Layered Control Architectures: Constructive Theory and Application to Legged Robots

Thesis by
Noel Csomay-Shanklin

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy, Control and Dynamical Systems

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2025
Defended May 28, 2025

Noel Csomay-Shanklin
ORCID: 0000-0002-2361-1694

ACKNOWLEDGEMENTS

I would be remiss if I didn't begin by thanking my parents, Enikő Csomay and Trevor Shanklin, for their plentiful support throughout my many, many years of schooling. Between encouraging me to keep working and to keep taking breaks, respectively, I've been able to find myself as a researcher, and as a person. Thank you also to the rest of my family, those still with us and those no longer — George and Doreen Shanklin, Chris Shanklin, Maria Horváth, and Tibor Csomay.

To my advisor, Aaron Ames, it has been almost ten years since we first met. I am grateful for the opportunity you've given me to develop into the researcher that I am today — my world is brighter knowing what I know now. To my undergraduate advisor, Aaron Young, thank you for giving me the platform and guidance to get me started with research. Thank you also to the many mentors I had in industry along the way — Alex Brinkman, Lanny Smoot, and Farbod Farshidian.

This experience would have paled in comparison to what it was without those around me in and out of the lab. To my mentors along the way, I thank you for your guidance and help. Eric Ambrose, you were the first one to take a chance on me. Under your mentorship, I developed from machining blocks of metal to getting my feet wet with control. The summer I spent working with you at Caltech was a true highlight and convinced me that I wanted to go to grad school. Wen-Loong Ma, you took me in as an undergrad when I was in my first year, and exposed me to so much knowledge and experience. You jump-started my research career, and I still think about your mentorship. Jenna Reher, you showed me how to make beautiful figures, and I learned so much from you during our shared internship. Maegan Tucker, thank you for enduring endless research debates and always being good company in the lab.

Andrew Taylor, when I decided I wanted to dip my toes into theory, you welcomed me with open arms. From being a tolerant TA to an encouraging collaborator, I will always hold the precision of thought and mathematical rigor that you instilled in me. Ugo Rosolia, along the way you guided me into the ways of optimal control, MPC, and a hierarchical perspective. Your research philosophy shaped much of my outlook on control. To the other postdocs in the lab — Preston Culbertson, Vince Kurtz, Jeesop Kim, Jaemin Lee, Ryan Bena, Pio Ong, Max Cohen — I've enjoyed our many conversations, and each discussion has carved a new facet into my understanding of the fields of robotics and control.

To my peers, thanks for the time we spent together, split equally between laughs and commiserations. Ryan Cosner, a good friend and colleague, we are in it from the start to the end. Wyatt Ubellacker, for showing me that it is ok to still code in Vim, for many discussions and projects together, and for referring me to my upcoming job. Amy Li, for our conversations, coffee runs, and walks around campus. Will Compton, you've been a great partner in crime, and I hope I haven't soiled your research outlook too much. From the 100^{th} hopper experiment to blackboards full of math, the topics we've explored together have added so much richness to my education. This extends to you too Ivan Jimenez Rodriguez — thank you for helping me get into learning for control. To Min Dai, Skylar Wei, Angela Gao, and Berthy Feng — the first year would have been significantly more strenuous if it weren't for our collective problem set sessions. And to the younger students in the lab — Zach Olkin, Serigo Esteban, Adrian Boedtker Ghansah, Gilbert Bahati, Albert Li, Gary Yang — "just keep pushing."

I would also like to say a resounding thank you to the friends I have made along the way (besides those listed above, who I also consider friends). For fear of omitting someone and never being able to add them, I will not list all of you — but you know who you are. Without you, life would not have the brightness that it does. To the mountains — you are my refuge, and you keep me sane. Finally, to my partner Hannah: from tolerating my crazy schedules to our evening dance parties with Kaio dog, I cherish every moment with you. I can't wait to continue exploring our shared passions, and for our future together.

ABSTRACT

Fueled in part by the imagination of science fiction, every decade since the 1950s has expected robots to enter our everyday lives in the subsequent decade. Despite this anticipation, the widespread adoption of robots has consistently fallen short of societal expectations. This delay is attributable to the sheer variety of complexities in robotics — perception, contact-rich dynamics, human-robot interactions. Each sub-discipline of robotics poses unique challenges that must be addressed to achieve general autonomy. As progress is made in these sub-fields, it is increasingly important to adopt a *layered architecture* perspective that combines isolated controller blocks into a unified framework.

This thesis argues that on the road to general autonomy, adopting layered architectures enables three key benefits: *efficiency*, *feasibility*, and *generalizability*. We root our discussion in a general problem in robotics: the design of a controller that navigates a robot to a goal state while satisfying all state and input constraints that are present. Throughout the thesis, we focus on solutions that are both general — applicable across a wide variety of robotic platforms — and concrete — deployed and tested on specific hardware platforms. As such, we aim to not only propose a framework for reasoning about this problem, but also methods to synthesize controllers that solve it in practice for legged robots.

We begin by motivating and formalizing the notion of layered architectures and use this to build our control stack from the bottom up. We start with low-level planning and tracking layers that stabilize the system within a tracking tube for both the actuated and underactuated states of legged robots. We then introduce high-level planning and tracking layers that generate and follow sparse, dynamically feasible graphs for coarse global navigation through cluttered environments. By decomposing the global control problem into interacting levels and layers, each operating with disparate timescales and system abstractions, we enable tractable, reliable, and extensible robot autonomy.

Throughout this thesis, an emphasis will be placed on mathematical structure, constructive synthesis, and experimental validation. We demonstrate that adopting a layered architecture perspective is not merely an implementation convenience, but a fundamental organizing principle that can enable true robot autonomy.

PUBLISHED CONTENT AND CONTRIBUTIONS

[1] N. Csomay-Shanklin, W. D. Compton, and A. D. Ames, "Hierarchies in motion: From layered control architectures to perceptive 3D hopping," *Submitted to IEEE Transactions on Robotics*, 2025,
N.C.-S. architected this project, developed a significant portion of the code architecture running on the robot, and collaborated on the mathematical proofs, plotting, and writing of the manuscript.

[2] M. H. Cohen, N. Csomay-Shanklin, W. D. Compton, T. G. Molnar, and A. D. Ames, "Safety-critical controller synthesis with reduced-order models," *arXiv preprint arXiv:2411.16479*, 2024,
N.C.-S. contributed to some of the mathematical formulations in the manuscript, and aided with writing and proofreading.

[3] W. D. Compton, N. Csomay-Shanklin, C. Johnson, and A. D. Ames, "Dynamic tube mpc: Learning tube dynamics with massively parallel simulation for robust safety in practice," *Submitted to ICRA 2025. arXiv preprint arXiv:2411.15350*, 2024,
N.C.-S. contributed to the design of the project, the experimental demonstration, and the writing of the manuscript.

[4] W. D. Compton*, I. D. J. Rodriguez*, N. Csomay-Shanklin*, Y. Yue, and A. D. Ames, "Constructive nonlinear control of underactuated systems via zero dynamics policies," in *2024 IEEE 63rd Conference on Decision and Control (CDC)*, IEEE, 2024, pp. 8350–8357,
N.C.-S. collaborated on the design and execution of this project, the mathematical proofs, plotting, and writing of the manuscript, and developed code that ran on the robot.

[5] N. Csomay-Shanklin and A. D. Ames, "Bezier reachable polytopes: Efficient certificates for robust motion planning with layered architectures," *Submitted to ACC 2025. arXiv preprint arXiv:2411.13506*, 2024,
N.C.-S. designed this project, completed the mathematical proofs, and wrote the manuscript.

[6] N. Csomay-Shanklin, W. D. Compton, and A. D. Ames, "Dynamically feasible path planning in cluttered environments via reachable bezier polytopes," *Submitted to ICRA 2025. arXiv preprint arXiv:2411.13507*, 2024,
N.C.-S. designed this project, developed a significant portion of the code architechture running on the robot, and collaborated on the mathematical proofs, plotting, and writing of the manuscript.

[7] N. Csomay-Shanklin*, W. D. Compton*, I. D. J. Rodriguez*, E. R. Ambrose, Y. Yue, and A. D. Ames, "Robust agility via learned zero dynamics policies," in *2024 IEEE/RSJ International Conference on Intelligent Robots and*

Systems (IROS). arXiv preprint arXiv:2409.06125, IEEE, 2024, pp. 9116–9123,
N.C.-S. collaborated on the design and execution of this project, the mathematical proofs, plotting, and writing of the manuscript, and developed code that ran on the robot.

[8] I. Incer, N. Csomay-Shanklin, A. D. Ames, and R. M. Murray, "Layered control systems operating on multiple clocks," *IEEE Control Systems Letters*, 2024,
N.C.-S. wrote code that ran on the robot and aided with writing and proofreading.

[9] W. Ubellacker, N. Csomay-Shanklin, and A. D. Ames, "Approximating regions of attraction via flow-control barrier functions and constrained polytope expansion," in *2024 American Control Conference (ACC)*, 2024, pp. 3383–3390. DOI: 10.23919/ACC60939.2024.10644985,
N.C.-S. contributed to some of the mathematical formulations in the manuscript, and aided with writing and proofreading.

[10] N. Csomay-Shanklin, V. D. Dorobantu, and A. D. Ames, "Nonlinear model predictive control of a 3D hopping robot: Leveraging lie group integrators for dynamically stable behaviors," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 12 106–12 112,
N.C.-S. architechted this project, developed a significant portion of the code architechture running on the robot, and collaborated on the mathematical proofs, plotting, and writing of the manuscript.

[11] M. Tucker, N. Csomay-Shanklin, and A. D. Ames, "Robust bipedal locomotion: Leveraging saltation matrices for gait optimization," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 12 218–12 225,
N.C.-S. ran hardware experiments and aided with revisions on the manuscript.

[12] Y. Chen, U. Rosolia, W. Ubellacker, N. Csomay-Shanklin, and A. D. Ames, "Interactive multi-modal motion planning with branch model predictive control," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5365–5372, 2022,
N.C.-S. aided with experiments, and helped proofread the manuscript.

[13] N. Csomay-Shanklin, M. Tucker, M. Dai, J. Reher, and A. D. Ames, "Learning controller gains on bipedal walking robots via user preferences," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 10 405–10 411,
N.C.-S. collaborated on the design and execution of this project, plotting, and writing of the manuscript, and wrote code that ran on the robot.

[14] N. Csomay-Shanklin*, A. J. Taylor*, U. Rosolia, and A. D. Ames, "Multi-rate planning and control of uncertain nonlinear systems: Model predictive control and control lyapunov functions," in *2022 IEEE 61st Conference on*

Decision and Control (CDC), IEEE, 2022, pp. 3732–3739,
N.C.-S. collaborated on the design and execution of this project, the mathematical proofs, plotting, and writing of the manuscript.

[15] M. Y. Galliker*, N. Csomay-Shanklin*, R. Grandia, A. J. Taylor, F. Farshidian, M. Hutter, and A. D. Ames, "Planar bipedal locomotion with nonlinear model predictive control: Online gait generation using whole-body dynamics," in *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, IEEE, 2022, pp. 622–629,
N.C.-S. collaborated on the design and execution of this project, plotting, and writing of the manuscript, and wrote code that ran on the robot.

[16] I. D. J. Rodriguez*, N. Csomay-Shanklin*, Y. Yue, and A. D. Ames, "Neural gaits: Learning bipedal locomotion via control barrier functions and zero dynamics policies," *arXiv preprint arXiv:2204.08120*, 2022,
N.C.-S. collaborated on the design and execution of this project, plotting, and writing of the manuscript, and wrote code that ran on the robot.

[17] N. Csomay-Shanklin*, R. K. Cosner*, M. Dai*, A. J. Taylor, and A. D. Ames, "Episodic learning for safe bipedal locomotion with control barrier functions and projection-to-state safety," in *Learning for dynamics and control*, PMLR, 2021, pp. 1041–1053,
N.C.-S. collaborated on the design and execution of this project, the mathematical proofs, plotting, and writing of the manuscript, and developed code that ran on the robot.

[18] W.-L. Ma, N. Csomay-Shanklin, S. Kolathaya, K. A. Hamed, and A. D. Ames, "Coupled control lyapunov functions for interconnected systems, with application to quadrupedal locomotion," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3761–3768, 2021,
N.C.-S. aided with experiments, helped proofread the manuscript, and wrote code that ran on the robot.

[19] Y. Sun, W. L. Ubellacker, W.-L. Ma, X. Zhang, C. Wang, N. Csomay-Shanklin, M. Tomizuka, K. Sreenath, and A. D. Ames, "Online learning of unknown dynamics for model-based controllers in legged locomotion," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 8442–8449, 2021,
N.C.-S. aided with experiments, and helped proofread the manuscript.

[20] M. Tucker, N. Csomay-Shanklin, W.-L. Ma, and A. D. Ames, "Preference-based learning for user-guided hzd gait generation on bipedal walking robots," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 2804–2810,
N.C.-S. contributed to some of the mathematical formulations in the manuscript, and aided with writing and proofreading, and wrote code that ran on the robot.

[21] W. Ubellacker, N. Csomay-Shanklin, T. G. Molnar, and A. D. Ames, "Verifying safe transitions between dynamic motion primitives on legged robots," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 8477–8484,
N.C.-S. contributed to some of the mathematical formulations in the manuscript, and aided with writing and proofreading.

[22] W.-L. Ma, N. Csomay-Shanklin, and A. D. Ames, "Coupled control systems: Periodic orbit generation with application to quadrupedal locomotion," *IEEE Control Systems Letters*, vol. 5, no. 3, pp. 935–940, 2020,
N.C.-S. aided with experiments, helped proofread the manuscript, and wrote code that ran on the robot.

[23] W.-L. Ma, N. Csomay-Shanklin, and A. D. Ames, "Quadrupedal robotic walking on sloped terrains via exact decomposition into coupled bipedal robots," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 4006–4011,
N.C.-S. aided with experiments, helped proofread the manuscript, and wrote code that ran on the robot.

[24] J. Reher, N. Csomay-Shanklin, D. L. Christensen, B. Bristow, A. D. Ames, and L. Smoot, "Passive dynamic balancing and walking in actuated environments," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 9775–9781,
N.C.-S. aided with experiments, helped proofread the manuscript, and wrote code that ran on the robot.

[25] E. Ambrose, N. Csomay-Shanklin, Y. Or, and A. D. Ames, "Design and comparative analysis of 1d hopping robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 5717–5724,
N.C.-S. aided with experiments, helped proofread the manuscript, and helped with the design of the robot.

RELEVANT PUBLISHED VIDEO CONTENT

[1] *Quadrupedal robotic walking on sloped terrains.* https://youtu.be/uJedboyzDjc.

[2] *Episodic learning for safe bipedal locomotion with control barrier functions and projection-to-state safety*, https://vimeo.com/481809664.

[3] *Preference-based learning for user-guided hzd gait generation on bipedal robots*, https://youtu.be/rLJ-m65F6C4.

[4] *Learning controller gains on bipedal walking robots via user preferences.* https://youtu.be/jMX5a_6Xcuw?si=w9KSizYj6S16A66F.

[5] *Learning to walk by enforcing forward invariance*, https://www.youtube.com/watch?v=8TeXd0AYtpA.

[6] *Bipedal locomotion with nonlinear model predictive control*, https://youtu.be/3g8ZNsCWdOA.

[7] *3D hopping with geometric model predictive control*, https://youtu.be/PqjFwrshL-o.

[8] *Robust agility via learned zero dynamics policies: 3D hopping.* https://vimeo.com/923800815.

[9] *Obstacle avoidance on a 3D hopping robot*, https://vimeo.com/1009702220.

[10] *Indoor hopping compilation.* https://youtu.be/k3YuoKA4HNk.

[11] *Outdoor hopping compilation.* https://youtu.be/WnPIML8oG78.

[12] *Hierarchies in motion: From layered architechtures to perceptive 3D hopping*, https://youtu.be/-PHxfJALLS8.

TABLE OF CONTENTS

NOMENCLATURE

CARE. Continuous Algebraic Riccati Equation.

CBF. Control Barrier Function.

CL. Closed Loop.

CLF. Control Lyapunov Function.

CTLE. Continuous Time Lyapunov Equation.

DARE. Discrete Algebraic Riccati Equation.

DDP. Differential Dynamic Programming.

DTLE. Discrete Time Lyapunov Equation.

E-CBF. Exponential Control Barrier Function.

E-ISS. Exponential Input to State Stability.

ERM. Empirical Risk Minimization.

ES-CLF. Exponentially Stable Control Lyapunov Function.

FB. Feedback.

FBL. Feedback Linearization.

FTOCP. Finite-Time Optimal Control Problem.

GCS. Graph of Convex Sets.

HJB. Hamilton-Jacobi-Bellman.

HZD. Hybrid Zero Dynamics.

IL. Imitation Learning.

iLQR. Iterative Linear Quadratic Regulator.

ISS. Input to State Stability.

LIP. Linear Inverted Pendulum.

LP. Linear Program.

LQR. Linear Quadratic Regulator.

MCMC. Markov Chain Monte Carlo.

MPC. Model Predictive Control.

OCP. Optimal Control Problem.

ODE. Ordinary Differential Equation.

PBL. Preference Based Learning.

PD. Proportional Derivative.

PDE. Partial Differential Equation.

PID. Proportional Integral Derivative.

PMP. Pontryagin's Maximum Principle.

PRM. Probabilistic Roadmap.

PSSf. Projection to State Safety.

QP. Quadratic Program.

RL. Reinforcement Learning.

RRT. Rapidly-exploring Random Tree.

SLIP. Spring Loaded Inverted Pendulum.

SOCP. Second-Order Cone Program.

SQP. Sequential Quadratic Programming.

ZD. Zero Dynamics.

ZDP. Zero Dynamics Policies.

ZMP. Zero Moment Point.

Chapter 1: Introduction

Contents

Layered architectures enable efficient, feasible, and generalizable controller design.

1.1 Motivation of Layered Architectures

Fueled in part by the imagination of science fiction, every decade since the 1950s has anticipated that robots would enter our everyday lives in the subsequent decade. Despite this, the widespread adoption of robots has consistently fallen short of societal expectations. Modern robotic systems are capable of impressive demonstrations in isolation — from running half marathons [1] to autonomously folding laundry [2] — but they often remain fragile, specialized, and constrained to narrow operational domains. This gap between expectation and reality — a gap widened by the egos of engineers and entrepreneurs alike — stems from the sheer variety of complexities inherent to the field of robotics. Perception systems must interpret noisy, partially observable states to infer structure of the environment. Planning models must reason about how current actions affect future states under uncertainty and contact-rich dynamics. Robots must interact with humans in ways that are both explainable and safe. All sub-disciplines of robotics pose unique challenges that must be addressed to achieve general autonomy.

Crucially, it is not enough for each subsystem to work well in isolation; their interactions must be coherent to ensure reliable system-level behavior. This challenge is exacerbated by the fact that each subsystem operates on different timescales, relies on distinct modeling assumptions, and faces unique sources of uncertainty. While individual components are implemented independently, their interconnections are often handled implicitly — ensuring only that one block can be "plugged into" another. This lack of structured coordination leads to brittle systems that can fail unexpectedly when implicit assumptions between components are violated.

As progress continues within individual subfields, it becomes increasingly important to adopt a systems-level perspective — one that makes the interaction between components explicit and provides tools for reasoning about their coordination. The road to general autonomy is not via monolithic design, but rather by exploiting the domain knowledge of each of these sub-disciplines and combining them in a structured and modular way. This holistic perspective of controller synthesis is captured by the notion of *layered architectures*, which combine isolated blocks into a unified framework.

1.2 From Nature to Engineering, Layered Architectures are Universal

Layered architectures (often referred to as hierarchies) exist in a far broader sense than just their application to control systems. Social hierarchies emerge naturally

across a wide array of species, structuring interactions and enabling cohesion within groups of animals. Humans themselves are intrinsically hierarchical — the nervous and muscular systems, for example, combine multiple sensing and actuation modalities to balance the trade-off between speed and flexibility [3]. Even human cognition is hierarchical [4], splitting attention into slower planning layers and faster feedback layers. The pervasiveness of layered architectures across nature underscores their relevance and utility, making them a compelling subject of study.

Within engineered systems, layered architectures are similarly ubiquitous for their ability to divide and conquer complex tasks by breaking them down into manageable sub-problems. Early examples of hierarchical controllers appear in the context of rocket navigation in the 1950s [5] and control of the Apollo Lunar Module in the 1960s [6], where engineers found that guiding a fast, dynamic vehicle was best achieved by splitting the control task into a fast inner loop that maintains stabilization and a slower outer loop that guides towards targets. Around the same time, Shakey the Robot — the world's first mobile robot to perceive and reason about its surroundings — utilized a multi-level control architecture consisting of low-level feedback loops, mid-level planners, and high-level symbolic reasoning [7]. Shakey was the first autonomous robot to use a layered architecture, and this three-level hierarchy still serves as a template for the architectures explored in this thesis.

As the use of control hierarchies became ubiquitous in a broad collection of application domains, the need for a theory of these layered architectures also grew. This need was first addressed in 1970 [8], in which a theory of abstraction and coordination across levels of hierarchical systems was developed. The motivation was clear: rather than designing a single controller, complex processes could be more effectively controlled by an upper level setting targets for lower levels to execute. Hierarchical control proved especially advantageous when different levels could operate at different time scales or levels of detail – a principle that became a hallmark of modern control engineering.

Today, hierarchies remain a central pillar for controller design — from flight control systems [9] to the DARPA robotics challenges [10], [11], there are countless examples of layered architectures used in practice (see [12] for a recent overview). Given their ubiquitousness, the need for a theory of layered architectures has again sparked growing interest in recent years, with several works advancing the theoretical framework for control hierarchies [13]–[15]. Such theories have ranged from highly abstract instantiations of blocks and their interconnections [16] to very

concrete formulations investigating the trade-off between speed and flexibility in the coordination of planning and tracking layers [17]. The goal of this thesis is to follow along this lineage and provide a structured approach to designing, analyzing, and deploying layered architectures for robotic systems, grounded in theory and validated through experiment on legged robots.

