frequency identification in some cases, it is important to note that our desired end result is not a transfer function, but rather individual parameter values to be used in the complete non-linear model. Only experimental methods and simulation results are presented here, the numerical parameters extracted from the data can be seen in Table 2.1.

A.1 Leg Spring

The simplest parameters to identify are those of the system spring. For these experiments, the SEA is set to zero for all time, decoupling it from the dynamics. Additionally, the body is mechanically locked and the robot's leg is actively locked, resulting in the

passive 1D dynamics

$$\ddot{L} = \frac{1}{m_B}(K(-L + L_0 + c) - b_2\dot{L} - m_B g) - f_2 \text{sgn}(\dot{L}) \tag{A.1}$$

Both m_B and L_0 can be measured directly from the robot. The unknown parameters are therefore: K, c, b_2, and f_2. The experiments performed are passive drop tests, where the robot is initialized at some vertical height and released. The resulting stance phase trajectory, consisting of simply the spring dynamics compressing and expanding, are then measured. The resulting curve is approximately a classical 2nd order response of a spring-mass system, with the pre-load c and Coulombic term f_2 causing slight magnitude and phase differences.

Approximately 100 trials of drop test data were recorded, and stored in a database along with corresponding initial conditions. A numerical simulator implementing the dynamics in (A.1) then iterates through each set of initial conditions inside the non-linear minimization function **nlfit**, implemented in MATLAB. This method does take some time to iterate, however fairly accurate results can be obtained, as we see in Fig. A.1.

A.2 Series-elastic Actuator

In order to identify the model parameters of the SEA, the robot is rested on a pedestal, and the SEA is actuated. This decouples the leg state dynamics from the system, resulting in the dynamics

$$\ddot{L}_a = \frac{1}{m_e}(K(-L_a + c) - b_1\dot{L}_a + k_p(2L_a - c_p) - \nu u_{leg}) - f_1 \text{sgn}(\dot{L}_a) \tag{A.2}$$

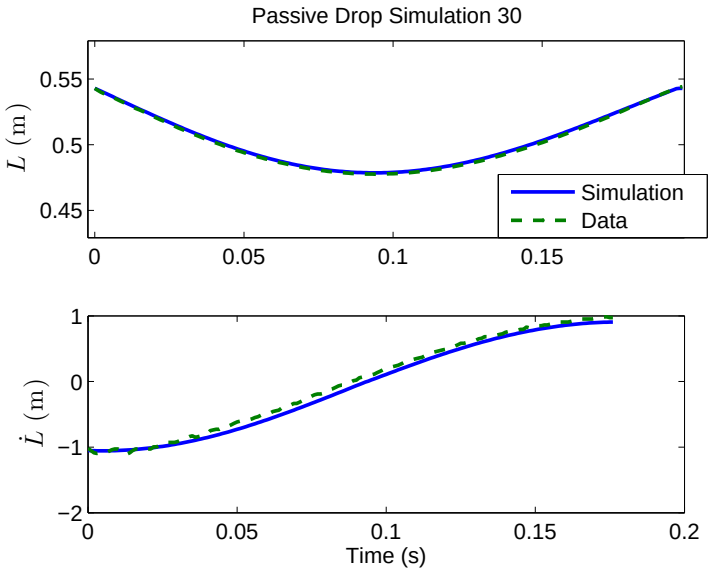


Figure A.1: Simulated accuracy of the identified model parameters for the leg state during a passive drop test. Although some noise is present on the velocity of the leg state, due to the filter, the model accuracy is reasonable.

The spring constant K is known from the previous experiment, and we also know $\nu = \frac{0.5Nk_t}{r}$, where the motor torque constant k_t, gear ratio N, and pulley radius r are all known. Similarly, the tension spring compression c_p can be measured with callipers. Therefore, the unknown parameters in this case are: m_e, b_1, and f_1. Similar to the decoupled leg dynamics, this system's dynamics are quite similar to an active 2nd order spring-damper system, therefore we look to the method of swept sine identification. In order to avoid impacting boundary conditions, it is necessary to vary the amplitudes when taking experimental data, therefore we expect some discontinuities in the frequency response due to the Coulombic term.

Approximately 500 frequency response points were experimentally measured, and stored in a database along with input magnitudes and initial conditions. A numerical simulator implementing the dynamics in (A.2) again iterates through each set of initial conditions inside a non-linear minimization function to determine the model parameters. The resulting simulated frequency response, along with collected experimental data, is

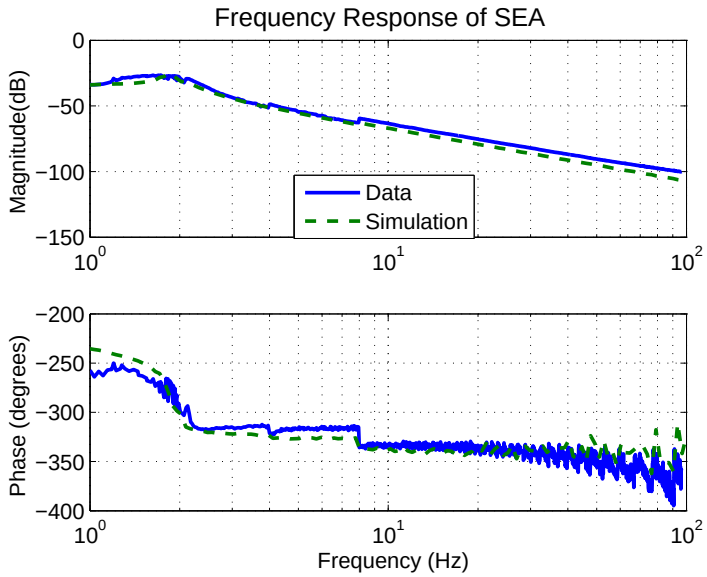


Figure A.2: The frequency response of the SEA for both measured data and simulated non-linear model is shown above. Since this data is constructed from several sets of experiments, some discontinuities are present due to varying input amplitudes, since Coulombic friction is present.

shown in Fig. A.2.

A.3 Leg Angle Actuator

Similar to the SEA identification experiment, the model parameters for the leg angle actuator are also determined via swept sine. Therefore, we again rest the robot on a pedestal and zero out all other active terms. It is important to note that for this experiment the robot is weighted down to prevent the body of the robot from flexing due to torsional forces during the data collection. Additional system identification results attempting to identify these torsional dynamics can be seen in Appendix D. In this case, the dynamics are simply a pendulum with mass and inertia as

$$\ddot{\theta} = \frac{-(Nk_t u_{hip} + g\frac{L_0}{2} m_l sin(\theta))}{(m_l(\frac{L_0}{2})^2 + J_l)} - \tilde{b}_\theta \dot{\theta} \tag{A.3}$$