1.3 The Pros and Cons of Layered Architectures

Layered architectures provide structural advantages that go beyond computational tractability, offering fundamental benefits that enable scalable, reliable, and adaptable robotic behavior. Whether explicitly acknowledged or not, the architectures underlying modern robotic autonomy rely on a decomposition of the control problem across levels with differing assumptions, objectives, and operating rates. Hierarchical control thus emerges not merely as a design choice, but as a structural necessity for managing the complexity, uncertainty, and computational demands inherent to modern robotics.

Efficiency

Complex robotic tasks involve reasoning across vast spatial, temporal, and dynamic scales. Attempting to solve the full problem is typically infeasible due to computational intractability. Layered architectures provide a natural means of divide-and-conquer: decomposing the global control problem into smaller, more tractable subproblems that can be solved locally within each level. By appropriately splitting responsibility, each level can operate on a manageable problem size tailored to its timescale and abstraction level, enabling efficient computation.

Feasibility

Efficiency alone is insufficient if plans generated at higher levels cannot be realized by lower levels. When layered architectures are well structured and share information bidirectionally — waypoints being passed down and certificates of performance going up — limitations at each level can be systematically accounted for in the design process. This makes it possible to guarantee feasibility of system-level behavior, despite the approximations and abstractions present at each individual level.

Generalizability

Perhaps most importantly, layered architectures enable generalizability — the ability to transfer components, solution methods, and guarantees across tasks, environ-

ments, and systems. Because each level is responsible for a distinct aspect of the overall control problem, modules can be reused or adapted independently. This modularity allows systems to adapt to new challenges without the need for complete redesign of solution strategies.

Philosophical Benefits

Adopting a layered architecture mindset also offers several intangible benefits. First, it helps engineers fight their own dogmatism — cleanly partitioning behaviors and objectives allows each component to be evaluated on its own merits, encouraging the adoption of methods that work rather than methods that conform to a favored ideology. Second, it serves as a design tool — if a module does not have the appropriate input/output structure to fit within the hierarchy, it indicates what kind of block needs to be developed to ensure cohesion. Finally, it enables the separation of responsibilities within an engineering team — people can work on independent problems while still making progress to a shared goal.

Limitations

Although there is elegance in splitting and coordination, the benefits of layered architectures come at a cost. Decomposing the problem into levels, each operating on assumptions and abstractions of those around it, inevitably introduces suboptimality when compared to globally designed solutions. This trade-off is central to layered architectures: what we lose in global optimality, we gain in modularity, scalability, and real-time feasibility. In many cases, this trade-off is not just acceptable, but necessary.

At the same time, infinite division and abstraction can lead to rigidity. Hierarchies can themselves become institutionalized into a dogma that resists paradigm shifts. For example, in the modern aerospace industry, legacy architectures often persist not because they are optimal, but because they are deeply entrenched and expensive to replace. Thus, while layered architectures are powerful, they must be approached with these limitations in mind.

1.4 Legged Robots as A Case Study for Layered Architectures

Legged robots provide a compelling case study to explore the benefits of layered architectures — from fast dynamic instabilities induced by underactuation to decision-making problems associated with path planning, there are a variety of temporal and spatial scales that must be resolved to achieve general autonomy. Furthermore, in

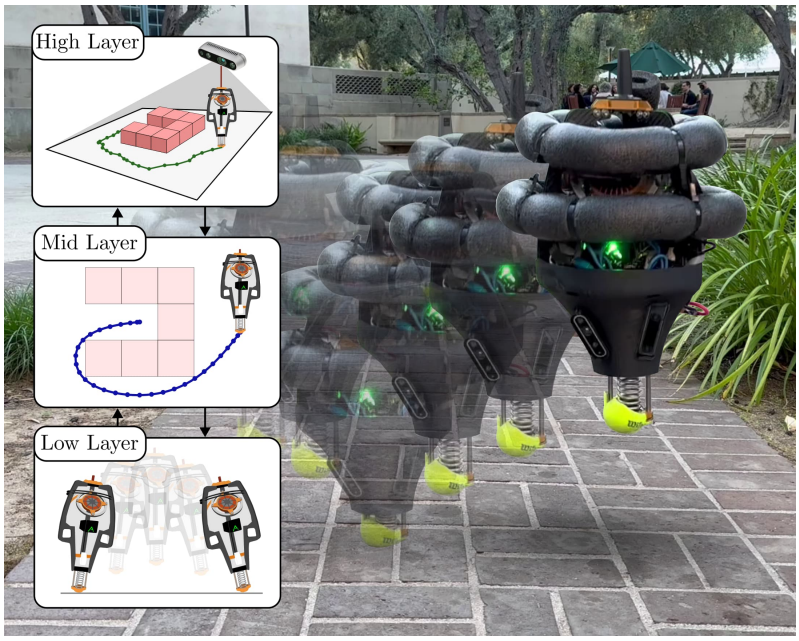


Figure 1.1: ARCHER, a dynamic 3D hopping robot, leveraging layered control architectures to navigate outdoor terrain.

the last few years the focus in legged systems has shifted from the synthesis of individual behaviors to the pursuit of full-scale autonomy; as such, this is an opportune time to refine a systems-level perspective in the context of legged robots.

Historically, enabling robots to operate in cluttered environments has been a focal challenge in control, dating back to the early days of mobile robots [18]. Although it is theoretically possible to accomplish the planning and control tasks in a single shot via policy-design [19]–[22], this approach poses significant challenges for legged robots due to high-dimensional state and action spaces, nonlinear dynamics, and a lack of safety and feasibility guarantees. As a result, it is more practical for legged robots to deploy layered control architectures, whereby high-level path planning algorithms are paired with low-level tracking controllers. Typically, kinematically feasible (i.e., collision-free) paths are passed down the control stack, and dynamic feasibility (i.e., satisfaction of the underlying system dynamics) is an assumed property of the low-level controller. Such planner-tracker paradigms are extremely common in legged robotic control [10], [23]. With this context established, we next examine how hierarchical principles manifest across the control stack for legged robots, beginning with low-level control.

Low Level

From the formal perspective, a rich catalog of methods has been developed for stabilizing nonlinear systems in the presence of unknown disturbances by utilizing underlying structural properties of the system [24]–[27]. In particular, the tools of Control Lyapunov Functions (CLFs) [28], [29] and Input-to-State Stability [30] have enabled the joint synthesis of stabilizing controllers and Lyapunov certificates of stability in the presence of disturbances, including through convex optimization [31]–[33]. However, the underactuated dynamics inherent to legged locomotion impose fundamental limits on controller performance and necessitate a critical understanding of the system's flow to achieve complex behaviors. Underactuation prevents arbitrarily shaping a system's dynamics, undermining the assumptions of many control-theoretic methods such as feedback linearization [34] and offline trajectory tracking.

For dynamic and underactuated legged systems, low-level feedback has had a long history. Early works on stabilizing legged robots focused on static stability like Zero-Moment Point (ZMP) [35], which ensures that the weight of the robot is always balanced over the feet. Although this is a straightforward way of ensuring that a robot will not fall over, it severely limits the range of dynamic behaviors that a robot can achieve.

The shift from quasi-static motion to dynamic motion was pioneered in the 1980s by the MIT Leg Lab [36]. This sparked a substantial body of work on reduced-order models — such as the linear inverted pendulum (LIP) [37], spring loaded inverted pendulum (SLIP) [38], and centroidal models, which remain widely used due to their analytical simplicity and physical interpretability. Beyond being tools for real-time control, these models also serve as components within layered control architectures [39].

In search of formal guarantees for these systems, [40] proposed a feedback strategy whereby an offline trajectory was paired with an exponentially stabilizing output controller to achieve provable stability on a planar biped. An algorithmic approach to finding this offline trajectory was explored in [41], and [42]–[44] incorporated a library of behaviors to get more expressive performance. These libraries of gaits were interpolated to produce a feedback controller in [45].

Planning-based control such as model predictive control (MPC) [46]–[48] has become a mainstream method for controlling legged systems [49]. MPC is particularly well suited for legged systems due to its ability to handle nonlinear dynamics, un-

deractuation, and state/input constraints. Although MPC has been successfully demonstrated in several challenging control settings [50]–[60], its application to legged robots faces significant challenges due to the high dimensionality and stiffness of the hybrid dynamics.

The difficulty in realizing MPC-based controllers at a fast enough rate to allow for real-time implementation is often resolved by using an approximate model of the system dynamics that is amenable to efficient planning, typically through linearization and temporal discretization of the continuous-time nonlinear system dynamics [48], [58]–[61]. Many MPC implementations rely on reduced-order models to retain computational tractability; see [62] for an overview. Using a full model instead was explored in [63] for a planar biped, and [64] for a 3D hopping robot. Boston Dynamics has long relied on predictive methods for stabilizing its high degree of freedom humanoids, and has recently moved from using reduced order models to full order models at the planning level [65]. To address the high computational cost of full-model optimization problems, some methods leverage a gradation of model fidelities along a time horizon [66], [67]. Other methods rely on offline trajectory optimization to generate desirable behaviors, and then track these behaviors online [40].

Reinforcement learning (RL) [68] takes the concept of offline computation even further, using concepts from stochastic optimal control and parallelized simulation environments to synthesize feedback controllers. RL methods have shown robust performance [69], [70] when the policy is trained in sufficiently randomized domains. Current methods in RL improve policies through simulator rollouts [71], typically at the expense of high data complexity. Although these can work well, they exhibit extreme sensitivity to cost function parameters and ignore the underlying system structure.

High Level

While low-level control ensures robustness and stabilization, high-level planning plays a crucial role in producing dynamically feasible trajectories to provide progress to a goal. Planners can ensure feasibility by adjusting the trajectories they generate based on a tracking certificate, i.e., a representation of what the tracking controller can reasonably accomplish. This bidirectional paradigm has theoretical roots in hierarchically consistent control [72], approximate simulation relations [73], and bisimulation [74]. This concept of levels communicating through achievable perfor-

mance metrics serves as the foundation for robust motion planning [75]. For linear systems, such tracking certificates can be synthesized directly [76]. For nonlinear systems like legged robots, generating tracking certificates is a more challenging task, and remains an active area of research. One option leverages Hamilton Jacobi reachability analysis to produce tracking upper bounds [77]. Depending on the existing system structure, notions of Input to State Stability [78] can also be used to constructively produce tracking certificates for nonlinear control systems [79]. Alternatively, the linearization of the nonlinear system can be used to get approximate polytopic reachable sets [80].

In practice, approximate methods for planning are used for their convenience and computational tractability. The use of such approximations creates a gap between the system which is being planned for and the actual evolution of the nonlinear system, requiring an additional measure of robustness to ensure constraint satisfaction. This robustness is often achieved by tightening the constraint sets by the maximum deviation between the approximate model and the continuous time nonlinear system dynamics [77], [81]–[87]. Approximating worse-case deviations is typically done using properties of the dynamics which may be difficult to compute, such as Lipschitz constants for which over-approximations yield conservativeness, or by solving computationally intensive optimization programs. More recently, hierarchical control frameworks have been proposed that plan with an approximate model, but directly address nonlinear dynamics with a low-level controller [88], [89].

Finally, to extend the notion of guaranteed feasibility to a decision making level, we require a certificate for the combined planner-tracker model, i.e., a representation of which states can be reached while satisfying state and input constraints. For discrete systems, this is often done by planning sequences of discrete actions based on motion primitives [90], [91]. Extending this to arbitrary continuous behaviors often requires solving two point boundary value problems [92], which can be computationally expensive.

The literature of high-level planning for legged systems has been comparatively sparse. This is attributable to the fact that robust and generalizable feedback controllers have only recently been designed for these systems. Constructing a pipeline which is able to guarantee feasibility necessitates reasoning about both the dynamics and free space information during the planning phase; algorithms which plan for both kinematic and dynamic feasibility are termed *kinodynamic planners* [93]. Kinodynamic planners can broadly be placed into two categories: sampling-based

and optimization-based methods. Both of these methodologies produce a graph, either of states or of sets, whereby the problem of path generation is reduced to a graph search.

In the sampling-based paradigm, one main way of producing kinodynamically feasible paths is by sampling policies from a discrete collection of predefined primitives [90], [91]. This method suffers from the fact that the predefined policies induce a bias and may not make meaningful progress towards the goal. Another approach is to randomly sample new points and connect them to the nearest node on the graph if a two-point boundary value problem is feasible [92], which can be viewed as an extension to the classic RRT [94] method of kinematic path planning. This method requires solving as many such problems as there are pairs of nodes in the graph, which can be expensive to compute for high-dimensional nonlinear systems. Sampling-based methods face two major challenges — path suboptimality due to the probability of sampling the true optimal path being zero, and bias, either through the discretized policy design, or Voronoi bias introduced by using a Euclidean metric [95]. To address suboptimality, various methods have proposed refinements to the path which increase the optimality [92]. To address the notion of bias, the notion of reachability has been introduced into sampling-based planners [96]. In particular, authors have developed reachability-guided RRT variants [17], [97], [98] that avoid solving boundary value problems during graph construction. For nonlinear systems, however, this top-down approach provides guarantees of feasibility which are often approximate and rely on using local linearizations of the nonlinearities; this requires the system to remain sufficiently close to the nominal trajectory during execution.

In contrast to building a graph of sampled states, optimization-based approaches instead try to directly generate optimal paths to the goal state. These methods require the solution of large mixed-integer programs to generate optimal paths to the goal, either via a keep-out philosophy [99], or a keep-in philosophy via convex decomposition [100]–[102]. To address the benefits and drawbacks of sampling-based and optimization techniques, various approaches have been proposed to combine these methods [103]–[105]. Despite these advancements, the computational burden associated with these approaches hinders their deployment in obstacle-rich environments in real time. As a means to mitigate these computational limitations, layered approaches to control synthesis have proven effective [12]. Before introducing the control strategies developed to address Problem 3.1, we must formally introduce the robotic platforms used throughout this work to provide appropriate context.

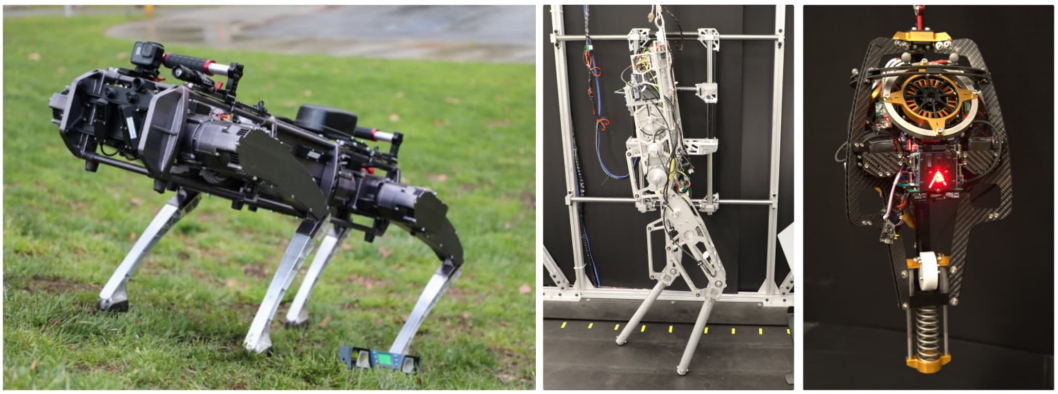


Figure 1.2: Robots discussed in this thesis, from left to right: Vision 60, AMBER, and ARCHER.

1.5 Legged Robot Introduction

The complexity of legged systems primarily arises from two sources. First, the floating base of the system is unactuated, requiring careful coordination of the systems actuated coordinates and natural dynamics for stabilization. Second, legged robots repeatedly make and break contact with the environment, introducing discrete transitions that must be handled explicitly as they play a central role in system stability. As such, this thesis leverages three robots that span various floating base representations as well as varying number of legs (and therefore distinct combinatorial contact combinations).

Vision 60

The first platforms is the Vision 60 v3.9 quadruped, manufactured by Ghost Robotics. This robot has 12 actuated joint degrees of freedom (three per leg), and 6 unactuated base frame degrees of freedom, resulting in a 36 dimensional state space. This robot weighs 44 kg, is 54 cm wide and stands roughly 50-60 cm tall. It is equipped with an Nvidia Jetson AGX Xavier for onboard compute, whose control loop runs at 1 kHz.

Quadrupeds like the Vision 60, although facing the challenge of having a large number degrees of freedom, benefit from having a large support polygon — the size of the convex hull of their stance feet. This leads to an inherent stability with quadrupeds that enables the use of relatively simple control methods to stabilize them.

AMBER

The second platform is AMBER-3M, a planar bipedal robot with modular legs and point feet [106]. AMBER has 4 actuated joints and 3 unactuated base degrees of freedom, giving it a 12 dimensional state. Actuation is provided through harmonic drives with 90-1 gear ratios attached to 4 bar linkages to drive the hips and knees. Between the viscous damping in the gearboxes and the backlash in the linkages, the model of AMBER is quite poor. AMBER stands 137 cm tall and weighs 22 kg. The system is controlled by an off-board i7-6700HQ CPU @ 2.6GHz with 16GB RAM. Desired torques are computed at 1kHz (unless otherwise noted) and communicated with the ELMO Gold Whistle motor drivers at 2kHz via EtherCat using the SOEM open source library.

Despite its lower dimensionality, AMBER is considerably harder to stabilize than Vision 60. The use of point feet results in an infinitesimal support polygon, which requires stabilization strategies that leverage the system's underlying dynamics. In some experiments, the legs were replaced with spring point feet, which introduced two additional unactuated degrees of freedom and made the robot model even worse.

ARCHER

The final platform is ARCHER, a 3D hopping robot [107]. ARCHER has 4 actuated degrees of freedom — three reaction wheels and one leg actuator — and 6 unactuated base degrees of freedom, leading to a 20-dimensional state. ARCHER has a carbon fiber frame with a structural aluminum spine. Three KV115 T-Motors with 250 g flywheel masses are attached for orientation control, and one U10-plus T-Motor is attached to a 3-1 planetary gearbox which connects to the foot via a cable and pulley system. The foot spring has a spring constant of 11,700 N/m, selected to be as small as possible while not bottoming out during hopping, which the authors note empirically maximizes hop height. The robot actuators are powered by two 6 cell LiPo batteries connected in series, which supply 50.8 V at up to 100 A of current to the four ELMO Gold Solo Twitter motor controllers. Two compute configurations were used: an external AMD 5950x CPU with an Nvidia 4090 GPU (for indoor trials), and a 16 GB RAM Nvidia Orin powered by two separate 6 cell LiPo batteries connected in parallel (for outdoor trials). The computers were connected over Ethernet to two Arduino Teensys which communicate with the motor controllers. Two estimation stacks were also used: optitrack for state estimates and an overhead ZED camera for color images (for indoors trials) and two on-board cameras, an Intel Realsense T265 for state estimates, and an Intel Realsense D435 for depth

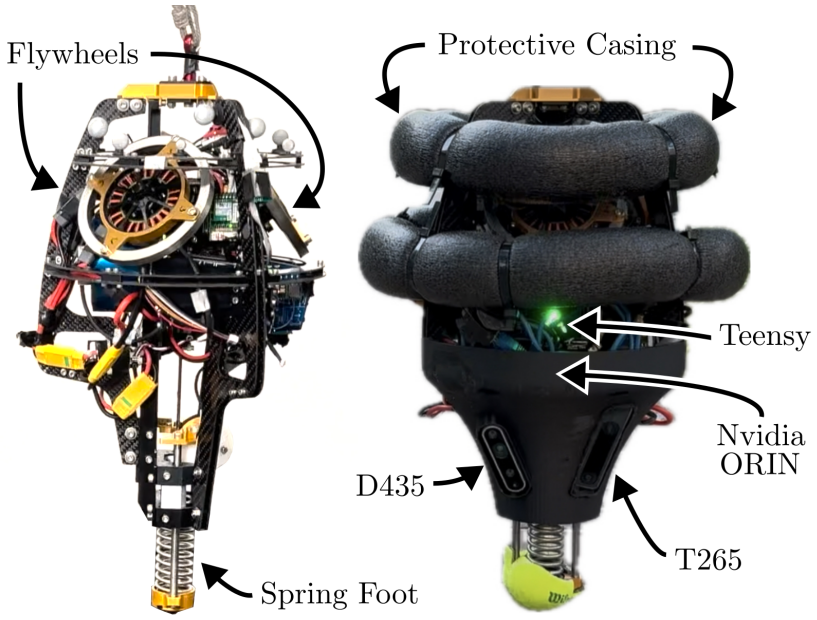


Figure 1.3: ARCHER, the 3D hopping robot used in this work with power, actuators, compute, and sensors annotated.

and color images (for outdoor trials). For outdoor trials, the robot was wrapped in pool noodles, the foot protected by a tennis ball, and the compute stack housed in a protective 3D printed shell. Fully equipped, ARCHER weighs 8.25 kg and stands 60 cm tall.

ARCHER is highly underactuated — as it flies through the air its center of mass states follow ballistic trajectories with no direct control over its translational motion. On the other hand, the orientation of this robot is directly controllable. Therefore, the question of interest becomes: how can we leverage the angle through impact to stabilize the full system state? These features make ARCHER the perfect platform to study hybrid underactuated control.

Chapter 2: A Gentle Mathematical Introduction

Contents

Dynamics asks the question: what will happen?
Control asks the question: how do we make it happen?

Mathematics plays a pivotal role within the field of engineering, providing a language through which we can model and describe the world around us. This section is devoted to developing the relevant mathematical preliminaries required to understand robot autonomy. Along the way, there will be a number of modeling choices, which we will make explicit as they arise.

We begin by choosing a *state* to describe our system. This already presents a modeling choice, as the state must encode all relevant features of the system's behavior. In general, any physical system contains infinite complexity — a single blade of grass, for instance, contains billions of atoms, all of which could, in theory, be modeled. Limits on computation, numerical resolution, and patience, however, force us to characterize only the predominant features.

2.1 Nonlinear Dynamics

For the time being, we model a state as a vector $\mathbf{x} \in \mathbb{R}^n$, an element of an $n-$dimensional real vector space. A powerful method of describing how systems evolve over time is through the lens of *differential equations*, which specify that the rate of change of the state is a function of the current state. To make this precise, consider the following ordinary differential equation (ODE):

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}) \tag{2.1}$$

where $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a function that describes this relationship. The function $\mathbf{f}$ describes a *vector field* on the state space — it maps states $\mathbf{x}$ to vectors that lie tangent to the evolution of the system over time. To avoid any mathematical pathologies, we assume that the function $\mathbf{f}$ is continuously differentiable, i.e., that its Jacobian $\frac{d\mathbf{f}}{d\mathbf{x}}$ exists and is continuous, unless otherwise stated.

The choice of model in (2.1) already carries certain assumptions. First, we have taken the dynamics to be *autonomous* — the dynamics have no explicit dependence on time, meaning that the behavior of the system at any given state is independent of when it is evaluated. Second, we have assumed that no spatial gradients appear — the dynamics only contain the state and its time derivative, which helps us avoid the added complexities of partial differential equations. What we lose in modeling ability, we gain in system structure — the class of ODEs is extremely descriptive, but simple enough to reason about. Given the dynamics (2.1), we can write the time evolution of the system as:

$$\mathbf{x}(t) = \boldsymbol{\varphi}_t(\mathbf{x}_0) \triangleq \mathbf{x}_0 + \int_0^t \mathbf{f}(\mathbf{x}(\tau))d\tau, \tag{2.2}$$

where $\mathbf{x}_0 \in \mathbb{R}^n$ is the initial state and $\boldsymbol{\varphi} : \mathbb{R}_{\geq 0} \times \mathbb{R}^n \to \mathbb{R}^n$ is the *flow*. We assume that this solution is forward complete — it exists and is well defined for all time.

Analysis

One of the central objectives of dynamics and control is to analyze the behavior of the system flow over time. The first natural question that arises is:

> **Question:** *Does the flow of the system $\boldsymbol{\varphi}_t(\mathbf{x}_0)$ converge to any particular point in space as $t \to \infty$?*

That is, given an equilibrium point $\mathbf{x}^*$, a point where $\mathbf{f}(\mathbf{x}^*) = \mathbf{0}$, we would like to know whether any of the following properties hold:

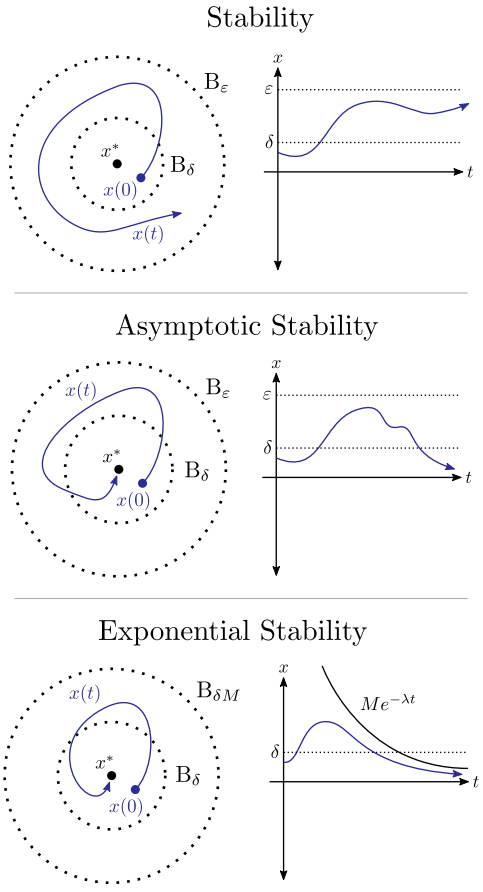


Figure 2.1: Various notions of stability.

Definition 2.1. A system is said to be *stable* if, for every $\varepsilon > 0$, there exists a $\delta > 0$ such that:

$$\|\mathbf{x}(0) - \mathbf{x}^*\| \leq \delta \implies \|\mathbf{x}(t) - \mathbf{x}^*\| \leq \varepsilon,$$

for all $t \geq 0$.

Definition 2.2. A system is said to be *asymptotically stable* if it is stable and additionally satisfies:

$$\lim_{t \to \infty} \|\mathbf{x}(t) - \mathbf{x}^*\| = \mathbf{0}.$$

Definition 2.3. A system is said to be *exponentially stable* there exist constants $M, \lambda > 0$ such that:

$$\|\mathbf{x}(t) - \mathbf{x}^*\| \leq Me^{-\lambda t}\|\mathbf{x}(0) - \mathbf{x}^*\|,$$

for all $t \geq 0$.

One's first instinct might be to attempt to solve explicitly for $\mathbf{x}(t)$ and then study its behavior as $t \to \infty$. In some cases, this is tractable; for example, if the dynamics

are linear, that is $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ for some matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, then the solution can be written as:

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0 \triangleq \left(\mathbf{I} + \mathbf{A}t + \frac{1}{2!}\mathbf{A}^2t^2 + \frac{1}{3!}\mathbf{A}^3t^3 + \dots\right)\mathbf{x}_0 \tag{2.3}$$

where $e^{\mathbf{A}t}$ is the *matrix exponential*, defined via the above infinite series. To analyze the stability of this flow, we can simply look at an eigenvalue decomposition of the matrix $\mathbf{A}$ — if the real part of all eigenvalues lie in the left half plane, then the solution will be upper bounded by a decaying exponential [108].

Unfortunately, for most nonlinear systems, explicit closed-form solutions are not available. To make this concrete, we introduce the nonlinear oscillator. This system has the same dynamics as the pendulum under a different topology. Once we introduce the notion of manifold we will present the pendulum, which will serve as a running illustration throughout the background section.

Example: The Nonlinear Oscillator

Consider an oscillator with position $\theta \in \mathbb{R}$ and velocity $\dot{\boldsymbol{\theta}} \in \mathbb{R}$. The dynamics of the system are given by:

$$\frac{d}{dt}\begin{bmatrix}\theta \\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\dot{\theta} \\ -\sin(\theta)\end{bmatrix}.$$

Even for this extremely simple nonlinear system, we cannot write down an analytic expression for the evolution $\theta(t)$ over time. Given this immediate hurdle:

> **Question:** *If we cannot even produce solution curves for simple systems, what hope do we have for analyzing the behavior of general nonlinear systems?*

Linearization

The first method for analyzing nonlinear systems is to approximate them up to first order via *linearization*, whereby we replace the nonlinear system near an equilibrium point with a linear system. Specifically, consider the system linearized around $\mathbf{x}^*$ given by:

$$\dot{\mathbf{x}} = \underbrace{\frac{d\mathbf{f}}{d\mathbf{x}}(\mathbf{x}^*)}_{\triangleq \mathbf{A}}\mathbf{x}$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the Jacobian of $\mathbf{f}$ evaluated at $\mathbf{x}^*$. Note that this linearized model is only valid in a neighborhood of the equilibrium — far away this approximation significantly degrades, as the nonlinearities can be the dominant forces present. Given that we now have a linear system, we can leverage the linear system solution (2.3). Additionally, the Hartman-Grobman Theorem [109] guarantees that if the linearized system is exponential stability, then there exists a neighborhood around the origin where the nonlinear system is also exponentially stable. However, the domain of validity of the analysis is limited to the (potentially infinitesimal) region where the linear dynamics dominate the nonlinearities.

> **Question:** *Is there any hope for a broader theory that extends beyond linearizations?*

Lyapunov Theory

To move beyond the limitations of linearization, we turn to a broader and more powerful framework: Lyapunov theory. Rather than reasoning directly about system trajectories, Lyapunov theory studies systems indirectly, through the projection of a systems' dynamics onto a scalar-valued function. The main idea is powerfully simple: if we can find a function that always decreases along trajectories of a dynamical system, we can infer properties about the system's evolution without solving for $\mathbf{x}(t)$ explicitly. With this intuition in mind, we state Lyapunov's theorem, adapted from its original 1892 version to match our nomenclature:

Theorem 2.4 (Lyapunov)**.** *Consider the equilibrium point $\mathbf{x}^* = \mathbf{0}$ and let $V : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ be a continuously differentiable, positive definite function. If we have that:*

$$\dot{V}(\mathbf{x}) \leq 0,$$

then the equilibrium $\mathbf{x}^* = \mathbf{0}$ is stable. If moreover we have that:

$$\dot{V}(\mathbf{x}) \prec 0,$$

then the equilibrium $\mathbf{x}^* = \mathbf{0}$ is asymptotically stable.

To build intuition, consider the visual interpretation of Lyapunov's Theorem in Figure 2.2. The left plot shows the system dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ as a vector field. The middle plot shows another vector field, this time defined by the gradient of the candidate Lyapunov function V. The right plot illustrates that $\dot{V} = \frac{dV}{d\mathbf{x}}\mathbf{f}(\mathbf{x})$ can be thought of as the inner product of these two vector fields, measuring how much they

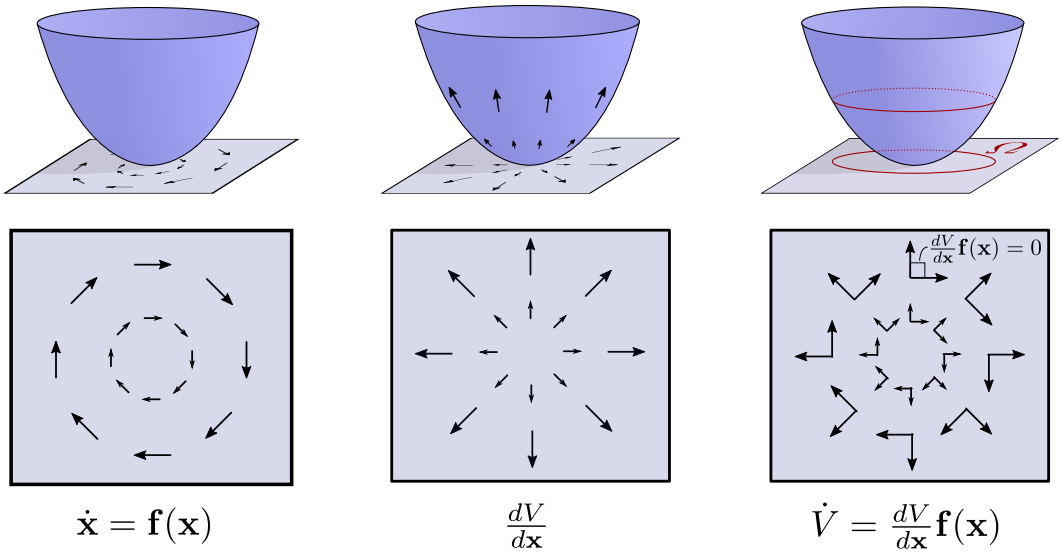


Figure 2.2: Lyapunov Theory.

align. If the angle between ∇V and $\mathbf{f}$ is "obtuse" at all points, then $\dot{V} \prec 0$. This means that the flow of the system is guaranteed to descend level sets of the function V, implying that the solutions are bounded and therefore stable.

Remark: *The power of Lyapunov's method lies in its pointwise nature: the condition can be verified independently at each point in the state space. Therefore, conclusions of the system's behavior can be made without ever needing to reason about the solution curves of the system.*

Note that Lyapunov's theorem was not only applicable to linear systems — any system that satisfies the Lyapunov condition is certifiably stable. The classic formulation of Lyapunov's theorem is powerful, but it is useful to restate it in a way that is more conducive to later analysis. Before proceeding, we require a few function classes to be defined:

Definition 2.5 (Class-$\mathcal{K}$)**.** A continuous function $\alpha : [0, a) \to \mathbb{R}_{\geq 0}$, with $a > 0$, is said to belong to *class* $\mathcal{K}$ denoted $\alpha \in \mathcal{K}$ if $\alpha(0) = 0$ and α is strictly monotonically increasing. If $a = \infty$ and $\lim_{r\to\infty} \alpha(r) = \infty$, then α is *class* $\mathcal{K}_\infty$ ($\alpha \in \mathcal{K}_\infty$).

Using these constructions, we can restate Lyapunov's theorem as:

Theorem 2.6 (Modern Lyapunov [26])**.** *Consider the equilibrium point $\mathbf{x}^* \in \mathbb{R}^n$ and the continuously differentiable function $V : \mathbb{R}^n \to \mathbb{R}_+$. If V satisfies:*

$$\alpha_1(\|\mathbf{x}\|) \leq V(\mathbf{x}) \leq \alpha_2(\|\mathbf{x}\|)$$
$$\dot{V}(\mathbf{x}) = \frac{dV}{d\mathbf{x}}(\mathbf{x})\mathbf{f}(\mathbf{x}) \leq -\alpha_3(\|\mathbf{x}\|)$$

for $\alpha_i \in \mathcal{K}$, then the system is asymptotically stable. If α_i is of the form $\|\mathbf{x}\|^c$ for some constant $c > 0$, then the system is exponentially stable.

Let us return briefly to our linear system $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$. Consider a positive definite symmetric matrix $\mathbf{P} \in \mathbb{S}^n$ and define a candidate Lyapunov function as the positive definite function $V(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}\mathbf{x}$. Taking the time derivative of V yields:

$$\dot{V}(\mathbf{x}) = \frac{dV}{d\mathbf{x}}(\mathbf{x})\dot{\mathbf{x}} = 2\mathbf{x}^\top \mathbf{A}^\top \mathbf{P}\mathbf{x} = \mathbf{x}^\top(\mathbf{A}^\top \mathbf{P} + \mathbf{P}\mathbf{A})\mathbf{x},$$

where the last rearrangement puts the product in a symmetric form. Therefore, if we can find a matrix $\mathbf{P}$ satisfying the condition:

$$\mathbf{A}^\top \mathbf{P} + \mathbf{P}\mathbf{A} = -\mathbf{Q} \tag{2.4}$$

for some positive definite matrix $\mathbf{Q}$, then $\dot{V} = -\mathbf{x}^\top \mathbf{Q}\mathbf{x}$. As this is a negative definite function, the requirements of Lyapunov's theorem are satisfied and therefore the function V certifies that the equilibrium point $\mathbf{x}^*$ is exponentially stable. The condition (2.4) has the fitting name of the continuous time Lyapunov equation (CLTE). When $\mathbf{A}$ is Hurwitz (all of its eigenvalues have strictly negative real parts), a unique positive definite matrix $\mathbf{P}$ exists for every positive definite choice of $\mathbf{Q}$.

We also state a powerful converse result:

Theorem 2.7 (Converse Lyapunov Theorem [29])**.** *If a system is exponentially stable, then there exists a function $V : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ satisfying:*

$$k_1\|\mathbf{x}\|^2 \leq V(\mathbf{x}) \leq k_2\|\mathbf{x}\|^2$$
$$\dot{V}(\mathbf{x}) = \frac{dV}{d\mathbf{x}}(\mathbf{x})\mathbf{f}(\mathbf{x}) \leq -k_3\|\mathbf{x}\|^2$$

for $k_i > 0$.

Thus, Lyapunov functions serve as *certificates of performance* — if the conditions in Lyapunov's theorem are satisfied, then the function V certifies that the system has a particular stability type. Certificates of performance such as these will play a central role in the theory of layered architectures.

Input to State Stability

Given the constructions in the previous section, a natural question arises:

> **Question:** *If disturbances are present, what can we say about the stability of the system?*

To address this question, consider the dynamics with disturbances added as:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{d}(t)$$

for $\mathbf{d} : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ a time varying disturbance. Let $\|\mathbf{d}\|_\infty \triangleq \sup_{t \geq 0} \|\mathbf{d}(t)\|_2$ of a time varying signal denote the supremum of its two norm over all time. Then, we have the following definition:

Definition 2.8. The system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{d}$ with $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is said to be *Exponentially Input to State Stable* (E-ISS) with respect to the equilibrium pair $(\mathbf{x}^*, \mathbf{d}^*) = (\mathbf{0}, \mathbf{0})$ if there exist constants $M, \lambda > 0$, $a \in \mathbb{R}_{>0} \cup \{+\infty\}$, and a function $\gamma \in \mathcal{K}$ such that:

$$\|\mathbf{x}(0)\| \leq a \implies \|\mathbf{x}(t)\| \leq Me^{-\lambda t}\|\mathbf{x}(0)\| + \gamma(\|\mathbf{d}\|_\infty).$$

The notion of input to state stability (ISS) generalizes to broader stability types, but in this thesis, we will only focus on the exponential case.

Just as Lyapunov theory provided a certificate of stability for the system without disturbances, it can also provide a certificate for input to state stability:

Definition 2.9. Consider the system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{d})$ where $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$. A continuously differentiable function $V : \mathcal{X} \to \mathbb{R}_{\geq 0}$ is an *E-ISS Lyapunov function* with respect to the equilibrium pair $(\mathbf{x}^*, \mathbf{d}^*) = (\mathbf{0}, \mathbf{0})$ if:

$$k_1\|\mathbf{x}\|^c \leq V(\mathbf{x}) \leq k_2\|\mathbf{x}\|^c$$
$$\|\mathbf{x}\| \geq \rho(\|\mathbf{d}\|_\infty) \implies \dot{V}(\mathbf{x}, \mathbf{d}) \leq -k_3\|\mathbf{x}\|^c,$$

for $k_i, c > 0$ and $\rho \in \mathcal{K}$.

If an E-ISS Lyapunov function exists, then the system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{d})$ is E-ISS. We can alternatively write the E-ISS condition as:

$$\dot{V}(\mathbf{x}, \mathbf{d}) \leq -k_3\|\mathbf{x}\|^c + \sigma(\|\mathbf{d}\|_\infty),$$

for $\sigma \in \mathcal{K}$, which will be convenient for later analysis.

Safety

Besides stability, another key property we may wish our dynamical systems to exhibit is *safety*. In the context of this thesis, we will take safety to mean that the system remains within a predefined *safe set* at all times. Consider a set $\mathcal{C} \subset \mathbb{R}^n$ defined as the 0-superlevel set of a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$, yielding:

$$\mathcal{C} \triangleq \{\mathbf{x} \in \mathbb{R}^n : h(\mathbf{x}) \geq 0\}. \tag{2.5}$$

We denote the boundary of $\mathcal{C}$ as $\partial\mathcal{C} \triangleq \{\mathbf{x} \in \mathbb{R}^n : h(\mathbf{x}) = 0\}$ and its interior as $\text{Int}(\mathcal{C}) \triangleq \{\mathbf{x} \in \mathbb{R}^n : h(\mathbf{x}) > 0\}$. We assume that $\mathcal{C}$ is nonempty and has no isolated points, that is, $\text{Int}(\mathcal{C}) \neq \emptyset$ and $\overline{\text{Int}(\mathcal{C})} = \mathcal{C}$. We will refer to $\mathcal{C}$ as the *safe set*. This construction motivates the following definitions of forward invariance and safety:

Definition 2.10 (*Forward Invariance & Safety*)**.** A set $\mathcal{C} \subset \mathbb{R}^n$ is *forward invariant* if for every $\mathbf{x}_0 \in \mathcal{C}$, the solution $\mathbf{x}(t)$ to (2.1) satisfies $\mathbf{x}(t) \in \mathcal{C}$ for all $t > 0$. The system (2.1) is *safe* on the set $\mathcal{C}$ if the set $\mathcal{C}$ is forward invariant.

Once again, we would like to provide a certificate — this time, that a system is safe — without reasoning about the solution curves of that system. As with stability theory, there is a seminal result from the 1940's from Nagumo that relates safety to a derivative condition:

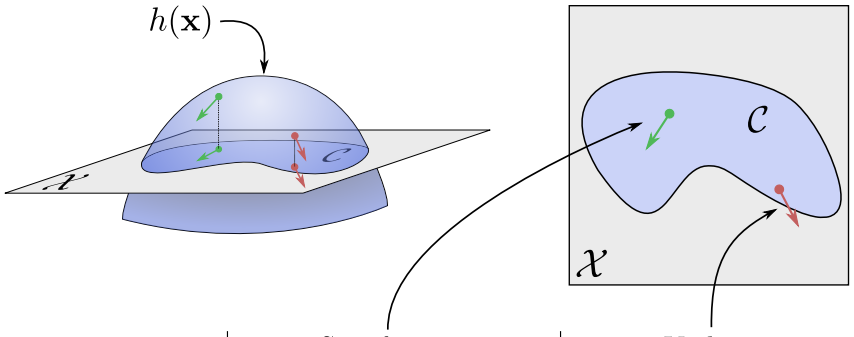
Theorem 2.11 ([110])**.** *Let $\mathcal{C} \subset \mathbb{R}^n$ be the 0-superlevel set of a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$. We have that:*

$$\mathcal{S} \textit{ is forward invariant} \iff \dot{\mathbf{h}} \geq 0 \textit{ for all } \mathbf{x} \in \partial\mathcal{C}.$$

Nagumo's theorem provides another powerfully simple condition: to certify that a set is forward invariant, it suffices to verify that on its boundary, the dynamics are flowing into the set.

In many settings, particularly involving controller synthesis, it is advantageous to formulate conditions that hold over the entire state space. To achieve this, we introduce a family of functions similar to those used in the stability analysis:

Definition 2.12 (Extended Class-$\mathcal{K}$)**.** A continuous function $\alpha : (-b, a) \to \mathbb{R}$, with $a, b > 0$, is said to belong to *extended class* $\mathcal{K}$ ($\alpha \in \mathcal{K}_e$) if $\alpha(0) = 0$ and α is strictly monotonically increasing. If $a, b = \infty$, $\lim_{r\to\infty} \alpha(r) = \infty$, and $\lim_{r\to-\infty} \alpha(r) = -\infty$, then α is said to belong to *extended class* $\mathcal{K}_\infty$ ($\alpha \in \mathcal{K}_{\infty,e}$).



	Satisfaction	Violation
Continuous	$\dot{h}(\mathbf{x}) \geq -\alpha(h(\mathbf{x}))$	$\dot{h}(\mathbf{x}) < -\alpha(h(\mathbf{x}))$
Discrete	$\Delta h(\mathbf{x}_k, \mathbf{u}_k) \geq -\gamma h(\mathbf{x}_k)$	$\Delta h(\mathbf{x}_k, \mathbf{u}_k) < -\gamma h(\mathbf{x}_k)$

Figure 2.3: Continuous and discrete barrier function conditions.