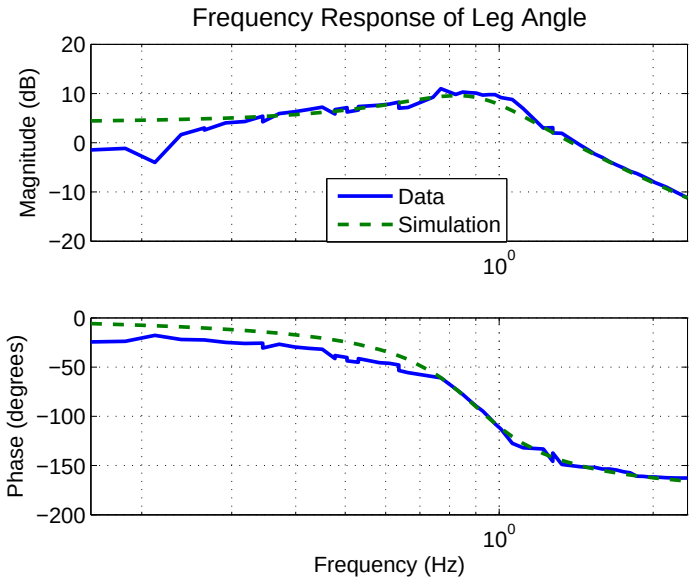


Figure A.3: The frequency response of the leg angle fits fairly well to a simple 2nd order linear transfer function, as shown. The low frequency data has some mis-alignments, likely due to some minor frictional effects in the motor assembly.

The unknown model parameters here are only m_l and J_l, and if we choose to include damping $\tilde{b}_\theta$. For this system we can simply apply a small angle approximation and implement swept sine identification directly, with the model parameters determined from the natural frequency and a few points from the frequency response. Figure A.3 illustrates that this model matches reasonably well.

A.4 Body Inertia

The experiments performed to provide data for the purpose of determining body inertial parameters are as follows. First, the leg of the robot is mechanically locked, and the spring is disabled. The robot's leg is fixed on the ground such that it cannot translate or rotate. Next, the body is mechanically unlocked and allowed to fall from it's

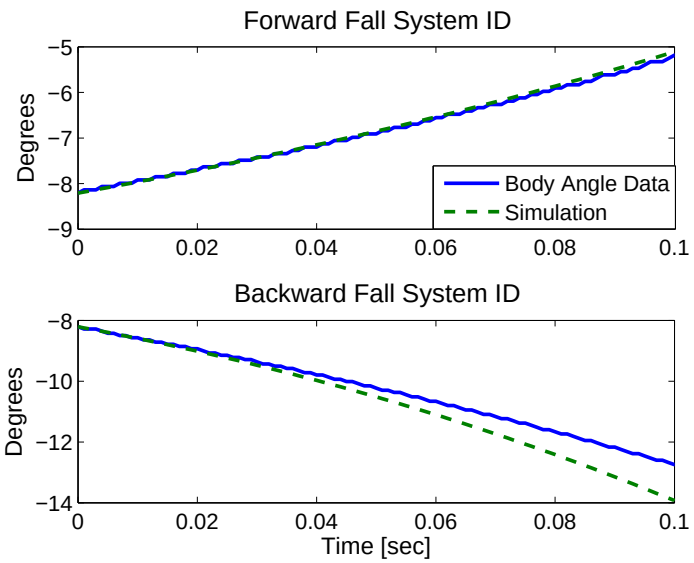


Figure A.4: The leg assembly of the robot was mechanically locked and fastened in place, allowing free-fall experiments on the body to be performed. Above shows simulation results for both forward and reverse fall experiments of the body of the robot. The non-zero initial conditions are due to the asymmetry of the body, which was later removed by adding weights to the edge of the robot's frame.

equilibrium position. The dynamics of this modified system are

$$\ddot{\phi} = \frac{(m_B - m_l)gl_1}{(m_B - m_l)l_1^2 + J} sin(\phi + \lambda) \tag{A.4}$$

where λ represents asymmetry in the body assembly of the robot. The unknown parameters are therefore λ, J, and l_1. Several free-fall experiments were performed, and the parameters were determined by again using non-linear curve fitting minimization functions. The resulting simulated fall responses for both forward and reverse directions can be seen in Fig. A.4. In order to remove the asymmetry in the body, weights were later paced on the edge of the robot's frame. This approximately results in $\lambda = 0$ and the robot's body being symmetric in the *horizontal* direction. The body CoM is still not collocated with the hip joint, as l_1 is non-zero.

A.5 Boom Angle Dynamics in Flight

This Section provides the model for the dynamics of the boom angle ψ, along with analytical SAA approximations that can be used. Note that in simulation, we still typically assume purely ballistic dynamics. During hardware operation however, the boom angle ψ follows non-linear dynamics that must be accounted for when determining proper take-off velocities. The hip position of the robot, effectively the body position for small ϕ, can be calculated referenced from the foot or the boom as

$$\begin{aligned} y_h &= y_B + L_B \sin(\psi) \\ y_h &= y_{foot} + L\cos(\theta - \phi)\cos(\theta_\perp) \end{aligned} \tag{A.5}$$

where L_B and y_B represent the boom length and boom joint height respectively. This allows us to convert information from the leg states L and $\dot{L}$ to boom states ψ and $\dot{\psi}$. Similarly, we can convert the leg take-off velocity $\dot{L}_{TO}^-$ by evaluating the impact dynamics in Eq. (2.16), and convert the resulting post-impact take-off velocity $\dot{r}^+$ to initial boom angle take-off velocity, $\dot{\psi}$, using

$$\begin{aligned} \dot{\psi} &= \frac{1}{L_B\cos(\psi)}(-L_0\cos(\theta-\phi)\sin(\theta_\perp)\dot{\theta}_\perp + Q) \\ Q &= \cos(\theta_\perp)(\dot{r}^+\cos(\theta-\phi) - L_0\sin(\theta-\phi)(\dot{\theta}-\dot{\phi})) \end{aligned} \tag{A.6}$$

where the planar leg angle θ, perpendicular angle $\theta_\perp$, body angle ϕ, velocities, and boom angle ψ are measured at take-off. It can be shown using a Lagrangian approach that the dynamics of the flight phase can be written in the form $\ddot{\psi} = F_{cbl} + A_1(B_1 cos(\psi) + C_1 sin(\psi))$, where F_{cbl} represents frictional effects due to boom cables, and A_1,B_1,C_1 are functions of the boom parameters. Since during operation the boom angle remains relatively small (certainly < 15 degrees), we can use a small angle approximation, resulting

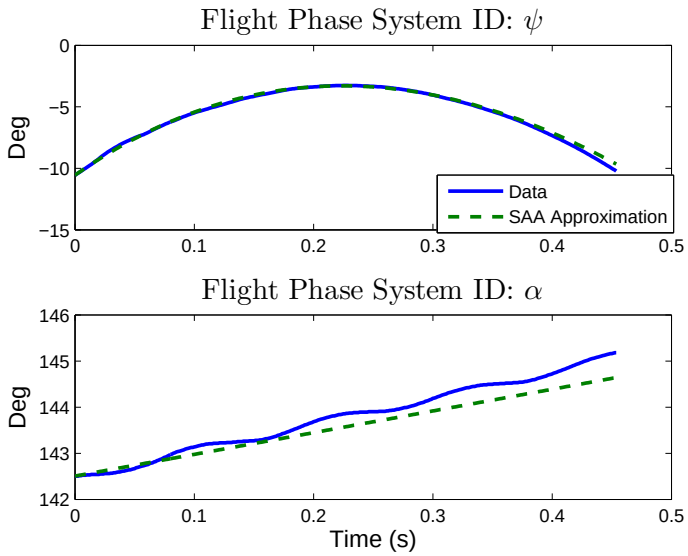


Figure A.5: The flight phase during operation on the boom hardware can be approximated using analytical SAA calculations. However, unmodeled torsional dynamics in the system result in lower accuracy for the translational boom angle α. While a higher order model can be adopted to account for these dynamics, the hardware currently lacks sensing and state estimation to measure them in real-time.

in the dynamics in the form $\ddot{\psi} \approx F_{cbl} + A_1 B_1 + A_1 C_1 \psi$.