With this, we are now equipped to introduce the notion of barrier functions:

Definition 2.13 (*Barrier Function* [111])**.** Let $\mathcal{C} \subset \mathbb{R}^n$ be the 0-superlevel set of a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$. The function h is a *barrier function* for (2.1) on $\mathcal{C}$ if there exists $\alpha \in \mathcal{K}_{\infty,e}$ such that for all $\mathbf{x} \in \mathbb{R}^n$:

$$\dot{h}(\mathbf{x}) = \frac{dh}{d\mathbf{x}}\mathbf{f}(\mathbf{x}) \geq -\alpha(h(\mathbf{x})), \tag{2.6}$$

We now have the tools required to introduce a certificate of safety:

Theorem 2.14 ([112])**.** *Given a set $\mathcal{C} \subset \mathbb{R}^n$ defined as the 0-superlevel set of a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$, if h is a barrier function for* (2.1) *on $\mathcal{C}$, then the system* (2.1) *is safe with respect to the set $\mathcal{C}$.*

Thus, barrier functions provide certificates of safety, offering yet another means by which levels in layered architectures can communicate.

Discrete-Time Systems

Thus far, we have considered systems that evolve continuously over time. Next, we extend our discussion to systems the experience discrete state updates, and develop analogous notions of stability and safety. In discrete time, conditions on the differential of the state are replaced by conditions on the increment of the state across time steps. To formalize this, let $\mathbf{x}_k \in \mathbb{R}^n$ denote the state of the system at the index $k \in \mathbb{Z}_+$. The system dynamics are given by the update equation:

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k) \tag{2.7}$$

where $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ is a continuously differentiable function.

Similar to the continuous-time case, we can define a notion of exponential stability:

Definition 2.15 (Discrete Exponential Stability)**.** The discrete system (2.7) is *exponentially stable* to an equilibrium point $\mathbf{x}^* \in \mathbb{R}^n$ if there exists constants $M > 0$ and $\beta \in [0, 1)$ such that:

$$\|\mathbf{x}_k - \mathbf{x}^*\| \leq M\beta^k \|\mathbf{x}_0 - \mathbf{x}^*\|.$$

for the initial condition $\mathbf{x}_0$.

Other stability types also exist for discrete systems, but in this thesis we will only consider the exponential case.

The stability of discrete-time systems can be certified through a discrete analog of Lyapunov theory:

Definition 2.16. A positive definite function $V : \mathcal{E} \to \mathbb{R}$ is called a *discrete exponential Lyapunov function* if there exists constants $\alpha \in (0, 1],\ k_1, k_2 > 0$ such that:

$$k_1\|\mathbf{x}_k\|^2 \leq V(\mathbf{x}_k) \leq k_2\|\mathbf{x}_k\|^2$$
$$\Delta V(\mathbf{x}) = V(\mathbf{x}_{k+1}) - V(\mathbf{x}_k) \leq -\alpha V(\mathbf{x}_k).$$

Remark: *Because discrete-time systems directly map between states without evolving along continuous trajectories, the analysis and design of controllers becomes substantially more challenging. As a consequence, reasoning about constraints and guaranteeing properties such as stability or safety can often be more challenging in discrete time than continuous time.*

Similarly, we can introduce a notion of safety to discrete-time systems by introducing discrete-time barrier functions:

Definition 2.17 (Discrete Time Barrier Functions)**.** Let $\mathcal{C} \subset \mathbb{R}^n$ be the 0-superlevel set of a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$. The function h is a *discrete time barrier function* for (2.7) on $\mathcal{C}$ if there exists $\alpha \in \mathcal{K}_{\infty,e}$ such that for all $\mathbf{x}_k \in \mathbb{R}^n$:

$$\Delta h(\mathbf{x}_k, \mathbf{x}_{k+1}) = h(\mathbf{x}_{k+1}) - h(\mathbf{x}_k) \geq -\alpha(h(\mathbf{x}_k))$$

for an extended class $\mathcal{K}_\infty$ function α satisfying $\alpha(r) < r$.

Theorem 2.18. *Given a set $\mathcal{C} \subset \mathbb{R}^n$ defined as the 0-superlevel set of a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$, if h is a barrier function for* (2.7) *on $\mathcal{C}$, then the system is safe with respect to the set $\mathcal{C}$.*

As demonstrated in this section, we now have a complete language for describing stability and safety certificates for both continuous and discrete dynamical systems.

Manifolds

Thus far, we have treated the system state $\mathbf{x}$ as an element of the vector space $\mathbb{R}^n$, relying on the fact that $\mathbb{R}^n$ is a Euclidean space where the tangent space at any point is canonically identified with the space itself. As we move towards more sophisticated modeling — especially of rotations and orientations — we must generalize this perspective: the state may instead live on a manifold, whereby the dynamics must be understood in terms of the manifold's tangent spaces.

Consider a differentiable function $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^p$ with $\mathbf{0}$ a regular value — that is, $\mathbf{h}(\mathbf{x}) = \mathbf{0}$ implies that the Jacobian $\frac{\partial \mathbf{h}}{\partial \mathbf{x}}$ is full rank. Under these conditions, we have that $\mathcal{M} \triangleq \{\mathbf{x} \mid \mathbf{h}(\mathbf{x}) = \mathbf{0}\}$ defines a $n - p$-dimensional embedded submanifold of $\mathbb{R}^n$ [113]. Associated with each point $x \in \mathcal{M}$ is the tangent space $\mathsf{T}_x\mathcal{M}$, consisting of all vectors that lie *tangent* to the manifold at that point. A vector $\mathbf{v} \in \mathbb{R}^n$ is a *tangent vector* at x, denoted as $\mathbf{v} \in \mathsf{T}_x\mathcal{M}$, if and only if:

$$\frac{\partial \mathbf{h}}{\partial \mathbf{x}}^\top \mathbf{v} = \mathbf{0}.$$

This aligns with the classical notions of tangent vectors, as the gradient field of the function $\mathbf{h}$ forms a basis for the annihilators of the tangent space.

Dynamics on Manifolds

For a system whose state x evolves on a differentiable manifold $\mathcal{X}$, the dynamics take the form:

$$\dot{x} = f(x)$$

where $f : \mathcal{X} \to \mathsf{T}_x\mathcal{X}$. Given an initial point $x_0 \in \mathcal{X}$, the flow of the system generates a curve $\varphi : \mathcal{X} \times \mathbb{R}_{\geq 0} \to \mathcal{X}$ satisfying:

$$\begin{aligned} \frac{d}{dt}\varphi_t(x_0) &= f(\varphi_t(x_0)), \\ \varphi_0(x_0) &= x_0. \end{aligned}$$

Associated with the flow is the *pushforward* operation, which transports tangent vectors along the flow. The pushforward $(\varphi_t)_* : \mathsf{T}_{x_0}\mathcal{X} \to \mathsf{T}_{\varphi_t(x_0)}\mathcal{X}$ describes how perturbation vectors at time zero evolve under the dynamics to perturbation vectors at time t, and will be useful when comparing the dynamics equations at different points along the flow, as is often done during analysis.

Given a manifold $\mathcal{M}$ and dynamics f, a key property in our analysis will be the notion of controlled invariance:

Definition 2.19. A manifold $\mathcal{M}$ is *controlled invariant* under the dynamics f if for all $x \in \mathcal{M}$ we have that:

$$f(x) \in \mathsf{T}_{\mathbf{x}}\mathcal{M}.$$

That is, the vector field associated with the dynamics lies in the tangent space of the manifold. This condition ensures that the flow of the system remains on the manifold $\mathcal{M}$.

Lie Groups

Lie groups play a central role in modeling the configuration spaces of robotic systems, particularly in describing rotational motion. Before introducing these, we briefly recall the notion of a group:

Definition 2.20. A *group* is a nonempty set G equipped with a binary operation $\cdot : G \times G \to G$ satisfying:

- For all $g_1, g_2, g_3 \in G$, we have that $(g_1 \cdot g_2) \cdot g_3 = g_1 \cdot (g_2 \cdot g_3)$
- There exists an element e in G such that for every $g \in G$, we have $e \cdot g = g$ and $g \cdot e = g$
- For each $g_1 \in G$, there exists an inverse element $g_1^{-1} \in G$ such that $g_1 \circ g_1^{-1} = e$ and $g_1^{-1} \circ g_1 = e$.

A *Lie group* is a group that is also a smooth manifold, such that both multiplication and inversion are smooth maps. As with general smooth manifolds, we can also consider the tangent space at each point in a Lie Group. Of particular importance is the tangent space at the identity element, denoted $\mathsf{T}_e G$, called the *Lie algebra* and often written as $\mathfrak{g}$. The Lie algebra provides a canonical vector space associated with the Lie group, enabling a convenient location to perform linear analysis. Moreover,

the exponential map $\exp : \mathfrak{g} \to G$ and its inverse, the logarithmic map $\log : G \to \mathfrak{g}$ provide a local diffeomorphism between a neighborhood of the identity in G and elements of the Lie algebra $\mathfrak{g}$, allowing local behavior on the group to be studied entirely within a linear vector space.

Example: Construction of $SO(n)$

One of the most important Lie Groups for robotics is the special orthogonal group $SO(3)$, which represent 3D rotations. To construct $SO(n)$, consider the general linear group:

$$GL(n) = \{\mathbf{A} \in \mathbb{R}^{n \times n} \mid det(\mathbf{A}) \neq 0\},$$

which is the set of invertible $n \times n$ matrices. Next define the smooth map $\mathbf{f} : GL(n) \to \mathbb{R}^{n \times n}$ by:

$$f(\mathbf{A}) = \mathbf{A}^\top \mathbf{A}.$$

The orthogonal group $O(n)$ is the pre-image of the identity element $\mathbf{I}_n$ under the mapping f:

$$O(n) = \{\mathbf{A} \in \mathbb{R}^{n \times n} \mid \mathbf{A} \in GL(n), \mathbf{A}^\top \mathbf{A} = \mathbf{I}_n\}.$$

Finally, by restricting to matrices of determinant 1, we obtain the special orthogonal group:

$$SO(n) = \{\mathbf{A} \in \mathbb{R}^{n \times n} \mid \mathbf{A} \in GL(n), \mathbf{A}^\top \mathbf{A} = \mathbf{I}_n, det(\mathbf{A}) = 1\}.$$

From the inverse function theorem, we know that the pre-image of any regular value of f defines a submanifold of the space. As f and the determinant operation are both smooth in the space of matrices, we know that $SO(n)$ is a smooth submanifold of $GL(n)$.

With the goal of doing calculus on manifolds, we must be especially careful to understand what space the tangent vectors live in. Specifically, we will consider perturbations of tangent vectors, but in doing so we have to ensure that the constructions that we make have a mathematical consistency, i.e., perturbations of our vectors do not exit the domain of definition of the objects we create. As such, we will consider perturbations "vectors" to matrices in $SO(n)$ as elements of the embedded space $GL(n)$.

Next, consider the tangent space to $SO(n)$ at the "point" $\mathbf{A}$. Differentiating the constraint $\mathbf{A}^\top\mathbf{A} = \mathbf{I}_n$ in the direction of a variation (or tangent "vector") $\mathbf{V}$ gives:

$$[Df(\mathbf{A})](\mathbf{V}) = \lim_{\varepsilon\to 0} \frac{(\mathbf{A}+\varepsilon\mathbf{V})^\top(\mathbf{A}+\varepsilon\mathbf{V}) - \mathbf{A}^\top\mathbf{A}}{\varepsilon}.$$

By considering this expansion up to first order, the tangent vector $\mathbf{V}$ at $\mathbf{A}$ must satisfy:

$$\mathbf{A}^\top\mathbf{V} + \mathbf{V}^\top\mathbf{A} = \mathbf{0}_{n\times n}$$

as these are the directions in the null space of the differential of f.

If we want to associate a Lie Algebra to the Lie Group $SO(n)$, then we need to consider the tangent space at identity. At this point, this condition reduces to:

$$[Df(\mathbf{I}_n)](\mathbf{V}) = 0 \iff \mathbf{V} + \mathbf{V}^\top = 0_{n\times n},$$

i.e., $\mathbf{V}$ must be a skew symmetric matrices. The space of skew-symmetric matrices forms a vector space. Equipped with the binary operation of the standard Lie bracket (the matrix commutator) and noting that skew symmetric matrices are closed under this operator (as skew symmetric matrices commute), this defined the Lie algebra $\mathfrak{so}(n)$.

Example: Quaternions

Another useful manifold to consider is the three sphere, defined as $\mathcal{S}^3 \triangleq \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x}\| = \langle \mathbf{x}, \mathbf{x}\rangle = 1\}$. To imbue this set with group structure, we can parameterize it via unit quaternions, i.e. $\mathcal{S}^3 = \{q \in \mathbb{H} \mid \|q\| = qq^* = 1\}$ where q^* denotes the conjugate of q. Quaternion multiplication gives a smooth group operation, and conjugation gives a smooth inverse, ensuring that $\mathcal{S}^3$ is a Lie group. To understand the tangent space at a point $q \in \mathcal{S}^3$, we differentiate the constraint $qq^* = 1$ in the direction of a variation $\delta_q \in \mathbb{H}$:

$$[Df(q)](\delta_q) = \lim_{\varepsilon\to 0} \frac{(q+\varepsilon\delta_q)(q+\varepsilon\delta_q)^* - qq^*}{\varepsilon}.$$

Expanding and collecting first order terms yields the constraint:

$$q\delta_q^* + \delta_q q^* = 0.$$

Evaluating this at the identity element $\mathbb{1} = (1, 0, 0, 0)$ *simplifies this condition:*

$$\delta_q^* + \delta_q = 0,$$

implying that δ_q *must be a purely imaginary quaternion. Therefore the Lie algebra* $\mathfrak{s}^3$ *is naturally given by the set of purely imaginary quaternions, which can be naturally associated with vectors in* $\mathbb{R}^3$*.*

Lie Group Integrators

When the system state evolves on a Lie group, there are convenient tools we can leverage to perform integration that respects the group structure. For rigid body orientations represented by a unit quaternion $q \in \mathcal{S}^3$, the time derivative of q lies in the tangent space at that point, i.e., $\dot{q} \in T_q S^3$. Given the angular rate of the body $\boldsymbol{\omega} \in \mathfrak{s}^3$, we can calculate $\dot{q}$ as:

$$\dot{q} = q\boldsymbol{\omega}, \tag{2.8}$$

using standard quaternion multiplication. The formulation (2.8) is justified because the tangent map of left multiplication by a quaternion is itself given by left multiplication, mapping from the Lie algebra to elements of the tangent space at q.[1] Integrating (2.8) results in:

$$q(t) = q(0)\exp(\boldsymbol{\omega}(t)) \tag{2.9}$$

where $\exp : \mathfrak{s}^3 \to S^3$ is termed the exponential map and maps elements of the Lie algebra $\mathfrak{s}^3$ back to the Lie group S^3. This map is injective for imaginary quaternions with magnitude less than π; over this neighborhood, the inverse map is denoted as $\log : S^3 \to \mathfrak{s}^3$. Rather than computing the continuous-time flow explicitly, a discrete approximation is often used — the *Lie-Euler step* — which updates the state via:

$$q_{k+1} = q_k \exp(\boldsymbol{\omega}_k h) \tag{2.10}$$

[1] Often, the equation (2.8) is reported as $\dot{q} = \frac{1}{2}q\boldsymbol{\omega}$. This is because the isomorphism between $\boldsymbol{\omega} \in \mathfrak{s}^3$ and $\mathbb{R}^3$ is given by $\phi(\boldsymbol{\omega}) = \frac{1}{2}\boldsymbol{\omega}$ if the generators of $\mathfrak{s}^3$ are taken to be the canonical basis of imaginary 3-vectors, which arises from the fact that the action of quaternions parameterized by a rotation angle θ on vectors rotates them by an angle of 2θ.

for a (small) time step $h \in \mathbb{R}$. This is a simple example of a *Lie group integrator*, and preserves the unit-norm constraint of the quaternion at every step.

Differential Geometry

As dynamical systems are nothing but vector fields on manifolds, differential geometry provides a natural language to discuss them with. These tools will allow us to reason about the structure present that govern the system's behavior. We begin by defining one of the fundamental operations between vector fields: the Lie bracket.

Definition 2.21 (Lie Bracket)**.** Given two vector fields $\mathbf{f}, \mathbf{g} : \mathcal{X} \to \mathbb{R}^n$, the *Lie bracket* $[\mathbf{f}(\mathbf{x}), \mathbf{g}(\mathbf{x})] =: [\mathbf{f}, \mathbf{g}](\mathbf{x})$ is given by:

$$[\mathbf{f}, \mathbf{g}](\mathbf{x}) \triangleq \left.\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right|_{\mathbf{x}} \mathbf{f}(\mathbf{x}) - \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}} \mathbf{g}(\mathbf{x}) \triangleq \mathbf{ad}_{\mathbf{f}}\mathbf{g}(\mathbf{x})$$

where $\mathbf{ad}$ is termed the adjoint operator.

The Lie bracket captures the interaction between the flows generated by $\mathbf{f}$ and $\mathbf{g}$. It measures the extent to which the vector fields do *not* commute with one another, i.e., how much flowing along $\mathbf{f}$ then $\mathbf{g}$ and then backwards along $\mathbf{f}$ then $\mathbf{g}$ deviates from the initial point, as seen in Figure 2.4. By repeated application of the adjoint operator, we can define higher-order Lie brackets:

$$\begin{aligned}
\mathbf{ad}_{\mathbf{f}}^0\mathbf{g} &= \mathbf{g} \\
\mathbf{ad}_{\mathbf{f}}^1\mathbf{g} &= [\mathbf{f}, \mathbf{g}] \\
&\vdots \\
\mathbf{ad}_{\mathbf{f}}^k\mathbf{g} &= [\mathbf{f}, \mathbf{ad}_{\mathbf{f}}^{k-1}\mathbf{g}] = \underbrace{[\mathbf{f}, [\mathbf{f}, \ldots, [\mathbf{f}}_{\text{k times}}, \mathbf{g}]]].
\end{aligned}$$

Having introduced operations on vector fields, we move to constructing spaces spanned by collections of vector fields — namely, distributions.

Definition 2.22 (Distributions [109])**.** Given a set of smooth vector fields $\mathbf{f}_1, \ldots, \mathbf{f}_m$, the distribution $\boldsymbol{\Delta}(\mathbf{x})$ is defined as:

$$\boldsymbol{\Delta} = \text{span}\{\mathbf{f}_1, \ldots, \mathbf{f}_m\},$$

where the term span is used over the ring of smooth functions, i.e., elements of $\boldsymbol{\Delta}$ at the point $\mathbf{x}$ are of the form:

$$\boldsymbol{\alpha}_1(\mathbf{x})\mathbf{f}_1(\mathbf{x}) + \cdots + \boldsymbol{\alpha}_m(\mathbf{x})\mathbf{f}_m(\mathbf{x})$$

with $\boldsymbol{\alpha}_i(\mathbf{x})$ smooth functions of $\mathbf{x}$.

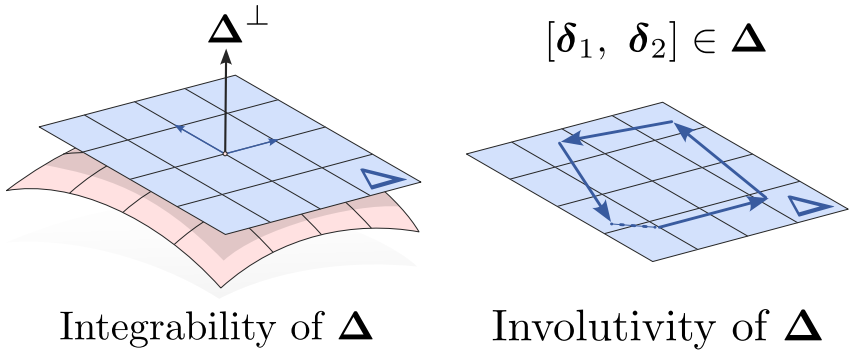


Figure 2.4: Integrability and involutivity of a distribution $\boldsymbol{\Delta}$.

Distributions allow us to reason about available directions of motion at each point in the state space. A natural next question is:

> **Question:** *When does a collection of vector fields form a basis for the tangent space of a manifold?*

That is, given a vector field, when is there an implied invariant manifold? This is precisely captured by the definition of integrability of a distribution:

Definition 2.23. A distribution $\Delta = \text{span}\{\mathbf{f}_1(\mathbf{x}), \ldots, \mathbf{f}_d(\mathbf{x})\}$ is *integrable* if there exist $n - d$ real valued functions $\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_{n-d}$ such that:

$$\text{span}\{d\boldsymbol{\phi}_1, \ldots, d\boldsymbol{\phi}_{n-d}\} = \Delta^{\perp},$$

where $\Delta^{\perp}$ is the codistribution of all covectors that annihilate Δ.

Not all distributions are integrable, however. In order to produce a condition on when a distribution is integrable, we require the notion of involutivity, which ensures that when vector fields are combined through their natural flows, the resulting direction remains within the original distribution:

Definition 2.24. A distribution is involutive if for any two vector fields $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2 \in \boldsymbol{\Delta}(\mathbf{x})$, their Lie bracket $[\boldsymbol{\delta}_1, \boldsymbol{\delta}_2](\mathbf{x}) \in \boldsymbol{\Delta}(\mathbf{x})$.

There is a connection between integrability and involutivity as shown in Figure 2.4, as captured by the following classical result:

Theorem 2.25 (Frobenius)**.** *A distribution is integrable if and only if it is involutive.*

We will leverage this result when understanding how the vector field spanned by our control input affects the system dynamics.

Finally, we introduce the notion of a diffeomorphism, which provides a way to re-parameterize the state space through a change of coordinates.

Definition 2.26. A differentiable function $\mathbf{\Phi} : \mathcal{X} \to \mathbb{R}^n$ is a *diffeomorphism* if it is bijective and has a differentiable inverse $\mathbf{\Phi}^{-1}$.

Proposition 2.27. *A function $\mathbf{\Phi}$ is a (local) diffeomorphism at a point $\mathbf{x}$ if the Jacobian $D\mathbf{\Phi}(\mathbf{x})$ is full rank.*

2.2 Nonlinear Control

The previous section focused on dynamical systems, wherein the vector field defining the system evolution is fixed. There, the central questions were those of analysis: given a system, what can we conclude about its behavior over time? In contrast, the field of control shifts the question to:

> **Question:** *Given a control system, how can we shape the vector field to get the system to do what we want?*

This change in perspective is extremely powerful. The focus now becomes one of synthesis: designing control inputs to steer the system toward objectives such as stability, safety, or performance.

To begin our discussion, consider a general nonlinear control system of the form:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$$

where $\mathbf{x} \in \mathbb{R}^n$ is the system state (returning to the vector space $\mathbb{R}^n$), $\mathbf{u} \in \mathbb{R}^m$ is the *control input*, and $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ now defines the controlled system dynamics. The fundamental challenge of control theory is understanding how the input $\mathbf{u}$ can be used to impact the system evolution.

To proceed systematically, we assume additional structure on the systems we consider — namely, we restrict our attention to control affine systems, where the dynamics can be expressed as:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u} \tag{2.11}$$

for continuously differentiable *drift vector* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ and *actuation matrix* $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^{n \times m}$.

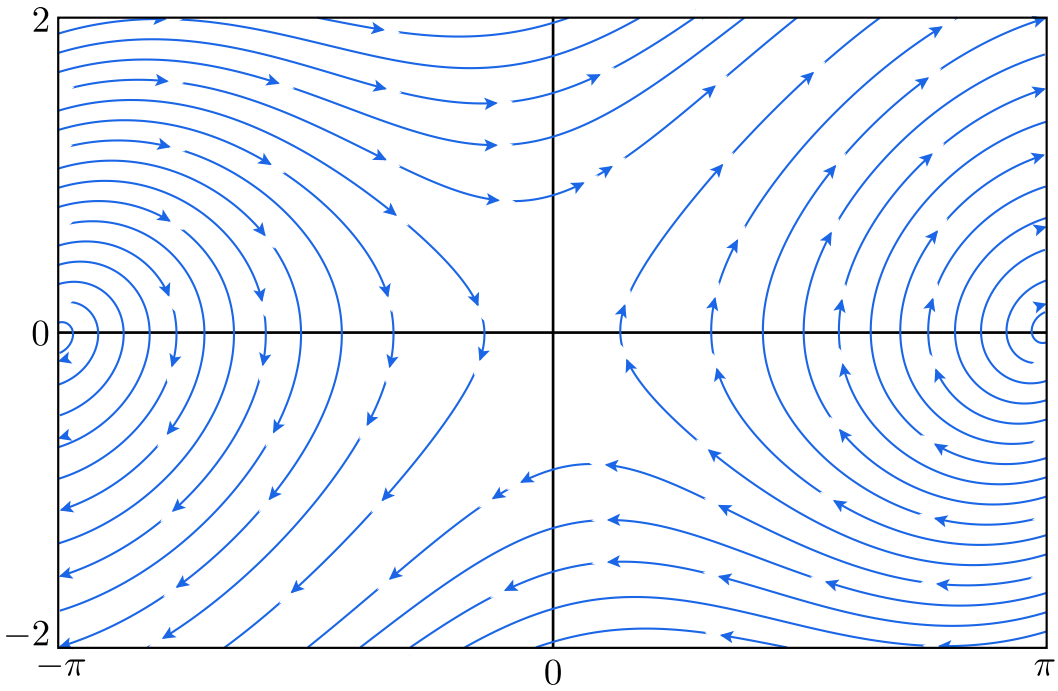


Figure 2.5: The phase portrait of the uncontrolled pendulum.

Example: Controlled Pendulum

Consider the controlled pendulum with angle $\theta \in \mathcal{S}^1$ and angular velocity $\dot{\theta} \in \mathfrak{s}^1$. The dynamics are given by:

$$\frac{d}{dt}\begin{bmatrix}\theta \\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\dot{\theta} \\ -\sin(\theta)\end{bmatrix} + \begin{bmatrix}0 \\ 1\end{bmatrix} u,$$

where $u \in \mathbb{R}$ is the applied torque.

Given that the uncontrolled dynamics of the system are unstable about the upright position, we would like to see if there exists a feedback controller that can instead render it stable. Generalizing this goal results in the leading question:

> **Question:** *Does there exist a choice of control input u that renders an equilibrium point stable?*

Once again, we turn to a notion of linearization, this time exact linearization via state feedback — as a first pass at controller design.

Feedback Linearization

To begin, consider an output $\mathbf{y} : \mathbb{R}^n \to \mathbb{R}^o$ which represents signals that we would like to drive to zero. In order for us to *actually* drive these to zero, the input $\mathbf{u}$ must appear in a derivative of $\mathbf{y}$ in a meaningful way. Taking the first time derivative of the output yields:

$$\dot{\mathbf{y}}(\mathbf{x}) = \underbrace{\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x})}_{\triangleq L_{\mathbf{f}}\mathbf{y}(\mathbf{x})} + \underbrace{\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\mathbf{g}(\mathbf{x})}_{\triangleq L_{\mathbf{g}}\mathbf{y}(\mathbf{x})}\mathbf{u}$$

where $L_{\mathbf{f}}\mathbf{y} : \mathbb{R}^n \to \mathbb{R}^o$ and $L_{\mathbf{g}}\mathbf{y} : \mathbb{R}^n \to \mathbb{R}^o$ denote the *Lie derivatives* of the output y along the vector fields $\mathbf{f}$ and $\mathbf{g}$, respectively. If $L_{\mathbf{g}}\mathbf{y}(\mathbf{x}) \equiv \mathbf{0}$, the control input does not appear in the first derivative, and we must differentiate again. Continuing in this way, we continue differentiating until a higher derivative is nonzero:

$$\mathbf{y}^{(\gamma)}(\mathbf{x}) = L_{\mathbf{f}}^{\gamma}\mathbf{y}(\mathbf{x}) + L_{\mathbf{g}}L_{\mathbf{f}}^{\gamma-1}\mathbf{y}(\mathbf{x})\mathbf{u}.$$

Differentiating the output until the input appears is captured in the following notion of strict relative degree:

Definition 2.28 (Strict Relative Degree [34])**.** An output $\mathbf{y} : \mathbb{R}^n \to \mathbb{R}^o$ for the system (2.11) is said to have relative degree $\gamma \in \mathbb{N}$ at $\mathbf{x}_0$ if:

$$L_{\mathbf{g}}L_{\mathbf{f}}^{k}\mathbf{y}(\mathbf{x}) \equiv 0, \quad 0 \leq k \leq \gamma - 2$$

and $L_{\mathbf{g}}L_{\mathbf{f}}^{\gamma-1}\mathbf{y}(\mathbf{x}) \neq 0$.

Given an output of relative degree $\gamma \in \mathbb{N}$, we can construct a map $\mathbf{\Phi}_{\boldsymbol{\eta}} : \mathbb{R}^n \to \mathcal{N} \triangleq \mathbb{R}^{\gamma}$, defined by:

$$\mathbf{\Phi}_{\boldsymbol{\eta}}(\mathbf{x}) \triangleq \begin{bmatrix}\mathbf{y}(\mathbf{x}) & \dot{\mathbf{y}}(\mathbf{x}) & \cdots & \mathbf{y}^{(\gamma-1)}(\mathbf{x})\end{bmatrix}^{\top}. \tag{2.12}$$

We will subsequently take $\boldsymbol{\eta} = \mathbf{\Phi}_{\boldsymbol{\eta}}(\mathbf{x}) \in \mathcal{N}$ to represent coordinates of the output space and $\boldsymbol{\eta}_i \in \mathbb{R}^{\gamma-1}$ to denote the collection of the ith output and its $\gamma-1$ derivatives.

Valid relative degree allows the constructive synthesis of controllers which exponentially stabilize the outputs [34]. To construct such a controller, assume that the system is *square*, i.e., the number of inputs m is equal to the number of outputs o, and consider the input:

$$\mathbf{k}_{\text{fbl}}(\mathbf{x}, \mathbf{v}) = \left(L_{\mathbf{g}}L_{\mathbf{f}}^{\gamma-1}\mathbf{y}(\mathbf{x})\right)^{-1}\left(-L_{\mathbf{f}}^{\gamma}\mathbf{y}(\mathbf{x}) + \mathbf{v}\right)$$

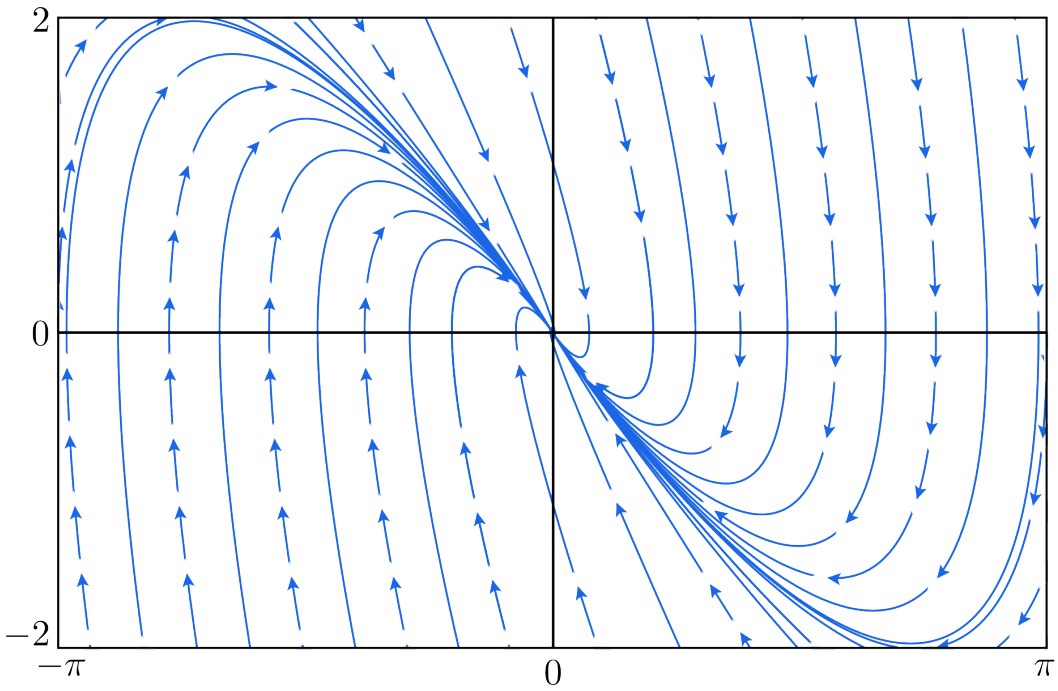


Figure 2.6: The phase portrait of the feedback linearized pendulum.

for $k_{\text{fbl}} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$, and an *auxiliary input* $\mathbf{v} \in \mathbb{R}^m$. Under this controller, the $\boldsymbol{\eta}_i$ dynamics become:

$$\dot{\boldsymbol{\eta}}_i = \underbrace{\begin{bmatrix} \mathbf{0} & \mathbf{I} \\ 0 & \mathbf{0} \end{bmatrix}}_{\triangleq \mathbf{F}_i} \boldsymbol{\eta}_i + \underbrace{\begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}}_{\triangleq \mathbf{G}_i} v_i,$$

for $\mathbf{F}_i \in \mathbb{R}^{\gamma-1 \times \gamma-1}$ and $\mathbf{G}_i \in \mathbb{R}^{\gamma-1}$. We can then collect the output dynamics to be:

$$\dot{\boldsymbol{\eta}} = \mathbf{F}\boldsymbol{\eta} + \mathbf{G}\mathbf{v} \tag{2.13}$$

for $\mathbf{F} \in \mathbb{R}^{\gamma \times \gamma}$ and $\mathbf{G} \in \mathbb{R}^{\gamma}$ constructed by concatenating the $\mathbf{F}_i$ and $\mathbf{G}_i$ components. Observe that the dynamics in (2.13) are linear and controllable — in fact, they are simply integrator dynamics. As such, we can readily design a feedback matrix $\mathbf{K} \in \mathbb{R}^{m \times \gamma}$ such that choosing the auxiliary input to be $\mathbf{v} = \mathbf{K}\boldsymbol{\eta}$ yields the stable closed loop system:

$$\dot{\boldsymbol{\eta}} = \underbrace{(\mathbf{F} - \mathbf{G}\mathbf{K})}_{\mathbf{A}_{cl}} \boldsymbol{\eta}. \tag{2.14}$$

Recall that a linear system is exponentially stable if the eigenvalues are in the left half plane — therefore, any choice of feedback matrix $\mathbf{K}$ which achieves this will render the nonlinear system exponentially stable.

Remark: *For robotic systems, outputs defined on the position coordinates typically have relative degree two. Consequentially, relative degree $\gamma = 2$ outputs will be the primary focus in later developments.*

Example: Feedback Linearizing the Pendulum

Consider the output $y = \theta - \pi$, whose zero value corresponds to the pendulum being upright. Differentiating this output twice yields:

$$\boldsymbol{\eta} = \begin{bmatrix} y \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \dot{\theta} \\ -\sin\theta \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} v.$$

Therefore, the feedback linearizing controller is given by:

$$u = \sin(\theta) + v$$

for some auxiliary input $v \in \mathbb{R}$. Taking this input to be $v = -k_p y - k_d \dot{y}$ for positive constants $k_p, k_d > 0$ ensures that the linear system is Hurwitz, which guarantees that the closed loop system is stable to the upright equilibrium.

Trajectory Tracking

When the desired output is time-varying, special care must be taken in the controller design. Consider the error signal:

$$\mathbf{y}(\mathbf{x}, t) = \mathbf{y}_a(\mathbf{x}) - \mathbf{y}_d(t)$$

for measured outputs $\mathbf{y}_a : \mathcal{X} \to \mathbb{R}^m$ and smooth desired outputs $\mathbf{y}_d : \mathbb{R}_{\geq 0} \to \mathbb{R}^m$. Once again, we can construct output coordinates as $\boldsymbol{\eta} = \boldsymbol{\eta}_a - \boldsymbol{\eta}_d$ where $\boldsymbol{\eta}_d$ collects $\mathbf{y}_d$ and its time derivatives up to $\gamma - 1$. The feedback linearizing control input now takes the form:

$$\mathbf{k}_{\text{fbl}}(\mathbf{x}, t, \mathbf{v}) = \left(L_{\mathbf{g}} L_{\mathbf{f}}^{\gamma-1} \mathbf{y}_a(\mathbf{x})\right)^{-1} (-L_{\mathbf{f}}^{\gamma} \mathbf{y}_a(\mathbf{x}) + \mathbf{y}_d^{(\gamma)}(t) + \mathbf{v}).$$

Setting $\mathbf{v} = -\mathbf{K}(\boldsymbol{\eta}_a - \boldsymbol{\eta}_d(t))$ for an appropriately designed matrix $\mathbf{K}$ yields stability of the time varying output coordinates., ensuring that $\mathbf{y}_a(\mathbf{x}) \to \mathbf{y}_d(t)$ as $t \to \infty$.

Control Lyapunov Functions

Now that we have produced an example of a stabilizing controller for a collection of outputs of the nonlinear system (2.11), we again seek a certificate of this stability. To this end, we turn to Control Lyapunov Functions:

Definition 2.29. [114] For the system (2.11), $V : \mathbb{R}^n \to \mathbb{R}$ is an *exponentially stabilizing control Lyapunov function (ES-CLF)* if there exists a $k_1, k_2, k_3 > 0$, such that:

$$\begin{aligned} k_1\|\mathbf{x}\|^2 \leq V(\mathbf{x}) \leq k_2\|\mathbf{x}\|^2 \\ \inf_{\mathbf{u}} \dot{V}(\mathbf{x}, \mathbf{u}) \leq -k_3 V(\mathbf{x}). \end{aligned} \tag{2.15}$$

Remark: *Control Lyapunov functions exist for other stability types as well (e.g., asymptotic), but in this thesis we will restrict our attention to exponential control Lyapunov functions.*

Given a candidate Lyapunov function (i.e., a positive definite function), verifying that it satisfies (2.15) is in general highly nontrivial — it requires checking a linear inequality everywhere in the state space. Instead, we often work backwards by first using feedback linearization as a proof of existence of a stabilizing controller, and second using converse Lyapunov theory to generate a control Lyapunov function for analysis. To demonstrate this workflow, recall the closed loop system $\dot{\boldsymbol{\eta}} = \mathbf{A}_{\text{cl}}\boldsymbol{\eta}$ from (2.14) achieved by applying the feedback linearizing control input. As this linear system is exponentially stable, we know from (2.4) that for any positive definite matrix $\mathbf{Q} \succ 0$, there exists a unique positive definite matrix $\mathbf{P} \succ 0$ satisfying the (CTLE). Thus, the function $V(\boldsymbol{\eta}) = \boldsymbol{\eta}\mathbf{P}\boldsymbol{\eta}$ is a Lyapunov function satisfying:

$$\begin{aligned} \lambda_{\min}(\mathbf{P})\|\boldsymbol{\eta}\|^2 \leq V(\boldsymbol{\eta}) \leq \lambda_{\max}(\mathbf{P})\|\boldsymbol{\eta}\|^2 \\ \dot{V}(\boldsymbol{\eta}, \mathbf{v}) = L_{\mathbf{F}}V(\boldsymbol{\eta}) + L_{\mathbf{G}}V(\boldsymbol{\eta})\mathbf{v} \leq -\frac{\lambda_{\min}(\mathbf{Q})}{\lambda_{\max}(\mathbf{P})}V(\boldsymbol{\eta}) \end{aligned}$$

and thus serves as a Control Lyapunov Function on the output coordinates. As exponential stability is preserved through diffeomorphism [26], if $\boldsymbol{\Phi}_{\boldsymbol{\eta}}$ represents a diffeomorphism between the entire state space and the output coordinates, then the function $V(\boldsymbol{\Phi}_{\boldsymbol{\eta}}(\mathbf{x}))$ is a Control Lyapunov function for the state $\mathbf{x}$.

If this map does not represent a diffeomorphism, i.e., there are remaining states, a natural question arises:

> **Question:** *If the outputs are stabilized, what happens to the remaining system states?*

The necessary tools to address this question are presented in Section 2.4, and constructively answering this question will serve as the inspiration for many of the later sections in this thesis. Before doing so, we first review some additional topics from classical nonlinear control theory.

Input to State Stable Control Lyapunov Functions

Very similar to the notion of an ISS Lyapunov function is the definition of an EISS-CLF:

Definition 2.30. For the nonlinear control system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u}$, a continuously differentiable function $V : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ is an *Exponential Input to State Stabilizing Control Lyapunov Function* (EISS-CLF) if it satisfies:

$$k_1\|\mathbf{x}\|^c \leq V(\mathbf{x}) \leq k_2\|\mathbf{x}\|^c$$
$$\inf_{\mathbf{u}\in\mathbb{R}^m}[L_{\mathbf{f}}V(\mathbf{x}) + L_{\mathbf{g}}V(\mathbf{x})\mathbf{u}] + \mathbf{d} \leq -k_3\|\mathbf{x}\|^c + \sigma(\|\mathbf{d}\|_\infty)$$

for $k_i, c > 0$ and $\sigma \in \mathcal{K}_\infty$.

We now present a lemma that will be critical for later constructions:

Lemma 2.31. *Consider the nominal system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u}$. If the equilibrium $\mathbf{x}^* = 0$ is exponentially stable under the controller $\mathbf{u} = \mathbf{k}(\mathbf{x})$, then there exists an EISS-CLF V satisfying the conditions in Definition 2.30 which certifies that the perturbed system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + g(\mathbf{x})\mathbf{k}(\mathbf{x}) + \mathbf{d}$ is E-ISS. In particular, we have that:*

$$\|\mathbf{x}\| \geq \frac{2k_4}{k_3}\|\mathbf{d}\|_\infty \implies \dot{V}(\mathbf{x}, \mathbf{k}(\mathbf{x}), \mathbf{d}) \leq -\frac{k_3}{2}\|\mathbf{x}\|^2.$$

This states that if we can design a controller that is exponentially stabilizing for a nonlinear system, we automatically obtain a certificate of exponential input-to-state stability for the perturbed system.

Control Barrier Functions

Paralleling the construction of barrier functions introduced in Section 2.1, we now extend the notion of safety to controlled systems via Control Barrier Functions:

Definition 2.32 (Control Barrier Function)**.** Let $\mathcal{C} \subset \mathbb{R}^n$ be the 0-superlevel set of a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$. The function h is said to be a *control barrier function* (CBF) for the control system (2.11) if there exists an extended class $\mathcal{K}_\infty$ function, $\alpha \in \mathcal{K}_\infty^e$ such that the following inequality holds for all $\mathbf{x} \in \mathbb{R}^n$:

$$\sup_{\mathbf{u}\in\mathbb{R}^m} \dot{h}(\mathbf{x}, \mathbf{u}) = \sup_{\mathbf{u}\in\mathbb{R}^m} [L_{\mathbf{f}}h(\mathbf{x}) + L_{\mathbf{g}}h(\mathbf{x})\mathbf{u}] \geq -\alpha(h(\mathbf{x})). \tag{2.16}$$

A control barrier function ensures that at every point in the state space, there exists a control input $\mathbf{u}$ capable of maintaining the forward invariance of the safe set $\mathcal{C}$.

Remark: *Verifying the CBF condition* (2.16) *over the state space is, in general, highly nontrivial and as such the construction of valid Control Barrier Functions remains an open and challenging problem. There exists a growing body of literature devoted to the synthesis and application of CBFs — we refer the reader to [111], [115] for a deeper exploration of these topics.*

Projection to State Safety

Just as Input-to-State Stability provides a framework for analyzing the stability of a dynamical system under disturbances, we now seek an analogous framework for safety. In the presence of disturbances, it is unlikely that the original set $\mathcal{C}$ is forward invariant. Instead, we aim to characterize an enlarged set $\mathcal{C}_\delta \supset \mathcal{C}$ that accounts for the disturbance and can be rendered forward invariant:

Definition 2.33 (*Projection to State Safety (PSSf)* ([116]))**.** Given a feedback controller $\mathbf{k}$, the closed-loop system $\dot{\mathbf{x}} = \mathbf{f}_{\text{cl}}(\mathbf{x}) = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{k}(\mathbf{x}) + \mathbf{d}(t)$ is *projection to state safe* (PSSf) on $\mathcal{C}$ with respect to the function h and projected disturbances $\delta : \mathbb{R}_{\geq 0} \to \mathbb{R}$ if there exists $\overline{\delta} > 0$ and $\gamma \in \mathcal{K}_\infty$ such that the set $\mathcal{C}_\delta \supset \mathcal{C}$,

$$\mathcal{C}_\delta \triangleq \{\mathbf{x} \in \mathbb{R}^n : h(\mathbf{x}) + \gamma(\|\delta\|_\infty) \geq 0\}, \tag{2.17}$$

is forward invariant for all δ satisfying $\|\delta\|_\infty \leq \overline{\delta}$.

This concept allows us to quantify the degradation of the safe set under uncertain dynamics and environments, and will be used during our later analysis.