Similarly, the translational component of the boom, α, can be measured to approximate horizontal displacement of the robot as $x_h = L_B \alpha$. The dynamics of the horizontal component are assumed to be purely ballistic. Simulation results for the obtained model parameters can be seen in Fig. A.5. Note that during the stance phase the dynamics of the boom are largely negligible due to the high spring forces, however, the subsequent ballistic phases must precisely account for boom losses if precision apex control is to be achieved as in Chapter 4.

Appendix B

Calculating and Evaluating HOPFL Coefficients

This Section discusses implementation details of the control law coefficients necessary to calculate our HOPFL controllers. If one were to symbolically attempt to represent the jounce of the robot in terms of only the state X, which is possible using the control methods presented here, it is likely the coefficients will consist of *millions* of characters, making implementation quite difficult. The HOPFL coefficients (both the CoM and leg construction) for the robot FRANK are an example of this. For example, at each time step, in order to implement the control law we saw in (3.9), the controller must evaluate a function call returning the HOPFL coefficients as:

$$[\tilde{\gamma}_L, \epsilon_L, \beta_L, \eta_L, \alpha_L] = f_{HOPFL}(t, X, u_{hip}) \tag{B.1}$$

where f_{HOPFL} represents the function call the controller makes on board the computer system. Rather than attempting to calculate everything at once, the implementation carried out to produce the results in this dissertation is as follows:

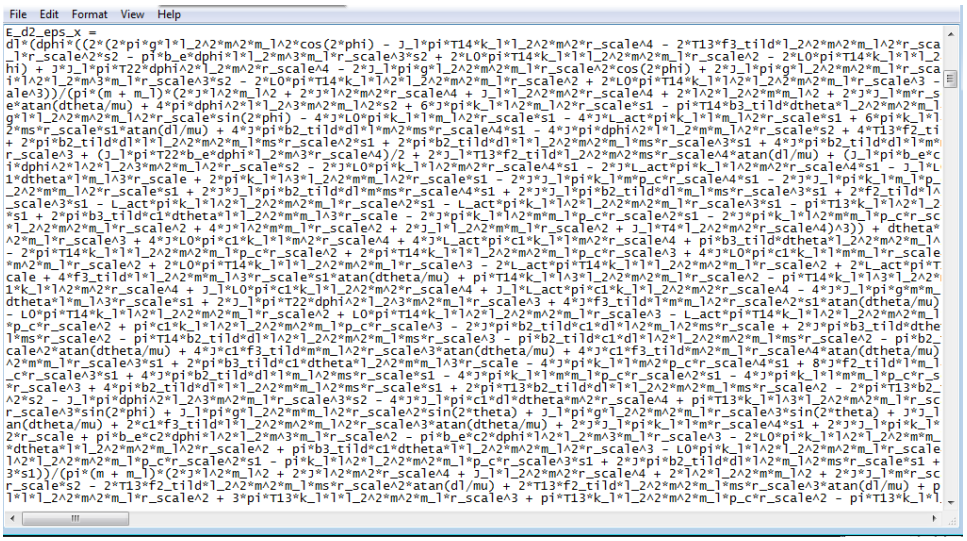


Figure B.1: Attempting to evaluate a HOPFL coefficient directly and only as a function as X results in very large equation, Here we see one coefficient consisting of approximately 3 million characters! It is typically much better to calculate each coefficient in stages.

i. Measure the current state X and calculate any control law for u_{hip}.

ii. Calculate common trigonometric terms. These are typically of the form $cos(i\theta - j\phi)$ and $sin(i\theta - j\phi)$ for i and j ranging from 1 to 8. These terms are locally stored as new variables, and all other equations can be written as a function of them.

iii. Calculate state accelerations $\ddot{L}$, $\ddot{\theta}$, $\ddot{\phi}$, $\ddot{L}_a$, using (2.20).

iv. Calculate the first derivative of any u_{hip} controller as a function of the state X and acceleration variables.

v. Calculate state jerk terms $\dddot{L}$, $\dddot{\theta}$, $\dddot{\phi}$, $\dddot{L}_a$ as a function of the state acceleration terms and hip control law derivative terms previously calculated.

vi. Calculate the second derivative of any u_{hip} controller as a function of the state X and it's first two derivatives.

vii. Calculate HOPFL leg coefficients in (3.9) as a function of the already defined state X, it's fist two derivatives, and the hip control law input and it's first two derivatives.

Using this method and defining the coefficients numerically in stages results in a significantly less complicated equation for each coefficient. In particular to the coefficient we saw in B.1, the resulting coefficient consists of roughly 250 thousand characters as opposed to over 3 million when attempting to evaluate everything at once. The results presented in this dissertation use this method, and the function call f_{HOPFL} was measured to take approximately 1ms to evaluate on a computer with an i7 2600k processor. This is reasonable and well suited for real-time deployment on a modern computational hardware setup.

Appendix C

1D SEA Hopper Analytical Solutions

The stance phase trajectories for 1D hoppers can be computed analytically in some cases. Note that this is *almost never* the case for 2D hopping systems, as the dynamics are often analytically unsolvable. We consider generating an analytical solution for a SEA 1D hopper with dynamics:

$$\ddot{x}_1 = \frac{1}{m_{eff}}(k(x_2 - x_1 - L_0) - b_1\dot{x}_1 - \gamma_2 u) - f_1 sign(\dot{x}_1) \tag{C.1}$$

$$\ddot{x}_2 = \frac{1}{m_B}(k(x_1 - x_2 + L_0 + c) - b_2\dot{x}_2 - m_B g) - f_2 sign(\dot{x}_2) \tag{C.2}$$

where the model parameters parallel those of the Hopper B and Hopper C from Chapter II. In this case, x_1 represents the position of the SEA, and x_2 represents the vertical distance from the robot hip to the ground, essentially the leg length during the stance phase. At touchdown a trajectory can be generated based on initial conditions that characterizes the states of the system for all future time. The solution generated follows

from two assumptions. First, the input current is a step change. A new solution can be generated from any point given new initial conditions and current step magnitude. Second, $\dot{x}_2$ remains negative during compression and positive during expansion. This allows the solution to be broken up into two pieces in order to overcome the non-linearity of the static friction term. If $\dot{x}_1$ does not remain positive sign during the stance phase, the equations must be broken up into N additional pieces, where N is the number of times $\dot{x}_1$ crosses the zero-axis.

The solution is generated using a Laplace Transform technique. Taking the Laplace Transform of (C.1) and (C.2) yields

$$X_1(s) = \frac{s^2 x_1(0) + s(\dot{x_1}(0) + \frac{x_1(0)b_1}{m_{eff}}) + \lambda_2 + sX_2(s)\frac{k}{m_{eff}}}{s(s^2 + s\frac{b_1}{m_{eff}} + \frac{k}{m_{eff}})} \tag{C.3}$$

$$X_2(s) = \frac{s^2 x_2(0) + s\beta - \gamma + s\frac{k}{m_2}X_1(s)}{s(s^2 + s\frac{b_2}{m_B} + \frac{k}{m_B})} \tag{C.4}$$

where

$$\gamma = -\frac{f_2}{m_B} + g - \frac{k(L_0 + c)}{m_B} \tag{C.5}$$

$$\beta = \dot{x_2}(0) + x_2(0)\frac{b_2}{m_B} \tag{C.6}$$

$$\lambda_2 = -\frac{\gamma_2 u}{m_{eff}} - \frac{kL_0}{m_{eff}} - f_1 \tag{C.7}$$

Substituting (C.3) into (C.4), it can be shown after some algebra that

$$X_2(s) = \frac{x_2(0)s^4 + H_4 s^3 + H_3 s^2 + H_2 s + H_1}{s^2(s+a)(s+b)(s+c)} \tag{C.8}$$

where a, b, and c are roots to the polynomial

$$s^3 + s^2(\frac{b_2}{m_B} + \frac{b_1}{m_{eff}}) + s(\frac{k}{m_B} + \frac{k}{m_{eff}} + \frac{b_1 b_2}{m_{eff} m_B}) + \frac{k(b_1 + b_2)}{m_{eff} ms}$$

and

$$H_4 = x_2(0)\frac{b_1}{m_{eff}} + \beta \tag{C.9}$$

$$H_3 = x_2(0)\frac{k}{m_{eff}} + \frac{b_1 \beta}{m_{eff}} - \gamma + \frac{k x_1(0)}{m_B} \tag{C.10}$$

$$H_2 = (\dot{x_1}(0) + \frac{x_1(0) b_1}{b_{eff}})\frac{k}{m_B} + \frac{k\beta}{m_{eff}} + \frac{b_1 \gamma}{m_1} \tag{C.11}$$

$$H_1 = \frac{-\gamma k}{m_{eff}} - \frac{u k \gamma_2}{m_B m_{eff}} - (\frac{L_0 k}{m_{eff}} + f_1)\frac{k}{m_B} \tag{C.12}$$

Using (C.8), the Laplace transform for $X_1(s)$ can be re-written as

$$X_1(s) = \frac{s^2 x_1(0) + s(\dot{x_1}(0) + \frac{x_1(0) b_1}{m_{eff}}) + \lambda_2}{s(s+d)(s+e)} + Z(s) \tag{C.13}$$

where d and e are roots to the denominator of (C.3), and

$$Z(s) = \frac{\frac{k}{m_{eff}}(x_2(0)s^4 + H_4 s^3 + H_3 s^2 + H_2 s + H_1)}{s^2(s+a)(s+b)(s+c)(s+d)(s+e)} \tag{C.14}$$

Finally, taking the Inverse Laplace Transform of (C.8) and (C.13) with (C.14), solutions for all states can be easily found as

$$x_2(t) = A + Bt + Ce^{-at} + De^{-bt} + Ee^{-ct} \tag{C.15}$$

$$\dot{x_2}(t) = B - aCe^{-at} - bDe^{-bt} - cEe^{-ct} \tag{C.16}$$

$$x_1(t) = \hat{A} + \hat{B}t + \hat{C}e^{-at} + \hat{D}e^{-bt} + \hat{E}e^{-ct} + \hat{F}e^{-dt} + \hat{G}e^{-et} \tag{C.17}$$

$$\dot{x_1}(t) = \hat{B} - a\hat{C}e^{-at} - b\hat{D}e^{-bt} - c\hat{E}e^{-ct} - d\hat{F}e^{-dt} - e\hat{G}e^{-et} \tag{C.18}$$

where A- E and $\hat{A}$- $\hat{G}$ are coefficients determined from partial fraction decomposition. The analytical solutions for $x_1(t)$ and $x_2(t)$ are thus constructed.

Appendix D

Importance of Accurate Touch-down Angle Control

In this Section we provide some insight into the importance of accurate touch-down angle control for hopping robots. All step length algorithms in this dissertation essentially require this as a necessary condition. Lack of this required precision is responsible for the absence of experimental results regarding 2D foothold placement on the robot FRANK. As we can observe in Fig. 2.1, the step length for a given step is given as:

$$S = S_{TO} + S_{flight} + S_{TD} \tag{D.1}$$

Where

$$S_{TD} = L_0 \cos(\theta_{TD}) \tag{D.2}$$

Clearly, S cannot be completely determined without defining the touch-down angle θ_{TD}. The duration of the ballistic term S_{flight} is also a function of this touch-down angle. Lastly, any feed-forward calculations that consider one or more steps as a lookahead horizon (such as the centroid-based method from chapter 5) also rely on the leg angle

being accuractly controlled. Therefore, regulating a step length is coupled to setting a touch-down angle. Controlling the robot to a higher energy state can alleviate this problem, as the ballistic term will dominate. However, this is typically constrained to some maximum ceiling by robot parameters. The analysis presented here concerns both SLIP and the SEA 2D Hopper (FRANK with a locked body). When the robot's body is able to rotate freely, visualization of the reachable space becomes more difficult as the body angle and angular rotation rate must also be considered. However, the same requirements on precise touch-down angles and basic trends of the reachable space still apply.

D.1 Effect of TO/TD Angle Noise on Reachable Space

The reachable space $\mathcal{R}$ of the system is typically calculated at the apex state $\mathcal{A}$, as this requires only two variables and thus can be visualized on a 2D plane. Methods for analytical approximating this 2D set were provided in Chapter 5. It is also possible, however, to visualize the reachable space of the system by considering the take-off states $\dot{L}_{TO}$, $\dot{\theta}_{TO}$, and θ_{TO} at the instant after take-off has occurred, as these variables completely determine the ballistic trajectory of the CoM of the system. An example of such a visualization is shown in Fig. D.1. The color represents the total sum of the square of the SEA displacement during the stance phase, which approximately represents total energy injected into the system by the SEA. Here we see that the reachable space in fact lives on a 2D surface, even when plotted with the 3 take-off variables. When we use our centroid-based method to determine the commanded touch-down angles, the binary search makes a future guess one step ahead of the next reachable space. Thus, any errors on the next commanded touch-down angle will appear as errors in the *initial condition* of the reachable space of the *next* step. To visualize the effect of varying the initial