2.3 Robotic Systems

Having introduced a general theory of dynamics and control, we now focus our attention on robotic system models, with particular emphasis on legged robots.

Continuous Dynamics

Let $\mathbf{q} \in \mathcal{Q}$ denote the configuration coordinates and $\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}}) \in \mathcal{X} \triangleq \mathsf{T}\mathcal{Q}$ denote the state of an $n-$degree of freedom robotic system. Various constraints on the system evolution will become active as the robot enters different modes of operation. These constraints are often modeled as *holonomic constraints*, which are integrable constraints depending only on the configuration. In order to model this, let $\mathbf{h} : \mathcal{Q} \to \mathbb{R}^h$ represent the active holonomic constraints, for example the

condition that the height of the stance foot of a walking robot is zero. Enforcing $\mathbf{h}(\mathbf{q}) = \mathbf{0}$ constrains the evolution of the system, which is seen by differentiating twice:

$$\begin{aligned}\mathbf{0} &= \mathbf{J}_h(\mathbf{q})\dot{\mathbf{q}} \\ \mathbf{0} &= \dot{\mathbf{J}}_h(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{J}_h(\mathbf{q})\ddot{\mathbf{q}},\end{aligned}$$

where $\mathbf{J}_h = \partial\mathbf{h}/\partial\mathbf{q}$ is the Jacobian of the holonomic constraint. Combining these with the unconstrained Euler-Lagrange equations of motion yields the following constrained dynamics:

$$\begin{bmatrix}\mathbf{D}(\mathbf{q}) & \mathbf{J}_h^\top(\mathbf{q}) \\ \mathbf{J}_h(\mathbf{q}) & \mathbf{0}\end{bmatrix}\begin{bmatrix}\ddot{\mathbf{q}} \\ \boldsymbol{\lambda}\end{bmatrix} = \begin{bmatrix}-\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}) + \mathbf{B}\mathbf{u} \\ -\dot{\mathbf{J}}_h(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}\end{bmatrix}$$

where $\mathbf{D} : \mathcal{Q} \to \mathbb{R}^{n\times n}$ is the positive-definite mass-inertia matrix, $\mathbf{C} : \mathcal{X} \to \mathbb{R}^{n\times n}$ contains the Coriolis forces, $\mathbf{G} : \mathcal{Q} \to \mathbb{R}^n$ the gravity terms, $\mathbf{B} \in \mathbb{R}^{n\times m}$ is the selection matrix, $\mathbf{u} \in \mathbb{R}^m$ is the control input, and $\boldsymbol{\lambda}$ are the reaction forces imposed by the holonomic constraints, which can be thought of as Lagrange multipliers as presented in Section 2.5.

Solving for $\boldsymbol{\lambda}$ yields:

$$\boldsymbol{\lambda} = -(\mathbf{J}_h\mathbf{D}^{-1}\mathbf{J}_h)^{-1}\left(\dot{\mathbf{J}}_h\dot{\mathbf{q}} + \mathbf{J}_h\mathbf{D}^{-1}(\mathbf{B}\mathbf{u} - \mathbf{C}\dot{\mathbf{q}} - \mathbf{G})\right)$$

where the arguments have been suppressed for readability. This results in the closed-form constrained dynamics, which can be written in the canonical control-affine form:

$$\dot{\mathbf{x}} = \underbrace{\begin{bmatrix}\dot{\mathbf{q}} \\ -\mathbf{D}(\mathbf{q})^{-1}\mathbf{H}(\mathbf{q},\dot{\mathbf{q}})\end{bmatrix}}_{\mathbf{f}(\mathbf{x})} + \underbrace{\begin{bmatrix}\mathbf{0} \\ \mathbf{D}(\mathbf{q})^{-1}\mathbf{B}\end{bmatrix}}_{\mathbf{g}(\mathbf{x})}\mathbf{u} + \mathbf{J}_h(\mathbf{q})^\top\boldsymbol{\lambda} \tag{2.18}$$

where $\mathbf{H} : \mathcal{X} \to \mathbb{R}^n$ collects the Coriolis and Gravity terms.

Discrete Dynamics

When the robot impacts the ground, it experiences impulsive effects governed by the momentum transfer equation. To describe this, let $\mathbf{h} : \mathcal{X} \to \mathbb{R}^h$ denote a collection of holonomic constraints that become active during the continuous mode just after impact. The momentum transfer across impact satisfies:

$$\mathbf{D}(\mathbf{q}^+)\dot{\mathbf{q}}^+ + \mathbf{J}_h(\mathbf{q}^-)\boldsymbol{\Delta}\mathbf{F} = \mathbf{D}(\mathbf{q}^-)\dot{\mathbf{q}}^-$$

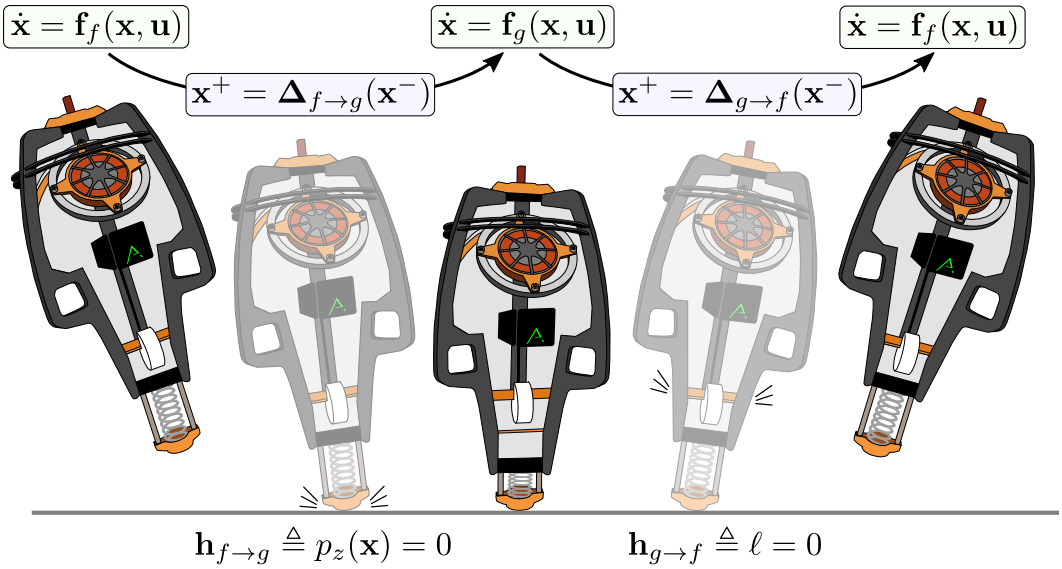


Figure 2.7: A 3D hopping robot traversing various hybrid domains.

where $\cdot^-$ denotes pre-impact quantities, $\cdot^+$ denotes post-impact quantities, and $\boldsymbol{\Delta}\mathbf{F}$ is the impulse that occurs through impact. To ensure consistency with the new contact mode, the holonomic constraint in the post-impact phase must be satisfied. Enforcing $\mathbf{h}(\mathbf{q}^+) = 0$ ensures position consistency, and differentiating this constraint once yields a velocity-level constraint:

$$\mathbf{J}_h^\top(\mathbf{q}^+)\dot{\mathbf{q}}^+ = \mathbf{0}.$$

Collecting these two equations yields a system of linear equations:

$$\begin{bmatrix} \mathbf{D}(\mathbf{q}^+) & \mathbf{J}_h(\mathbf{q}^-) \\ \mathbf{J}_h^\top(\mathbf{q}^+) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}}^+ \\ \boldsymbol{\Delta}\mathbf{F} \end{bmatrix} = \begin{bmatrix} \mathbf{D}(\mathbf{q}^-)\dot{\mathbf{q}}^- \\ \mathbf{0} \end{bmatrix}$$

which can be solved to determine the post impact velocity $\dot{\mathbf{q}}^+$. Solving for the impulse force $\boldsymbol{\Delta}\mathbf{F}$ yields:

$$\dot{\mathbf{q}}^+ = \underbrace{\dot{\mathbf{q}}^- - \mathbf{D}(\mathbf{q}^+)^{-1}\mathbf{J}_h(\mathbf{q}^-)\boldsymbol{\Delta}\mathbf{F}}_{\triangleq \boldsymbol{\Delta}(\mathbf{x}^-)}.$$

Hybrid Systems

A *hybrid system* experiences alternating phases of continuous and discrete dynamics. Such systems arise naturally in legged locomotion, the primary application considered in this thesis.

We model the hybrid system via a directed graph of vertices $v \in V$, each of which contain a unique set of continuous-time dynamics that the system evolves under, and edges $e \in E$ which characterize how the robot traverses the continuous modes. For each vertex $v \in V$, let $\mathcal{D}_v \subset \mathcal{X}$ represent the admissible domain in which the system state evolves, and n_v denote the number of holonomic constraints restricting the motion of the robot. Each hybrid transition $e = (v_1 \to v_2) \in E$ occurs when the system state intersects the *guard*, denoted as $\mathcal{S}_{v_1} \subset \mathcal{X}$, triggering a discrete reset governed by:

$$\mathbf{x}^+ = \mathbf{\Delta}_e(\mathbf{x}^-),$$

where $\mathbf{\Delta}_e : \mathcal{S}_{v_1} \to \mathcal{D}_{v_2}$ is the *reset map* describing the momentum transfer though impact, and $\mathbf{x}^- \in \mathcal{D}_{v_1}$ and $\mathbf{x}^+ \in \mathcal{D}_{v_2}$ are the pre and post-impact states, respectively. Collecting the various objects $\mathcal{D} = \{\mathcal{D}_v\}_{v\in V}$, $\mathcal{S} = \{\mathcal{S}_v\}_{v\in V}$, $\mathbf{\Delta} = \{\mathbf{\Delta}_e\}_{e\in E}$ and $F = \{\mathbf{f}_v\}_{v\in V}$, we can describe a hybrid control system via the tuple:

$$\mathcal{HC} = (V, E, \mathcal{D}, \mathcal{S}, \mathbf{\Delta}, F).$$

Remark: *This modeling choice may feel strong for other systems which experience intermittent contact, such as manipulation-based systems, as there are likely significantly more modes of operation.*

2.4 Underactuated Systems

The term *underactuation* refers to robotic systems that have fewer actuators than degrees of freedom. This imposes limitations on the achievable behaviors and necessitates careful consideration of how to most effectively use the actuation that is present. Many systems experience underactuation, from walking robots to swimming robots, and as such it is of general interest to construct platform-agnostic techniques for controlling such systems. To begin, we introduce what it means to be underactuated:

Definition 2.34. The robotic system (2.18) is said to be underactuated if rank$(B) <$ n, full actuated if rank$(B) = n$, and overactuated if rank$(B) > n$.

Given the tools developed in previous subsections, a natural first question is:

> **Question:** *Does there exist an output such that my system is full state feedback linearizeable?*

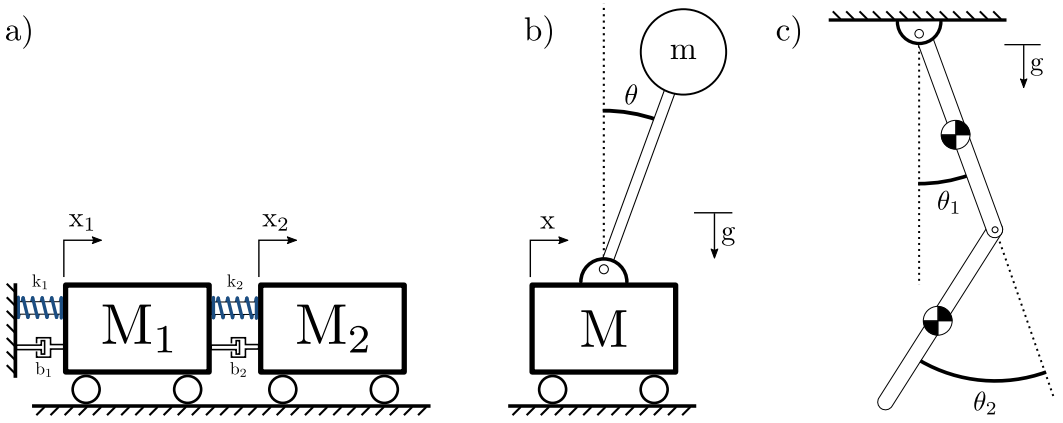


Figure 2.8: Classical examples of underactuated systems: **a)** Mass Spring Damper Model, **b)** Cart Pole Model, and **c)** Double Pendulum.

The existence of such an output would be convenient, as then we could readily apply the results from the previous section to stabilize the full system state. There exists a very elegant result from geometric nonlinear control theory that provides necessary and sufficient conditions under which a system can be rendered full state feedback linearizeable, which leverages the tools from Section 2.1:

Theorem 2.35 (Sastry, Brockett, Isidori)**.** *There exists an output* $\mathbf{y}(\mathbf{x})$ *such that the system* (2.11) *is full-state feedback linearizeable if and only if the following conditions are met:*

1. $\text{rank}\Big(\Big[\mathbf{ad}_{\mathbf{f}}^0\mathbf{g}(\mathbf{x}) \quad \cdots \quad \mathbf{ad}_{\mathbf{f}}^{n-1}\mathbf{g}(\mathbf{x})\Big]\Big) = n,$
2. The distribution $\boldsymbol{\Delta}(\mathbf{x}) = \text{span}\{\mathbf{ad}_{\mathbf{f}}^0\mathbf{g}(\mathbf{x}), \ldots, \mathbf{ad}_{\mathbf{f}}^{n-2}\mathbf{g}(\mathbf{x})\}$ is involutive.

The first condition is a controllability-like rank test, and is satisfied for almost all pairs of vector fields $\mathbf{f}, \mathbf{g}$ [109]. The second, however, imposes a stricter requirement and is much more difficult to verify for general systems. To illustrate the practical difficulty of applying this result, we turn to two motivating examples.

Example: 2D Mass Spring Damper

Consider the double mass-spring damper system in Figure 2.8a, in which actuation is only applied to the first mass. The equations of motion are:

$$\begin{bmatrix} M_1 & 0 \\ 0 & M_2 \end{bmatrix} \begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} + \begin{bmatrix} k_2(x_2 - x_1) - k_1 x_1 + b_2(\dot{x}_2 - \dot{x}_1) - b_1 \dot{x}_1 \\ -k_2(x_2 - x_1) - b_2(\dot{x}_2 - \dot{x}_1) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} u,$$

and the systems is underactuated because rank(B)$= 1 < 2$. Despite this, observe that the system is controllable. Next, two vector fields $\delta_1(x), \delta_2(x) \in \Delta(x)$ as:

$$\delta_1(x) = \sum_{i=0}^{n-1} \alpha_i(x) A^i B,$$
$$\delta_2(x) = \sum_{i=0}^{n-1} \beta_i(x) A^i B.$$

Then, their Lie bracket is given by:

$$\begin{aligned} [\delta_1, \delta_2](x) &= \sum_{i=0}^{n-1} \left(\frac{\partial \beta_i}{\partial x} \alpha_i(x) A^i B - \frac{\partial \alpha_i}{\partial x} \beta_i(x) A^i B \right) \\ &= \sum_{i=0}^{n-1} \underbrace{\left(\frac{\partial \beta_i}{\partial x} \alpha_i(x) - \frac{\partial \alpha_i}{\partial x} \beta_i(x) \right)}_{\triangleq \gamma_i(x)} A^i B, \end{aligned}$$

where each $\gamma_i(\mathbf{x})$ is smooth, implying $[\delta_1, \delta_2](\mathbf{x}) \in \Delta(\mathbf{x})$ and the distribution is involutive. Therefore, by Theorem 2.35, there exists an output $\mathbf{y}$ such that the system is full state feedback linearizeable. In fact, we can intuit that the output $y = x_2$ is a relative degree $k = n = 4$ output, and therefore allows us to full state feedback linearize the system.

The above analysis holds for any linear system — perhaps obviously, for any controllable linear system there exists an output such that the system is full state feedback linearizeable, regardless of whether it is underactuated or not. The next example demonstrates that even simple nonlinearities significantly increase the complexity of the analysis.

Example: Cart Pole

Consider the cart pole in Figure 2.8b, where actuation is applied only to the cart. The equations of motion are given by:

$$\begin{bmatrix} M+m & ml\cos\theta \\ ml\cos\theta & ml^2 \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} -ml\dot{\theta}^2\sin\theta \\ mgl\sin\theta \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} u,$$

which is again underactuated. To apply Theorem 2.35, we must compute the iterated Lie brackets:

$$\begin{aligned} \boldsymbol{ad}_f^0 g &= g = \begin{bmatrix} 0 \\ D(q)^{-1}B \end{bmatrix} \\ \boldsymbol{ad}_f^1 g &= [f,g] = \begin{bmatrix} D(q)^{-1}B \\ \alpha(q,\dot{q}) \end{bmatrix} \\ \boldsymbol{ad}_f^2 g &= [f,[f,g]] = \begin{bmatrix} \beta(q,\dot{q}) \\ \gamma(q,\dot{q}) \end{bmatrix} \end{aligned}$$

for appropriately defined (messy) functions α, β, γ. Is the distribution defined by the span of the above vector fields involutive? Your guess is as good as mine.

As we can see from these examples, the fact that a system is underactuated is insufficient to determine whether or not it is full state feedback linearizeable in general. In fact, for certain classes of underactuated systems, such outputs can exist [108]. These examples present a broader point though: even if the conditions of Theorem 2.35 could be verified, doing so does not aid in the synthesis of such an output. If true, we would know finding such an output is *possible* to find, but we would not know how to do so. Therefore, we abandon this line of thinking, and instead presume that systems with underactuation are not feedback linearizeable. Given this new perspective:

> **Question:** *How can we produce control inputs which stabilize underactuated systems?*

Answering this question will guide much of the remaining structure in this thesis. In the next section, we begin developing the necessary machinery required to address it. We begin by constructing a lens through which we can analyze not just the evolution of the output coordinates, but the complete system state.

Zero Dynamics Coordinates

Continuing from the previous section, consider a vector relative degree $r < n$ output $\mathbf{y}(\mathbf{x})$ and its associated error coordinates $\boldsymbol{\eta} \in \mathbb{R}^r$. The first question we would like to address is:

> **Question:** *When controlling the output* $\mathbf{y}$*, what happens to the complete system state* $\mathbf{x}$*?*

Given the error coordinates map $\boldsymbol{\Phi}_{\boldsymbol{\eta}} : \mathcal{X} \to \mathbb{R}^r$, we would like to construct a complete change of coordinates $\boldsymbol{\Phi} : \mathcal{X} \to \mathbb{R}^n$ that includes both the error and a complementary set of coordinates. Since $\boldsymbol{\Phi}_{\boldsymbol{\eta}}$ has r independent coordinates and therefore forms an r-dimensional basis [24], we must augment it with $n-r$ additional coordinates that are orthogonal to $\boldsymbol{\eta}$. The following classical result guarantees that this is always possible:

Proposition 2.36 (Existence of Output Nullspace Coordinates [24])**.** *Given an output* $\mathbf{y}$ *of vector relative degree* r*, it is always possible to find* $n - r$ *more functions* $\mathbf{z} = \boldsymbol{\Phi}_{\mathbf{z}}(\mathbf{x}) = [z_1(\mathbf{x}), \ldots, z_{n-r}(\mathbf{x})]$ *such that the mapping* $\boldsymbol{\Phi}(\mathbf{x}) = [\boldsymbol{\Phi}_{\boldsymbol{\eta}}(\mathbf{x})\ \boldsymbol{\Phi}_{\mathbf{z}}(\mathbf{x})]$ *is a (local) diffeomorphism.*

This result allows us to express the system coordinates in $(\boldsymbol{\eta}, \mathbf{z}) \in \mathcal{N} \times \mathcal{Z}$ space, where $\boldsymbol{\eta}$ captures the output dynamics and $\mathbf{z}$ captures the remaining behavior.

A subtle but import complication remains: although the input is being fully used to regulate the output $\boldsymbol{\eta}$, the residual dynamics $\dot{\mathbf{z}}$ may still appear to be effected by the control input. There is a powerful result that provides conditions under which the the zeroing coordinates can be expressed in a way that does not depend on the control input:

Proposition 2.37. *If the distribution* $\mathbf{G} = \mathit{span}\{\mathbf{g}_1(\mathbf{x}), \ldots, \mathbf{g}_m(\mathbf{x})\}$ *is involutive, then it is always possible to find* $n - r$ *more functions* $\mathbf{z} = \boldsymbol{\Phi}_{\mathbf{z}}(\mathbf{x}) = [z_1(\mathbf{x}), \ldots, z_{n-r}(\mathbf{x})]$ *such that* $L_{\mathbf{g}_j} z_i(\mathbf{x}) = 0$ *for all* $r+1 \leq i \leq n$ *and all* $1 \leq j \leq m$*.*

This again is a challenging condition to check in general. Luckily, however, in the next section we show that this is always possible for the robotic systems of interest.

With such coordinates, the residual dynamics are autonomous:

$$\dot{\mathbf{z}} = \underbrace{L_{\mathbf{f}} \boldsymbol{\Phi}_{\mathbf{z}} \circ \boldsymbol{\Phi}^{-1}(\boldsymbol{\eta}, \mathbf{z})}_{\triangleq \boldsymbol{\omega}(\boldsymbol{\eta}, \mathbf{z})},$$

where, importantly, they do not depend on the input $\mathbf{u}$. This enables us to analyze the behavior of such coordinates regardless of the control input applied. Furthermore, we can analyze what happens and design controllers such that they have desirable properties.

Remark: *If the condition in Proposition 2.37 is satisfied, then the system will always have a subspace that evolves autonomously, independently of the control input. As such, one should always choose coordinates that satisfy the invariance condition to provide a basis in which we can analyze this autonomous behavior.*

One of the benefits of this decomposition is the following stability result:

Theorem 2.38 ([24])**.** *Assume that the output coordinates $\boldsymbol{\eta}$ and $\mathbf{z}$ are exponentially stable with Lyapunov functions $V_{\boldsymbol{\eta}}(\boldsymbol{\eta})$ and $V_{\mathbf{z}}(\mathbf{z})$, respectively. For suitably chosen $\sigma > 0$, we have that $V = \sigma V_{\boldsymbol{\eta}}(\boldsymbol{\eta}) + V_{\mathbf{z}}(\mathbf{z})$ is an exponential Lyapunov function in the output and zero dynamics space.*

This result is extremely powerful — using our previously synthesized controllers which exponentially stabilize the outputs, if we can ensure that the (low dimensional) zeroing manifold is stable, then we can show stability of the complete system. One important issue remains unresolved:

Question: *How do we ensure that the zero dynamics are stable?*

This question, again, will serve as a critical motivator for many of the constructions later in this thesis, and lies at the heart of nonlinear control for underactuated systems.

Robotic Underactuated Systems

Once again, verifying the conditions in Proposition 2.37 is generally infeasible. However, for robotic systems, we can avoid this analysis by constructively generating zero dynamics coordinates that do not depend on the control input. To this end, assume that $\mathbf{B} \in \mathbb{R}^{n \times m}$ is full column rank and is tall, i.e., $m < n$, as is the case in underactuated systems. We define the following coordinate change:

$$\boldsymbol{\eta} = \boldsymbol{\Phi}_{\boldsymbol{\eta}}(\mathbf{x}) \triangleq \begin{bmatrix} \mathbf{B}^\top \mathbf{q} \\ \mathbf{B}^\top \dot{\mathbf{q}} \end{bmatrix}, \quad \mathbf{z} = \boldsymbol{\Phi}_{\mathbf{z}}(\mathbf{x}) \triangleq \begin{bmatrix} \mathbf{N}\mathbf{q} \\ \mathbf{N}\mathbf{D}(\mathbf{q})\dot{\mathbf{q}} \end{bmatrix} \tag{2.19}$$

for $\boldsymbol{\eta} \in \mathcal{N} \subset \mathcal{X}$ and $\mathbf{z} \in \mathcal{Z} \subset \mathcal{X}$, where $\mathbf{N} \in \mathbb{R}^{(n-m) \times n}$ is chosen to be a basis for the left nullspace of $\mathbf{B}$. Let us check the two required conditions: first, does the

coordinate change $\mathbf{\Phi}(\mathbf{x}) \triangleq (\mathbf{\Phi}_{\boldsymbol{\eta}}(\mathbf{x}), \mathbf{\Phi}_{\mathbf{z}}(\mathbf{x}))$ constitute a diffeomorphism between $\mathcal{X}$ and $\mathcal{N} \times \mathcal{Z}$? Take the Jacobian at a point $\mathbf{x}$:

$$\begin{bmatrix} \mathbf{B}^\top & \mathbf{0} \\ \mathbf{0} & \mathbf{B}^\top \\ \mathbf{N} & \mathbf{0} \\ \frac{d\mathbf{D}(\mathbf{q})}{d\mathbf{q}}\dot{\mathbf{q}} & \mathbf{N}\mathbf{D}(\mathbf{q}) \end{bmatrix}.$$

By construction, the null space of the row space of $\mathbf{B}$ is the orthogonal complement of the column space of $\mathbf{B}^\top$. Because $\mathbf{D}(\mathbf{q})$ is positive definite and therefore full rank for all $\mathbf{q}$, the rows of $\mathbf{N}\mathbf{D}(\mathbf{q})$ span the same space as the rows of $\mathbf{N}$. Thus the Jacobian is full rank and the transformation is locally invertible.

Next, verify that $L_{\mathbf{g}}\mathbf{\Phi}_{\mathbf{z}} \equiv \mathbf{0}$:

$$\begin{bmatrix} \mathbf{N} & \mathbf{0} \\ \frac{d\mathbf{D}(\mathbf{q})}{d\mathbf{q}}\dot{\mathbf{q}} & \mathbf{N}\mathbf{D}(\mathbf{q}) \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ \mathbf{D}(\mathbf{q})^{-1}\mathbf{B} \end{bmatrix}$$

which is equivalently zero as $\mathbf{N}\mathbf{B} = \mathbf{0}$. This implies that the $\mathbf{z}$ dynamics are independent of the control input. With this, the transformation in (2.19) is a valid decomposition, and therefore $\mathbf{\Phi}^{-1}$ exists and any conclusions of stability of $(\boldsymbol{\eta}, \mathbf{z})$ are directly transferable back to $\mathbf{x}$.

Putting this together, the system dynamics in these new coordinates takes the form:

$$\begin{bmatrix} \dot{\boldsymbol{\eta}} \\ \dot{\mathbf{z}} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{F}\boldsymbol{\eta} \\ \boldsymbol{\omega}(\boldsymbol{\eta}, \mathbf{z}) \end{bmatrix}}_{\mathbf{f}(\zeta)} + \underbrace{\begin{bmatrix} \mathbf{G} \\ \mathbf{0} \end{bmatrix}}_{\mathbf{g}(\zeta)} \mathbf{u} \tag{2.20}$$

where $\boldsymbol{\zeta} \triangleq (\boldsymbol{\eta}, \mathbf{z}) \in \mathcal{X} \triangleq \mathcal{N} \times \mathcal{Z}$. We refer to this transformation as the *actuation decomposition*; it separates the system states into a set of directly actuated coordinates ($\boldsymbol{\eta}$), and a set of unactuated coordinates ($\mathbf{z}$).

Remark: *This choice of output can be generalized to other bijective functions of the actuated joints, such as task space coordinates, as long as they form a square set of outputs.*

Hybrid Underactuated Systems

Given our discussion of hybrid systems, and of underactuated systems, we can begin to analyze their combination. To this end, consider the hybrid dynamics in the underactuation decomposition as:

$$\mathscr{N} = \begin{cases} \dot{\boldsymbol{\eta}} = \hat{\mathbf{f}}(\boldsymbol{\eta}, \mathbf{z}) + \hat{\mathbf{g}}(\boldsymbol{\eta}, \mathbf{z})\mathbf{u} \\ \dot{\mathbf{z}} = \boldsymbol{\omega}(\boldsymbol{\eta}, \mathbf{z}) \end{cases} \quad \boldsymbol{\Phi}^{-1}(\boldsymbol{\eta}, \mathbf{z}) \notin \mathcal{S}$$
$$\begin{cases} \boldsymbol{\eta}^+ = \boldsymbol{\Delta}_{\boldsymbol{\eta}}(\boldsymbol{\eta}^-, \mathbf{z}^-) \\ \mathbf{z}^+ = \boldsymbol{\Delta}_{\mathbf{z}}(\boldsymbol{\eta}^-, \mathbf{z}^-) \end{cases} \quad \boldsymbol{\Phi}^{-1}(\boldsymbol{\eta}, \mathbf{z}) \in \mathcal{S}.$$

Gronwall Bellman Lemma

We list a comparison principle that will be useful for later analysis:

Lemma 2.39 (Gronwall Bellman [117])**.** *Let $I = [a, b] \subset \mathbb{R}$ be an interval, let $y : I \to \mathbb{R}$ and $\varphi : I \to \mathbb{R}_{\geq 0}$ be continuous and nonnegative functions, and let $\lambda : I \to \mathbb{R}_{\geq 0}$ be a continuous, positive, and non-decreasing function. If $y : I \to \mathbb{R}$ satisfies:*

$$y(t) \leq \lambda(t) + \int_a^t \varphi(s)y(s)ds, \ \textit{for all } t \in [a, b],$$

then it follows that:

$$y(t) \leq \lambda(t)e^{\int_a^t \varphi(s)ds}, \ \textit{for all } t \in [a, b].$$

2.5 Optimization

Optimization plays a pivotal role in constructing stabilizing controllers for robotic systems. It provides a principled and modular framework for embedding both task objectives and physical constraints into a unified formulation. At its core, optimization theory is concerned with finding points that minimize a cost function while satisfying constraints. The general constrained optimization problem can be written as:

$$\begin{aligned} p^* = \inf_{\mathbf{x}} \quad & c(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{h}_{\text{in}}(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{h}_{\text{eq}}(\mathbf{x}) = \mathbf{0} \end{aligned}$$

where $\mathbf{x} \in \mathcal{X}$ is the decision variable, $c : \mathcal{X} \to \mathbb{R}$ is the cost, $\mathbf{h}_{\text{in}} : \mathcal{X} \to \mathbb{R}^{n_{\text{in}}}$ are inequality constraints, and $\mathbf{h}_{\text{eq}} : \mathcal{X} \to \mathbb{R}^{n_{\text{eq}}}$ are equality constraints.

To analyze and solve such problems, we often study the Lagrangian function, which augments the cost with the constraints via dual variables:

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = c(\mathbf{x}) + \boldsymbol{\lambda}^\top \mathbf{h}_{\text{in}}(\mathbf{x}) + \boldsymbol{\mu}^\top \mathbf{h}_{\text{eq}}(\mathbf{x})$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{n_{\text{in}}}$ and $\boldsymbol{\mu} \in \mathbb{R}^{n_{\text{eq}}}$ are the *dual variables*, also known as the Lagrange multipliers for the inequality and equality constraints, respectively. The augmented form enables us to define the corresponding (unconstrained) dual problem:

$$d^* = \inf_{\mathbf{x}} \sup_{\boldsymbol{\lambda} \geq 0, \boldsymbol{\mu}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}).$$

The solution to the dual problem provides a lower bound on the solution to the original (primal) problem. That is, $d^* \leq p^*$, which is known as *weak duality*. There is a very important class of problems, namely convex problems, for which stronger statements can be made. To define this class, we introduce the notion of convexity:

Definition 2.40. A set $\mathcal{C} \subset \mathbb{R}^n$ is *convex* if for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ and all $\eta \in [0, 1]$, we have that $\eta \mathbf{x}_1 + (1 - \eta)\mathbf{x}_2 \in \mathcal{C}$.

Definition 2.41. A function $c : \mathcal{X} \to \mathbb{R}$ is convex if for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ and all $\eta \in [0, 1]$, we have that $c(\eta \mathbf{x}_1 + (1 - \eta)\mathbf{x}_2) \leq \eta c(\mathbf{x}_1) + (1 - \eta) c(\mathbf{x}_2)$.

A function being convex is equivalent to the requirement that the epigraph of the function — the set defined by state/value pairs that lie above the function evaluation — is a convex set. Convexity enables us many conveniences: any local minimum is also global, and simple gradient-based methods can often efficiently reach the solution.

A well known condition that guarantees *strong duality* for convex programs is Slater's condition:

Proposition 2.42 (Slater)**.** *If* c, $\mathbf{h}_{\text{in}}(\mathbf{x})$, *and* $\mathbf{h}_{\text{eq}}(\mathbf{x})$ *are convex functions, and there exists a point* $\mathbf{x}^*$ *that is strictly feasible, then strong duality holds, i.e.,* $d^* = p^*$.

Many optimization problems encountered in control theory can be formulated as a member of the following class of convex programs:

Linear Prog. (LP)	**Quadratic Prog. (QP)**	**Second-Order Cone Prog. (SOCP)**
$\inf_{\mathbf{x}} \quad \mathbf{c}^\top \mathbf{x}$	$\inf_{\mathbf{x}} \quad \mathbf{x}^\top \mathbf{P} \mathbf{x} + \mathbf{q}^\top \mathbf{x}$	$\inf_{\mathbf{x}} \quad \mathbf{f}^\top \mathbf{x}$
s.t. $\quad \mathbf{A}\mathbf{x} \leq \mathbf{b}$	s.t. $\quad \mathbf{G}\mathbf{x} \leq \mathbf{h}$	s.t. $\quad \|\mathbf{A}_i \mathbf{x} + \mathbf{b}_i\|_2 \leq \mathbf{c}_i^\top \mathbf{x} + d_i$
	$\quad \mathbf{A}\mathbf{x} = \mathbf{b}$	$\quad \mathbf{F}\mathbf{x} = \mathbf{g}$

For these optimization programs, many efficient numerical solvers exist. While optimization is a broad and rich field in its own right, our focus moving forward will be its application to control synthesis.

Optimal Control

We can specialize the original optimization problem to the field control theory by constructing the following optimal control problem:

$$V(\mathbf{x}_0) = \min_{\mathbf{u}(\cdot)} \quad \int_0^\infty c(\mathbf{x}(t), \mathbf{u}(t))dt \tag{OCP}$$
$$\text{s.t.} \quad \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u}, \;\; \mathbf{x}(0) = \mathbf{x}_0$$

where $c : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is a positive-definite cost function and $V : \mathcal{X} \to \mathbb{R}_{\geq 0}$ denotes the infinite horizon cost at the given initial condition. This formulation is extremely powerful: it naturally incorporates system dynamics, constraints, and control objectives into a single framework. Different philosophies on how to solve this problem in practice are are presented next.

Hamilton Jacobi Bellman

We begin with the Bellman principle of optimality, which states that the value function satisfies:

$$V(\mathbf{x}(t)) = \inf_{\mathbf{u}_{[t,t+dt]}} \left\{ \int_t^{t+dt} c(\mathbf{x}, \mathbf{u})d\tau + V(\mathbf{x}(t+dt)) \right\}.$$

As this is a recursive constraint, in order to get a condition at a specific point, consider a linear Taylor approximation of the value function:

$$V(\mathbf{x}(t+dt)) = V(\mathbf{x}(t)) + \frac{dV}{d\mathbf{x}}(\mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u})dt.$$

Plugging this into the value function equation yields:

$$V(\mathbf{x}(t)) = \inf_{\mathbf{u}_{[t,t+dt]}} \left\{ c(\mathbf{x}, \mathbf{u})dt + V(\mathbf{x}(t)) + \frac{dV}{d\mathbf{x}}(\mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u}^*)dt) \right\}.$$

Rearranging terms and taking the limit as $dt \to 0$ yields the famous Hamilton Jacobi Bellman (HJB) equation:

$$0 = \inf_{\mathbf{u}} \left\{ c(\mathbf{x}, \mathbf{u}) + \frac{dV}{d\mathbf{x}}(\mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u})) \right\}.$$

Linear Quadratic Regulator (LQR)

As a first example, consider the simplest case to the above problem: an unconstrained linear system with quadratic state and input cost, known as the Linear Quadratic Regulator (LQR) problem:

$$V(\mathbf{x}_0) = \min_{\mathbf{u}(\cdot)} \quad \int_0^\infty \mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{u}^\top \mathbf{R}\mathbf{u} d\tau \qquad \text{(LQR)}$$
$$\text{s.t.} \quad \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \;\; \mathbf{x}(0) = \mathbf{x}_0$$

where $\mathbf{Q} \in \mathbb{R}^{n\times n}$ and $\mathbf{R} \in \mathbb{R}^{m\times m}$ are positive-definite symmetric matrices. In this case, we can solve for not only the optimal control action $\mathbf{u}^*$ everywhere, but also the global value function $V : \mathcal{X} \to \mathbb{R}_{\geq 0}$. We begin by assuming that the value function takes the form $V(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}\mathbf{x}$. Plugging this into the HJB condition from earlier yields:

$$\begin{aligned} 0 &= \min_{\mathbf{u}} \left[\dot{V}(\mathbf{x}, \mathbf{u}) + \mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{u}^\top \mathbf{R}\mathbf{u}\right] \\ &= \min_{\mathbf{u}} \left[(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u})^\top \mathbf{P}\mathbf{x} + \mathbf{x}^\top \mathbf{P}(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}) + \mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{u}^\top \mathbf{R}\mathbf{u}\right] \\ &= \mathbf{x}^\top (\mathbf{A}^\top \mathbf{P} + \mathbf{P}\mathbf{A} + \mathbf{Q})\mathbf{x} + \min_{\mathbf{u}} \left[\mathbf{u}^\top \mathbf{B}^\top \mathbf{P}\mathbf{x} + \mathbf{x}^\top \mathbf{P}\mathbf{B}\mathbf{u} + \mathbf{u}^\top \mathbf{R}\mathbf{u}\right]. \end{aligned}$$

Solving for the minimum over $\mathbf{u}$ yields:

$$\mathbf{u} = -\underbrace{\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{P}}_{\triangleq \mathbf{K}} \mathbf{x},$$

where $\mathbf{K}$ is the optimal feedback gain matrix from LQR. Plugging this form back in for the minimizer over $\mathbf{u}$ yields:

$$0 = \mathbf{x}^\top \left(\mathbf{A}^\top \mathbf{P} + \mathbf{P}\mathbf{A} + \mathbf{Q} - 2\mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{P} + \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{P}\right) \mathbf{x}$$

which is achieved when $\mathbf{P}$ satisfies the well known Continuous Algebraic Riccati Equation (CARE):

$$\mathbf{A}^\top \mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{P} + \mathbf{Q} = \mathbf{0}.$$

Remark: *Beautiful! Not only do we get a globally optimal feedback policy, but we also get its associated value function. The solution to LQR is a refreshing respite from the tortuous and prickly world of general solutions to optimal control problems.*

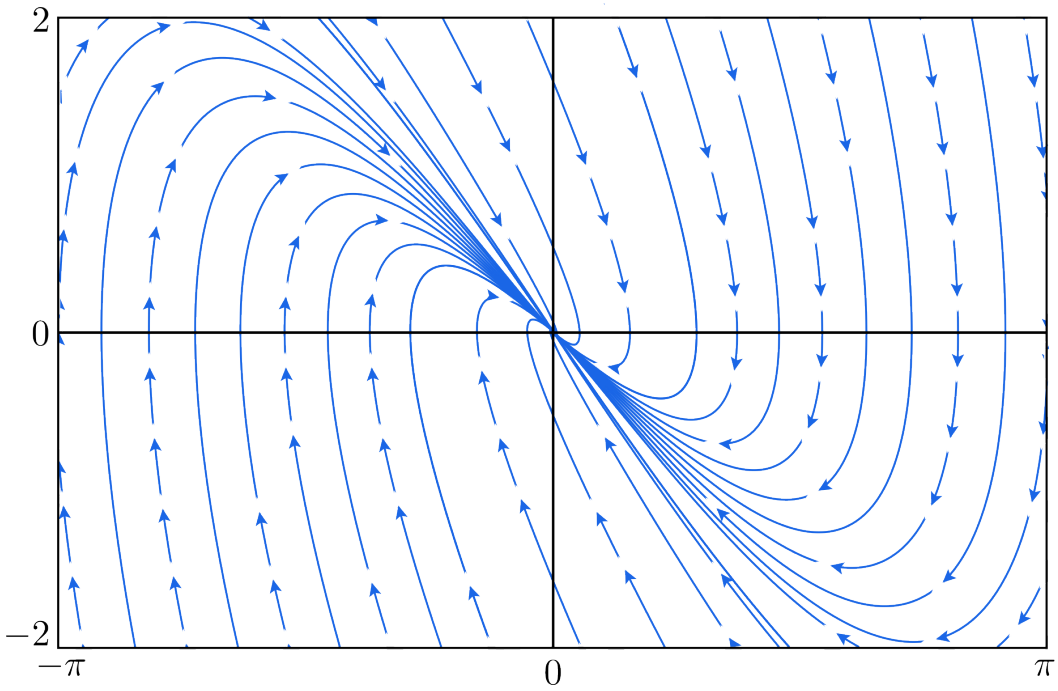


Figure 2.9: The phase portrait of the pendulum under the LQR feedback controller.

Finite-Horizon Optimal Control

Next, consider the optimization problem over a finite horizon of length T:

$$V(\mathbf{x}_0) = \min_{\mathbf{u}(\cdot)} \quad \int_0^T \mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{u}^\top \mathbf{R}\mathbf{u} d\tau + V(\mathbf{x}(T))$$
$$\text{s.t.} \quad \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{u}), \quad \mathbf{x}(0) = \mathbf{x}_0$$

where $V : \mathcal{X} \to \mathbb{R}_{\geq 0}$ is an estimate of the cost to go. If this is the true cost to go, then this is equivalent to the infinite horizon problem. When we add constraints, we can no longer find a closed form solution, and require turning to numerical optimization.

Indirect Methods (PMP):

We can try to directly apply the necessary conditions to the optimization problem. To this end, consider the Hamiltonian:

$$H(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) = c(\mathbf{x}, \mathbf{u}) + \boldsymbol{\lambda}^\top (\mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u}).$$

The Pontryagin principle of optimality yields the following condition:

$$\dot{\mathbf{x}} = \mathbf{H}_{\boldsymbol{\lambda}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u}^*$$
$$\dot{\boldsymbol{\lambda}} = -\mathbf{H}_{\mathbf{x}} = \frac{dc}{d\mathbf{x}}(\mathbf{x}) + \boldsymbol{\lambda}^\top \left(\frac{d\mathbf{f}}{d\mathbf{x}}(\mathbf{x}) + \frac{d\mathbf{g}}{d\mathbf{x}}(\mathbf{x})\mathbf{u} \right)$$

with the boundary conditions $\mathbf{x}(0) = \mathbf{x}_0$ and $\boldsymbol{\lambda}(T) = V(\mathbf{x}(T))$. One way of solving this two point boundary value ODE is to start in a neighborhood of the origin where the solution matches the LQR solution. This gives us values for $\mathbf{x}(T)$ and $\boldsymbol{\lambda}(T)$, whereby we can flow the ode backwards in time to produce an optimal trajectory. Unfortunately, using this method there is little way to shape the production of optimal controllers from arbitrary points in the state space.

Direct Single Shooting (DDP/iLQR):

We now turn our attention to *direct* methods, which turn the above infinite dimensional optimization program into a finite dimensional problem via discretization. In a similar vein, we would like to leverage the conditions of optimality, but produce a trajectory from an initial condition $\mathbf{x}_0$ in the state space. DDP follows a similar principle of updating trajectories under principles of optimality backwards in time. To begin, we assume that we have an initial condition $\mathbf{x}_0$ and a nominal control input $\mathbf{u}(\cdot)$ that converges us to a neighborhood of the origin.