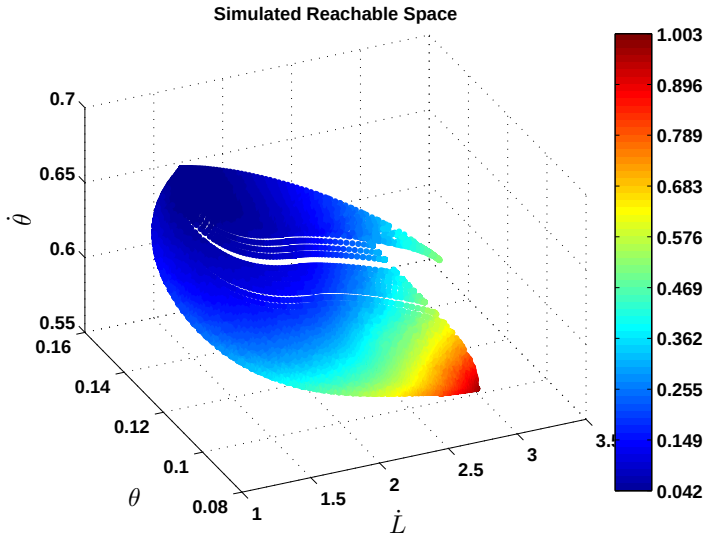


Figure D.1: A simulated reachable space for the robot FRANK is shown, plotted in take-off coordinates. The sharp edges are due to boundary conditions with the SEA involving cases when the SEA impacts a hard limit.

touch-down angle, simulations were performed with lower resolution for a large set of touch-down angle initial condition. The results of these simulations are shown in Fig. D.2, where it is clear the effect of changing the initial touch-down angle is a translation of the reachable space, similar to what we saw in Chapter 5.

Therefore, we expect random noise on the initial touch-down angle to result in a large 3D cube, made up of several stacks of the nominal reachable space. Simulation results applying random noise to the same reachable space we saw in Fig. D.1 illustrates that this is indeed the case. As we see, it is very difficult to predict the reachable space of the system if we cannot properly control the leg angle.

D.2 Leg Angle Control on FRANK

Appendix A provided system identification results for both the leg angle and body inertial values for FRANK via separate, decoupled hardware experiments. However,

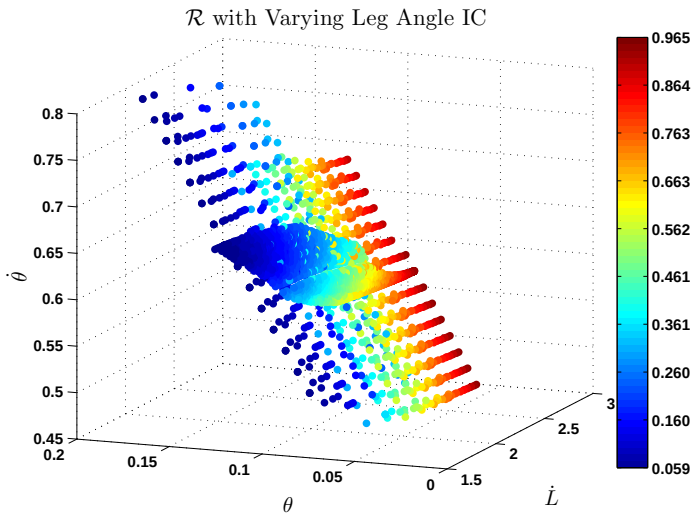


Figure D.2: As the touch-down initial condition is varied, the reachable space is seen to translate along a line, effectively creating vertical slanted stacks of the original reachable space for a set of initial touch-down angles. In this example the touch-down angle is only varied by +- 2 degrees from the optimal angle of 3 degrees.

these experiments intentionally eliminated any coupling between the body and the leg by manually locking the other variable down in each case. In an attempt to quantify the frequency characteristics of the torsional dynamics in the boom, system identification was performed an additional time via mounting one side of the robot's body on a large low-friction wheel, and applying swept sine *without* manually fastening the body down, but with the body still mechanically locked on the boom side. The resulting frequency response along with high-order transfer function approximations are shown in Figures D.4 and D.5.

Torsional forces cause the boom angle to flex during operation, as was briefly discussed in Chapter 4, which results in several resonant peaks appearing in the frequency response. Although the torsional forces are much alleviated when the body is unlocked, this resonance still appears, as shown in Fig. D.6. Unfortunately, this is not the only source of torsional vibration in the system, as the top component of the vertical boom assembly also flexes during operation. In order to estimate this small body rotation, an

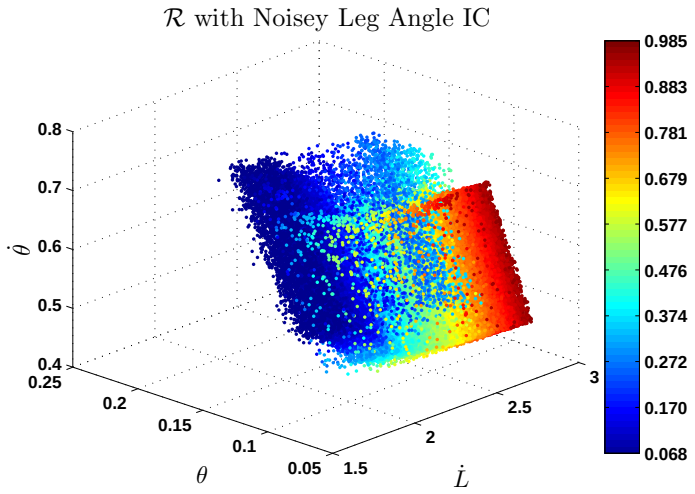


Figure D.3: When random noise is added to the initial touch-down angle, the reachable space expands considerably, making feed-forward based control potentially prohibitively difficult.

IMU and HC-SR4 distance sensor based measurement of the FRANK's body was implemented, however, the quality of the body angle estimation was far from perfect. In general, our system is currently only capable of estimating the body angle to 1-2 degrees of accuracy. This means we often expect touch-down angle errors of our system to be significant.

It is generally possible to design controllers with the purpose of rejecting these torsional disturbances. Using our linear transfer function model in Fig. D.5, which are above order 8 and each have at least one non-minimum phase zero, a touch-down angle controller was implemented using LQR/LQG Loop Gain Recovery. While a controller can reject these disturbances, in general, it cannot act fast enough to completely damp them out before the termination of the ballistic phase. The controller must also of course re-position the leg before touch-down. Reference tracking during hopping can be seen in Fig. D.7, where it is clear the controller has some difficulty regulating the correct touch-down angle.

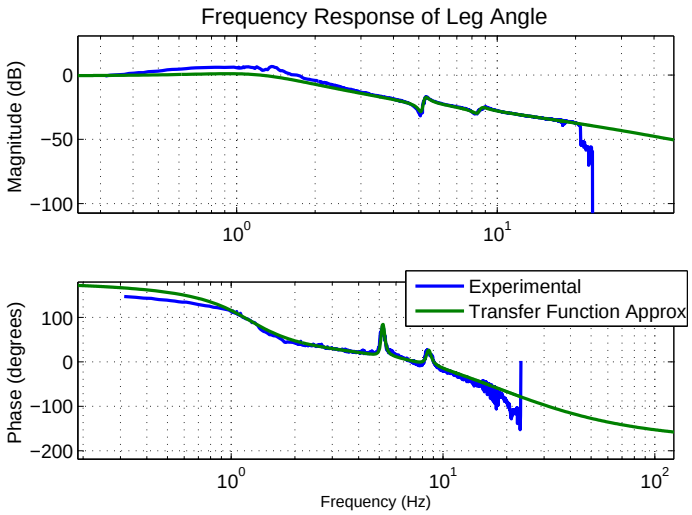


Figure D.4: Frequency response of the leg angle with the body locked on the boom side, but still allowed to torsion while swept-sine was applied. The order of the fit transfer function here is 8, and has one non-minimum phase zero.

D.3 Reachability Experiments on FRANK

The reachable space of the robot FRANK, with the body mechanically locked, was experimentally mapped for the initial condition apex state $\mathcal{A}_\prime = (\dot{x}, y) = (0.45\frac{m}{s}, 0.75m)$, with touch-down angle $\theta_0 = 3deg$. The experiment performed involved repeatably driving the system approximately to the state $\mathcal{A}_\prime$. When the algorithm detected the forward velocity and apex height were acceptably close to the target command, the touch-down angle θ_0 was commanded for the subsequent step, and the SEA parameters t_s and L_{a0} were chosen from a set of vectors spanning all possible combinations. Approximately 1000 data points were measured. The measured reachable space, plotted in take-off coordinates, is shown in Fig. D.8. The theoretical reachable space generated via simulation of the SEA 2D Hopper is also shown, and at first glance it appears to significantly mis-match the measured data. However, as we saw in Fig. D.3, we know the effect of noise on the reachable space distorts the shape significantly. Therefore, the mis-match is explained due to variation of the body and touch-down angle at each step. This can be confirmed

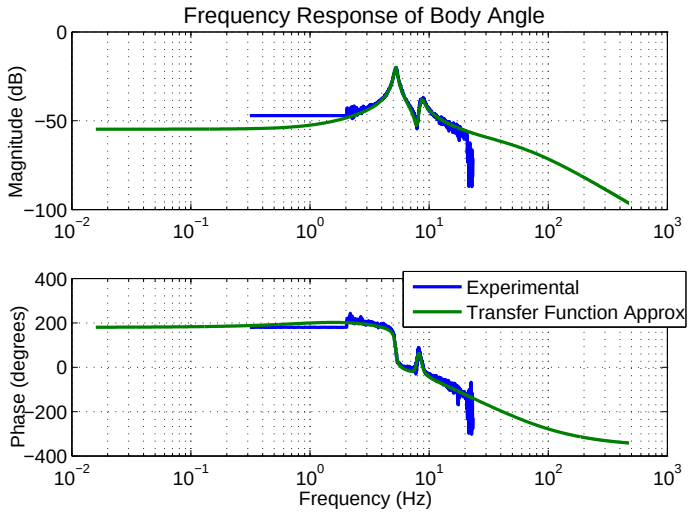


Figure D.5: Frequency response of the corresponding body angle for the modified swept sine experiment. The order of the fit transfer function here is 9, and has 2 non-minimum phase zeros.

by storing the initial conditions of the leg and body angle from each experiment. The sum of the body and the leg angle is defined as the *global leg angle*. The measured data can be approximately reproduced in simulation by careful post-processing of the data, and storing the global leg angle for each experiment. Simulating the SEA 2D Hopper using the stored global leg angle as corresponding initial conditions at each step produces very similar results, seen in Fig. D.9. As we can see, the hardware FRANK is currently unable to reliably drive the system to a desired apex state due to inaccurate touch-down angle control.

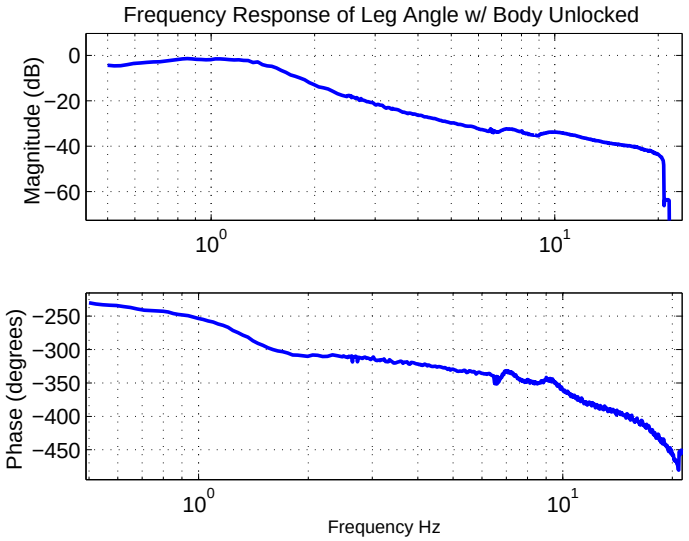


Figure D.6: This data represents modified swept-sine results with the body's mechanical lock loosened to allow the body to rotate up to roughly 15 degrees, and approximately represents operation when the body is unlocked. While it is difficult to accurately measure ϕ for this case, we can still see resonance peaks due to the torsional dynamics in the response for the leg angle, albeit somewhat attenuated.

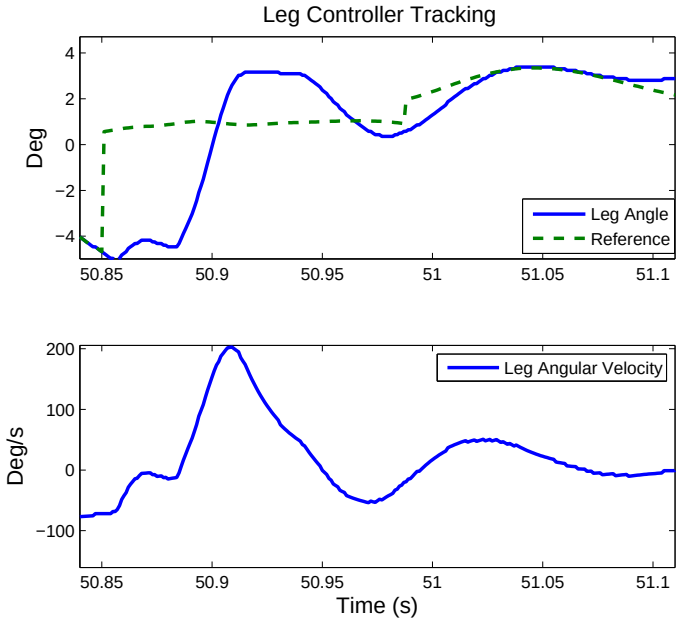


Figure D.7: Leg angle reference tracking during forward 2D hopping on the hardware FRANK. We see there are oscillations in the leg angle tracking, mostly due to the torsional effects of the boom. The reference command cannot be constant, as the body angle estimation must be included to set the global touch-down angle.

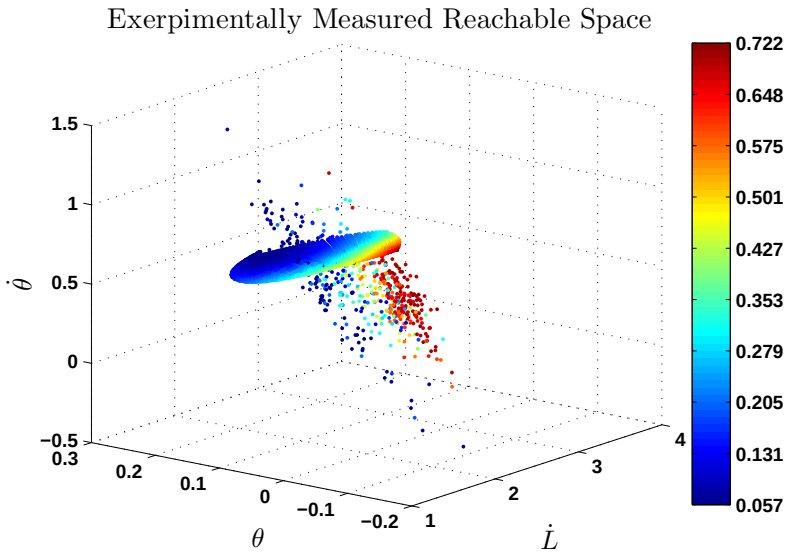


Figure D.8: Here we see the experimentally measured reachable space overlayed on the theoretical simulated reachable space. Essentially what we see here are multiple stacks of different reachable spaces corresponding to touch-down angle noise.

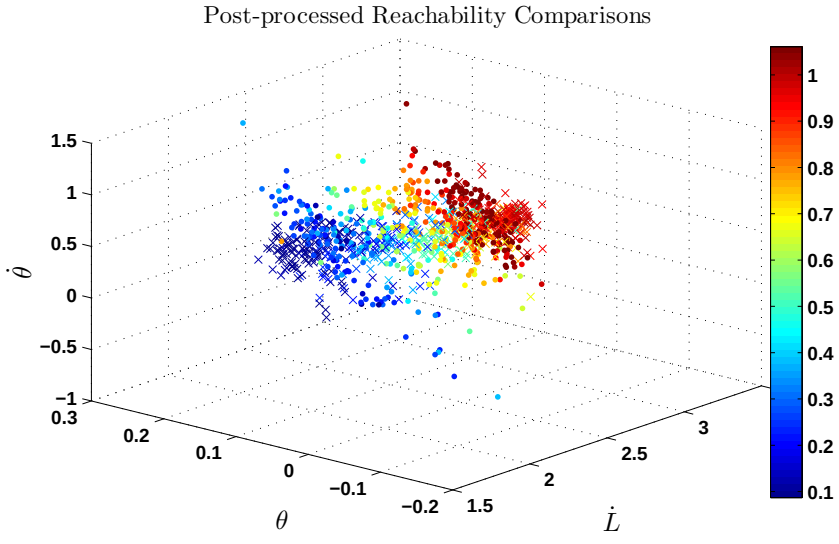


Figure D.9: Accounting for the varying initial condition of the body angle and leg angle allows us to reasonably reproduce the experimental data in simulation. The reachable space of the robot FRANK is very difficult to predict in real-time in the presence of noise on the touch-down angles. In this figure, the circles represent simulated hops and the x's represent measured data.

Bibliography

[1] C. T. Farley and D. P. Ferris, *Biomechanics of walking and running: Center of mass movements to muscle action*, *Exercise and Sport Science Reviews* **26** (1999) 253–283.

[2] T. M. Lejeune, P. A. Willems, and N. C. Heglund, *Mechanics and energetics of human locomotion on sand.*, *Journal of Experimental Biology* **201** (1998), no. 13 2071–2080.

[3] C. Atkeson, B. Babu, N. Banerjee, D. Berenson, C. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, P. Franklin, *et. al.*, *What happened at the DARPA robotics challenge, and why?*, *The Journal of Field Robotics (in review)*.

[4] R. M. Alexander, *Three uses for springs in legged locomotion*, *Int. Journal of Robotics Research* **9** (1990), no. 2 53–61.

[5] H. Geyer, A. Seyfarth, and R. Blickhan, *Compliant leg behaviour explains basic dynamics of walking and running*, in *Proc. of the Royal Society*, pp. 2861 – 2867, 2006.

[6] G. A. Cavagna, H. Thys, and A. Zamboni, *The sources of external work in level walking and running*, *Physiology* **262** (1976) 639–657.

[7] K. Sasaki and R. R. Neptune, *Muscle mechanical work and elastic energy utilization during walking and running near the preferred gait transition speed*, *Gait and Posture* **23** (2006) 383–390.

[8] M. H. Raibert, M. Chepponis, and H.Brown, *Running on four legs as though they were one*, *IEEE Journal of Robotics and Automation* **2** (1986), no. 2 70–82.

[9] R. Blickhan and R. J. Full, *Similarity in multilegged locomotion: Bouncing like a monopode*, *Journal of Comparative Physiology A: Neurothology, Sensory, Neural, and Behavioral Physiology* **173** (Nov., 1993) 509–517.

[10] A. Seyfarth, H. Geyer, M. Gunther, and R. Blickhan, *A movement criterion for running*, *Journal of Biomechanics* **35** (2002) 649–655.

[11] R. M. Ghigliazza, R. A. P. Holmes, and D. Koditschek, *A simply stabilized running model*, *SIAM Rev.* **47** (2005).

[12] M. Ahmadi and M. Buehler, *Stable control of a simulated one-legged running robot with a hip and leg compliance*, *IEEE Transactions on Robotics* **13** (1997), no. 1.

[13] J. G. D. Karssen, M. Haberland, M. Wisse, and S. Kim, *The optimal swing-leg retraction rate for running*, in *IEEE Int. Conf. on Robotics and Automation*, 2011.

[14] B. Andrews, B. Miller, J. Schmitt, and J. E. Clark, *Running over unknown rough terrain with a one-legged planar robot*, *Bioinspiration & Biomimetics* **6** (2011), no. 2 026009.

[15] R. Blickhan, *The spring-mass model for running and hopping*, *Journal of Biomechanics* (1989), no. 22 1217–1227.

[16] G. Piovan and K. Byl, *Reachability-based control for the active slip model*, *The International Journal of Robotics Research* **34** (2015), no. 3 270–287.

[17] P. Wensing and D. Orin, *High-speed humanoid running through control with a 3d-slip model*, in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 5134–5140, Nov, 2013.

[18] A. M. S. S. Seyed Hossein Tamaddoni, Farid Jafari, *Biped hopping control based on spring loaded inverted pendulum model*, *International Journal of Humanoid Robotics* **7** (2010), no. 3 263–280.

[19] D. Dudek and R. J. Full, *Passive mechanical properties of legs from running insects*, *Bioinspiration and Biomimetics* **209** (2006) 15021515.

[20] J. Schmitt and P. Holmes, *Mechanical models for insect locomotion: dynamics and stability in the horizontal plane i. theory*, *Biological Cybernetics* **83** (2000), no. 6 501–515.

[21] Daley, M. A., F. G., and A. A. Biewener, *Running stability is enhanced by a proximo-distal gradient in joint neuromechanical control*, *Journal of Experimental Biology* **210** (2007), no. 3 383–394.

[22] H. Geyer, A. Seyfarth, and R. Blickhan, *Spring-mass running: simple approximate solution and application to gait stability*, *Journal of Theoretical Biology* **232** (2005), no. 1 315–328.

[23] W. J. Schwind and D. E. Koditschek, *Approximating the stance map of a 2 DoF monoped runner*, *Journal of Nonlinear Science* **10** (2000), no. 5 533–588.

[24] U. Saranlı, O. Arslan, M. Ankarali, and O. Morgül, *Approximate analytic solutions to the non-symmetric stance trajectories of the passive spring-loaded inverted pendulum with damping*, *Nonlinear Dynamics* **64** (2010), no. 4 729–742.

[25] H. Yu, M. Li, and H. Cai, *Approximating the stance map of the slip runner based on perturbation approach*, in *IEEE Int. Conf. on Robotics and Automation*, 2012.

[26] Z. Shen and J. Seipel, *A piecewise-linear approximation of the canonical spring-loaded-inverted-pendulum model of legged locomotion*, *Journal of Computational and Nonlinear Dynamics* (January, 2015).

[27] Ö. Arslan, U. Saranl, and Ö. Morgul, *Approximate stance map of the spring mass hopper with gravity correction for nonsymmetric locomotions*, in *IEEE Int. Conf. on Robotics and Automation*, 2009.

[28] G. Piovan and K. Byl, *Approximation and control of the slip model dynamics via partial feedback linearization and two-element leg actuation strategy*, *IEEE Transactions on Robotics* (in publication).

[29] S. Riese and A. Seyfarth, *Stance leg control: variation of leg parameters supports stable hopping*, *Bioinspiration & Biomimetics* **7** (2012), no. 1 016006.

[30] H. R. Vejdani and J. W. Hurst, *Swing leg control for actuated spring-mass robots*, in *Int. Conf. on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, 2012.

[31] J. Seipel and P. Holmes, *A simple model for clock-actuated legged locomotion*, *Regular and Chaotic Dynamics* **12** (2007), no. 5 502–520.

[32] K. Byl, M. Byl, M. Rutschmann, B. Satzinger, L. van Blarigan, G. Piovan, and J. Cortell, *Series-elastic actuation prototype for rough terrain hopping*, in *Proc. IEEE Int. Conf. on Tech. for Practical Robot App. (TePRA)*, pp. 103–110, 2012.

[33] J. Schmitt and J. Clark, *Modeling posture-dependent leg actuation in sagittal plane locomotion*, *Bioinspiration and Biomimetics* **4** (2009), no. 1 1–17.

[34] M. Rutschmann, B. Satzinger, M. Byl, and K. Byl, *Nonlinear model predictive control for rough terrain hopping*, in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2012.

[35] R. Full and D. Koditschek, *Templates and anchors: neuromechanical hypotheses of legged locomotion on land*, *Journal of Experimental Biology* **202** (1999), no. 23 3325–3332.

[36] R. Altendorfer, D. E. Koditschek, and P. Holmes, *Stability analysis of legged locomotion models by symmetry-factored return maps*, *International Journal of Robotics Research* (October, 2004).

[37] N. Cherouvim and E. Papadopoulos, *Single actuator control analysis of a planar 3dof hopping robot*, in *RSS*, 2005.

[38] M. H. Raibert, *Legged Robots that Balance*. MIT Press Series in Artificial Intelligence, Cambridge, Massachusetts, 1986.

[39] S. H. Hyon and T. Mita, *Development of a biologically inspired hopping robot-"kenken"*, in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 4, pp. 3984–3991 vol.4, 2002.

[40] S. Hyon and T. Emura, *Symmetric walking control: Invariance and global stability*, in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 1443–1450, April, 2005.

[41] I. Poulakakis and J. W. Grizzle, *The spring loaded inverted pendulum as the hybrid zero dynamics of an asymmetric hopper*, *IEEE Transactions on Automatic Control* **54** (Aug, 2009) 1779–1793.

[42] I. Poulakakis and J. W. Grizzle, *Formal embedding of the spring loaded inverted pendulum in an asymmetric hopper*, in *Proc. of the European Control Conference*, pp. 3159 – 3166, July 2007.

[43] I. Poulakakis and J. W. Grizzle, *Modeling and control of the monopedal robot thumper*, in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 3327 – 3334, May 2009.

[44] J. K. Hodgins and M. N. Raibert, *Adjusting step length for rough terrain locomotion*, *IEEE Transactions on Robotics and Automation* **7** (Jun, 1991) 289–298.

[45] G. Zeglin and B. Brown, *Control of a bow leg hopping robot*, in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 1, pp. 793–798 vol.1, May, 1998.

[46] G. Zeglin and B. Brown, *First hops of the 3d bow leg*, in *Int. Conf. on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, pp. 357–364, 2002.

[47] M. Ahmadi and M. Buehler, *Controlled passive dynamic running experiments with the ARL-monopod II*, *IEEE Transactions on Robotics* **22** (2006), no. 5 974–986.

[48] J. W. Hurst, J. Chestnutt, and A. Rizzi, *Design and philosophy of the bimasc, a highly dynamic biped*, in *IEEE Int. Conf. on Robotics and Automation*, 2007.

[49] D. Koepl and J. Hurst, *Impulse control for planar spring-mass running*, *Journal of Intelligent & Robotic Systems* **74** (2014), no. 3-4 589–603.

[50] M. H. Raibert, *Dynamic stability and resonance in a one-legged hopping machine*, *Theory and Practice of Robots and Manipulators, Proceedings of RoManSy'81* (1983) 352–367.

[51] M. H. Raibert and J. Brown, H. B., *Experiments in balance with a 2d one-legged hopping machine*, *ASME J. Dynamic Systems, Measurement, and Control* (1984) 106:75–81.

[52] M. H. Raibert, J. Brown, H. B., and M. Chepponis, *Experiments in balance with a 3d one-legged hopping machine*, *International J. Robotics Research* (1984) V3:75–92.

[53] P. Terry, G. Piovan, and K. Byl, *Towards precise control of hoppers: Using high order partial feedback linearization to control the hopping robot FRANK*, in *Proc. IEEE Conf. on Decision and Control*, 2016.

[54] P. Terry, G. Piovan, and K. Byl, *Reachability based and high order partial feedback linearization enforced control strategies for realistic series-elastic actuated hopping robots*, *IEEE Transactions on Robotics* (in review).

[55] P. Terry, G. Piovan, and K. Byl, *Slip-based and HOPFL enforced step length control for stable body motions of realistic SEA hoppers*, in *to be submitted*, 2016.

[56] P. Terry and K. Byl, *CoM control for underactuated 2d hopping robots with series-elastic actuation via higher order partial feedback linearization*, in *Proc. IEEE Conf. on Decision and Control*, 2015.

[57] P. Terry and K. Byl, *A higher order partial feedback linearization based method for controlling an underactuated hopping robot with a compliant leg*, in *Proc. IEEE Conf. on Decision and Control*, 2014.

[58] M. W. Spong, *Underactuated mechanical systems*, in *Control Problems in Robotics and Automation*, pp. 135–150. Springer, 1998.

[59] M. W. Spong, *Partial feedback linearization of underactuated mechanical systems*, in *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, pp. 314–321, 1994.

[60] M. W. Spong, *Swing up control of the acrobot using partial feedback linearization*, in *Proc. American Control Conference (ACC)*, pp. 2158–2162, 1994.

[61] G. Piovan and K. Byl, *Enforced symmetry of the stance phase for the spring-loaded inverted pendulum*, in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 1908–1914, 2012.