As this is a direct method, we will first consider a numerical integration of the input trajectory, known as the forward pass, which produces a sequence of $\{\mathbf{x}_k\}$ and $\{\mathbf{u}_k\}$. It will be convenient to write the state-action value function as:

$$Q(\mathbf{x}_k, \mathbf{u}_k) = c(\mathbf{x}_k, \mathbf{u}_k) + V(\mathbf{x}_{k+1}).$$

Taking a second order approximation of this function yields:

$$\begin{aligned} Q(\mathbf{x}_k, \mathbf{u}_k) \cong Q(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k) + \begin{bmatrix} \mathbf{Q}_\mathbf{x} \\ \mathbf{Q}_\mathbf{u} \end{bmatrix}^\top \begin{bmatrix} \mathbf{x}_k - \bar{\mathbf{x}}_k \\ \mathbf{u}_k - \bar{\mathbf{u}}_k \end{bmatrix} \\ + \frac{1}{2} \begin{bmatrix} \mathbf{x}_k - \bar{\mathbf{x}}_k \\ \mathbf{u}_k - \bar{\mathbf{u}}_k \end{bmatrix}^\top \begin{bmatrix} \mathbf{Q}_{\mathbf{xx}} & \mathbf{Q}_{\mathbf{xu}} \\ \mathbf{Q}_{\mathbf{ux}} & \mathbf{Q}_{\mathbf{uu}} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k - \bar{\mathbf{x}}_k \\ \mathbf{u}_k - \bar{\mathbf{u}}_k \end{bmatrix}. \end{aligned}$$

As the state-action value function depends on future states, we store these terms backwards along the trajectory, termed the backward pass. Then, on the next forward pass, we can take a second order Taylor approximation of the state-action value function to get:

$$\mathbf{u}_k^* = \underset{\mathbf{u}_k}{\operatorname{argmin}}\, \mathbf{u}_k^\top \mathbf{Q}_{\mathbf{uu}} \mathbf{u}_k + (\mathbf{Q}_\mathbf{u} + \mathbf{Q}_{\mathbf{ux}}(\mathbf{x}_k - \bar{\mathbf{x}}_k))\mathbf{u}_k$$

where the constant terms can be dropped as they do not impact optimality. This yields a pointwise QP at each timestep of the next forward pass to produce updates $\delta\mathbf{u}_k$. This process of forwards and backwards passes is iterated until convergence.

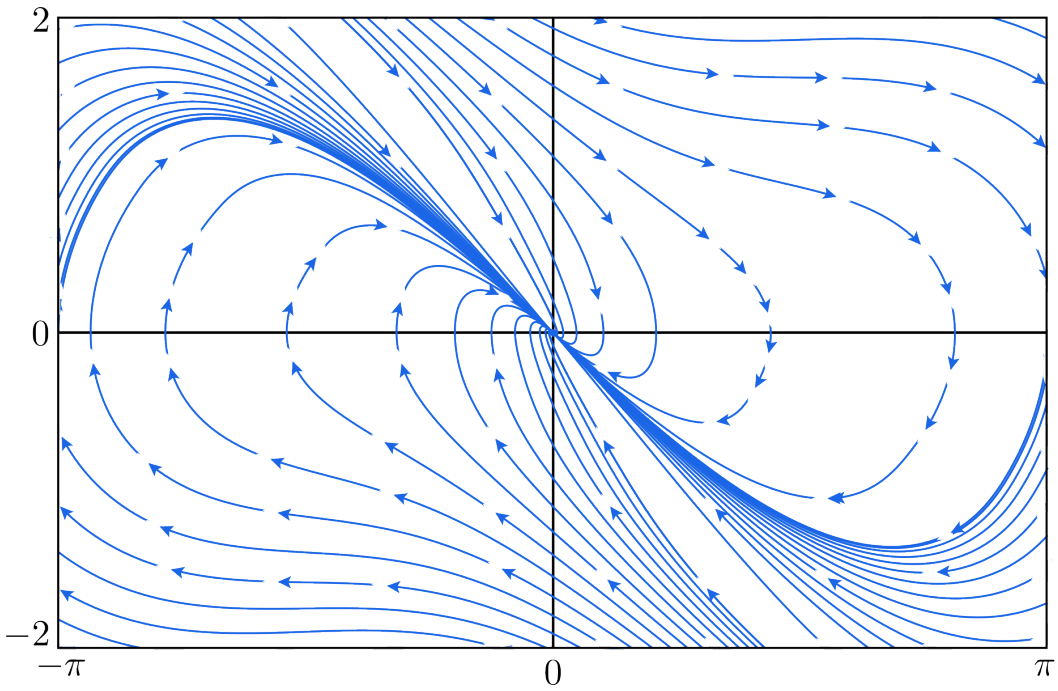


Figure 2.10: The phase portrait of the pendulum under an optimal controller with input bounds of $[-1.5\ 1.5]$.

If the Hessians of the dynamics are omitted from the above terms, this algorithm is known as iLQR.

The benefits of DDP and iLQR are only requiring small scale QPs to be solved in the backward pass, and an adaptive discretization scheme on the forward pass. Furthermore, (convex) input constraints are easily added to DDP/iLQR, as they simply change the constraints in the forward pass QP. However, the assumption of having a nominal controller that gets the state to a neighborhood of the origin is quite strong, and DDP/iLQR really only works for local optimization around a nominal trajectory.

Direct Multiple Shooting (SQP):

SQP lies on the other end of the spectrum, treating the optimal control problem as a general optimization problem and forgoing any notion of control theoretic necessary conditions for optimality. This method is again a direct method, which discretizes

the trajectory. We start with a linearization of the nonlinear dynamics at $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$:

$$\begin{aligned}\dot{\mathbf{x}} &\cong \mathbf{f}(\bar{\mathbf{x}}) + \mathbf{g}(\bar{\mathbf{x}})\bar{\mathbf{u}} + \underbrace{\left(\frac{d\mathbf{f}}{d\mathbf{x}}(\bar{\mathbf{x}}) + \frac{d\mathbf{g}}{d\mathbf{x}}(\bar{\mathbf{x}})(\bar{\mathbf{u}})\right)}_{\triangleq \mathbf{A}}(\mathbf{x} - \bar{\mathbf{x}}) + \underbrace{\mathbf{g}(\bar{\mathbf{x}})}_{\triangleq \mathbf{B}}(\mathbf{u} - \bar{\mathbf{u}}) \\ &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \underbrace{\mathbf{f}(\bar{\mathbf{x}}) + \mathbf{g}(\bar{\mathbf{x}})\bar{\mathbf{u}} - \mathbf{A}\bar{\mathbf{x}} - \mathbf{B}\bar{\mathbf{u}}}_{\triangleq \mathbf{C}}.\end{aligned}$$

If we assume that the control input is held constant over a discretization time $h > 0$, these can be discretized exactly using the matrix exponential from earlier by considering the augmented system:

$$\frac{d}{dt}\begin{bmatrix}\mathbf{x}\\ \mathbf{u}\\ 1\end{bmatrix} = \underbrace{\begin{bmatrix}\mathbf{A} & \mathbf{B} & \mathbf{C}\\ \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & 0\end{bmatrix}}_{\triangleq \mathbf{M}}\begin{bmatrix}\mathbf{x}\\ \mathbf{u}\\ 1\end{bmatrix} \xrightarrow{\text{Discretize}} \mathbf{x}_{k+1} = \begin{bmatrix}\mathbf{1} & \mathbf{0} & 0\end{bmatrix} e^{\mathbf{M}h}\begin{bmatrix}\mathbf{x}_k\\ \mathbf{u}_k\\ 1\end{bmatrix},$$

which gives us a linear discrete time system. Linearizing the cost and constraints in the same way results in a quadratic program that can quickly be solved:

$$\begin{aligned}\min_{\mathbf{x}_k, \mathbf{u}_k} \quad & \sum_{k=0}^{N} \mathbf{x}_k^\top \mathbf{Q}_k \mathbf{x}_k + \mathbf{N}_k^\top \mathbf{x}_k + \mathbf{u}_k^\top \mathbf{R}_k \mathbf{u}_k \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = \mathbf{A}_k \mathbf{x}_k + \mathbf{B}_k \mathbf{u}_k + \mathbf{C}_k, \;\; \mathbf{x}_0 = \mathbf{x}(0) \\ & \mathbf{x}_k \in \mathcal{X}, \;\; \mathbf{u}_k \in \mathcal{U}. \end{aligned} \tag{2.21}$$

This leads to large (but sparse) quadratic programs with many decision variables. One school of thought to avoid this large problem is to observe that the state variables are only a function of the initial condition and the control input sequence via:

$$\mathbf{x}_{k+1} = \mathbf{A}^k \mathbf{x}_0 + \sum_{j=0}^{k} \mathbf{A}^{k-j} \mathbf{B} \mathbf{u}_j.$$

This leads to an equivalent problem, but reduces the number of decision variables at the cost of creating a dense problem. Which one is better depends on the specific problem being solved, but keeping the states as decision variables preserves sparsity of the equality constraints (which can improve sparse solver speeds) and avoids high powers of potentially near singular matrices. These methods can also be mixed to produce banded density patters and keep the number of decision variables (relatively) low. Regardless, both perspectives yield a quadratic program, for which there are many open source solvers available.

In general, SQP is an extremely general and powerful method for solving optimal control problems. Because it optimizes over states and inputs directly, both state and input constraints can easily be added to the problem formulation.

Line Search

Once a state and input search direction $\delta\mathbf{x}_k$ and $\delta\mathbf{u}_k$ have been chosen, a line search finds the best step length in those directions to take. Choosing the correct step size is an important feature of a numerical optimizer as this choice can significantly effect the convergence of an iterative algorithm like DDP or LQR. Primarily, we would like to ensure that a step size α result in a decrease in the integrated cost function:

$$J(\mathbf{x}_k + \alpha\delta\mathbf{x}_k, \mathbf{u}_k + \alpha\delta\mathbf{u}_k) \leq J(\mathbf{x}_k, \mathbf{u}_k) + c\alpha(\nabla_{\mathbf{x}} J(\mathbf{x}_k, \mathbf{u}_k)^\top \delta\mathbf{x}_k + \nabla_{\mathbf{u}} J(\mathbf{x}_k, \mathbf{u}_k)^\top \delta\mathbf{u}_k)$$

for some constant $c \in (0, 1)$ [118]. This is often searched for using backtracing, which decreases the step size until this condition is satisfied. Many other methods of line search exist — from considering curvature conditions to leveraging second order information to select the optimal step size. When using these tools for real time optimization like in MPC, the current mindset however is to take full steps ($\alpha = 1$), only run 1 SQP/DDP iteration, and hope that the algorithm converges in time. This is motivated by the idea that the system is evolving in time as computation is taking place, so spending too long finding the optimal solution will result in the trajectory being stale and no longer useful.

2.6 Bézier Curves

Bézier curves offer a convenient basis for parameterizing solutions to optimal control problems, dynamical systems, and desired trajectories. A curve $\mathbf{b} : I \triangleq [0, \tau] \to \mathbb{R}^m$ for $\tau > 0$ is said to be a Bézier curve [119] of order $p \in \mathbb{N}$ if it is of the form:

$$\mathbf{b}(t) = \mathbf{p}\mathbf{z}(t),$$

where $\mathbf{z} : I \to \mathbb{R}^{p+1}$ is a Bernstein basis polynomial of degree p and $\mathbf{p} \in \mathbb{R}^{m \times p+1}$ is a matrix whose columns are the $p + 1$ *control points* of dimension m. There exists a matrix $\mathbf{H} \in \mathbb{R}^{p+1 \times p+1}$ (as in [79]) which defines a linear relationship between control points of a curve $\mathbf{b}$ and its derivative via:

$$\dot{\mathbf{b}}(t) = \mathbf{p}\mathbf{H}\mathbf{z}(t).$$

This enables us to define a state space curve $\mathbf{B} : I \to \mathbb{R}^n$:

$$\mathbf{B}(t) \triangleq \begin{bmatrix} \mathbf{b}(t) \\ \vdots \\ \mathbf{b}^{(\gamma-1)}(t) \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{p} \\ \vdots \\ \mathbf{p}\mathbf{H}^{\gamma-1} \end{bmatrix}}_{\triangleq \mathbf{P}} \mathbf{z}(t). \tag{2.22}$$

The columns of the matrix $\mathbf{P} \in \mathbb{R}^{n \times p+1}$, denoted as $\mathbf{P}_j$ for $j = 0, \ldots, p$, represent the collection of n dimensional control points of the Bézier curve $\mathbf{B}$ in the state space.

Bézier curves enjoy a number of desirable properties:

Property 2.43 (Convex Hull [119])**.**

$$\mathbf{B}(t) \in \text{conv}(\{\mathbf{P}_j\}), \;\; j = 0, \ldots, p, \;\; \forall t \in I.$$

Remark: *Herein lies the beauty of Bézier curves — the convex hull property allows us to make conclusions about the behavior of continuous time curves while only reasoning about the discrete collection of control points.*

Furthermore, we have that if the control points satisfy linear inequalities, then so does the continuous-time curve:

Property 2.44 (Linear Bounding [120])**.** For a vector $\mathbf{d} \in \mathbb{R}^k$ and a matrix $\mathbf{C} \in \mathbb{R}^{k \times n}$, we have:

$$\mathbf{C}\mathbf{P}_j \leq \mathbf{d}, \;\; j = 0, \ldots, p \implies \mathbf{C}\mathbf{B}(t) \leq \mathbf{d}, \;\; \forall t \in I.$$

This property will allow us to express convex constraints along the entire curve as a finite collection of convex constraints on the control points.

We will specifically be interested in producing Bézier curves that connect initial conditions $\mathbf{x}_d(0) \in \mathcal{X}_d$ and terminal conditions $\mathbf{x}_d(\tau) \in \mathcal{X}_d$ in a fixed time τ. Given such boundary conditions, a Bézier curve $\mathbf{B}(\cdot)$ which connects them must satisfy the following set of equality constraints:

$$\mathbf{b}^{(k)}(0) = \mathbf{p}\mathbf{H}^k\mathbf{z}(0) = \mathbf{q}_d^{(k)}(0), \;\; k = 0, \ldots, \gamma - 1, \tag{2.23}$$

$$\mathbf{b}^{(k)}(\tau) = \mathbf{p}\mathbf{H}^k\mathbf{z}(\tau) = \mathbf{q}_d^{(k)}(\tau), \;\; k = 0, \ldots, \gamma - 1. \tag{2.24}$$

Note that $\mathbf{z}(0) = [1 \; \mathbf{0}_{1\times p}]^\top$ and $\mathbf{z}(\tau) = [\mathbf{0}_{1\times p} \; 1]^\top$. Then, collecting the constraints in (2.23) and (2.24) yields:

$$\mathbf{p}\begin{bmatrix}\mathbf{H}_0^0 & \mathbf{H}_0^1 & \ldots & \mathbf{H}_0^{\gamma-1}\end{bmatrix} = \mathbf{x}_0,$$

$$\mathbf{p}\begin{bmatrix}\mathbf{H}_p^0 & \mathbf{H}_p^1 & \ldots & \mathbf{H}_p^{\gamma-1}\end{bmatrix} = \mathbf{x}_T,$$

where $\mathbf{H}_j^i$ denotes the j^{th} column of the matrix $\mathbf{H}$ raised to the i^{th} power. It can be algebraically verified that $\mathbf{H}$ has the form:

$$\mathbf{H}_0^i = \Big[\underbrace{\star \; \cdots \; \star}_{i+1} \; \underbrace{0 \; \cdots \; 0}_{p-i}\Big]^\top, \;\; \mathbf{H}_p^i = \Big[\underbrace{0 \; \cdots \; 0}_{p-i} \; \underbrace{\star \; \cdots \; \star}_{i+1}\Big]^\top,$$

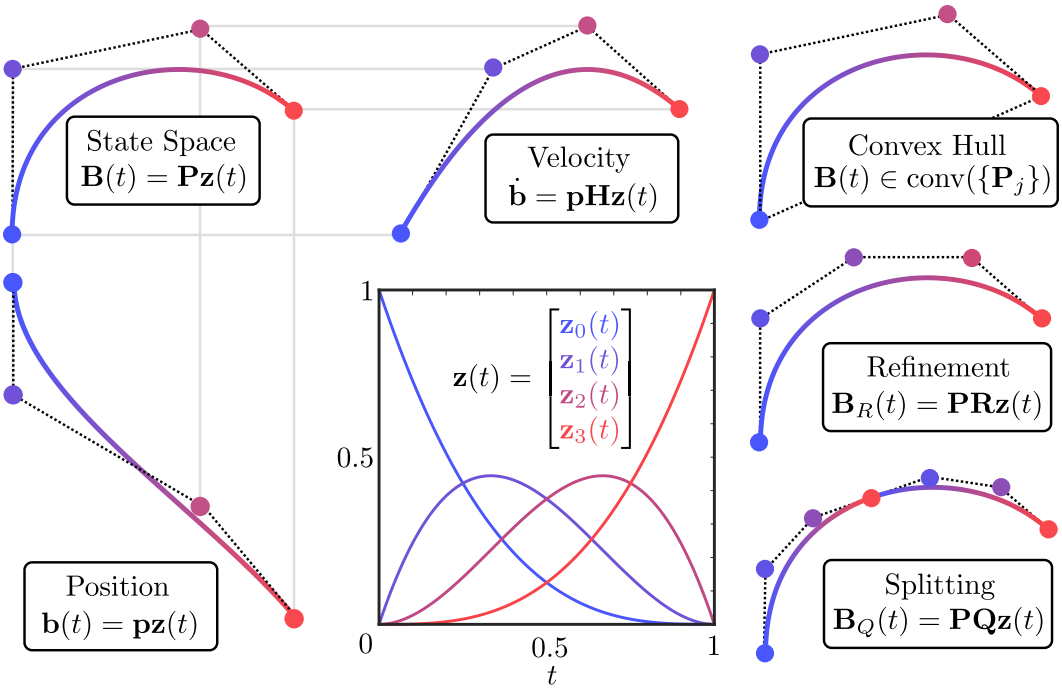


Figure 2.11: A visual guide to the properties of Bézier curves.

with nonzero entries $\star$. Taking $\mathbf{D} \in \mathbb{R}^{p+1\times 2n}$ as:

$$\mathbf{D} \triangleq \begin{bmatrix} \mathbf{H}_0^0 & \mathbf{H}_0^1 & \dots & \mathbf{H}_0^{\gamma-1} & \mathbf{H}_p^0 & \mathbf{H}_p^1 & \dots & \mathbf{H}_p^{\gamma-1} \end{bmatrix},$$

in the case that $p \geq 2\gamma - 1$ the columns are linearly independent and thus the matrix $\mathbf{D}$ has full column rank, implying that a solution $\mathbf{p}$ exists (but is not unique unless $p = 2\gamma - 1$). In the case that $p > 2\gamma - 1$, the constraint is under-determined and can be resolved via a least squares solution, allowing for additional cost terms to be optimized.

This leads to the following property:

Property 2.45 (Boundary Values [120])**.** Given a time $\tau > 0$, two points $\mathbf{x}_0, \mathbf{x}_\tau \in \mathbb{R}^n$, and order $p \geq 2\gamma - 1$, there exists a matrix $\mathbf{D} \in \mathbb{R}^{p+1\times 2n}$ such that any curve $\mathbf{x}_d(\cdot)$ with control points satisfying:

$$\mathbf{p}\mathbf{D} = \begin{bmatrix} \mathbf{x}_0^\top & \mathbf{x}_\tau^\top \end{bmatrix} \tag{2.25}$$

also satisfies $\mathbf{x}_d(0) = \mathbf{x}_0$ and $\mathbf{x}_d(\tau) = \mathbf{x}_\tau$.

The next property will be useful in increasing the resolution of Bézier curves and reduce the conservatism of their upper bounds. To do this, we introduce the notion of a refinement of the interval I as:

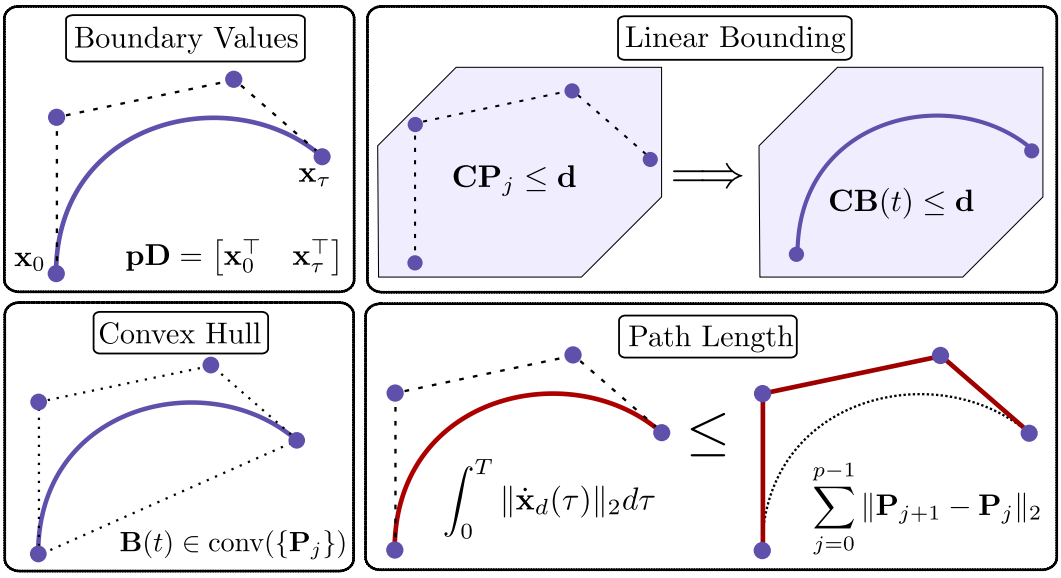


Figure 2.12: A collection of some useful properties of Bézier curves.

Definition 2.46. A *k-refinement* of an interval $[0, T]$ is a collection of times $\{\tau_i\}$ for $i = 0, \ldots, k$ and associated intervals $\{[\tau_{i-1}, \tau_i]\}$ with $\tau_{i-1} < \tau_i$, $\tau_0 = 0$, and $\tau_k = T$.

From this, we can split a Bézier polynomial $\mathbf{B}(\cdot)$ into a sequence of B-splines:

Property 2.47 (Splitting [119])**.** Given the control points $\mathbf{P}$ of a Bézier polynomial defined over the interval I and a k-refinement of I, there exists a collection of matrices $\{\mathbf{Q}_i\}$ for $i = 1, \ldots, k$ such that $\mathbf{B}_Q(t) = \mathbf{PQz}(t)$ satisfies $\mathbf{B}_Q(t) \triangleq \mathbf{B}(\tau_i + \frac{t}{T}(\tau_{i+1} - \tau_i))$ for all $t \in I$.

Furthermore, given a curve of a specific order, we can increase the resolution of the curve via:

Property 2.48 (Refinement [119])**.** Given control points $\mathbf{P}$ of an order p Bézier polynomial and a desired order $r > p$, there exists a matrix $\mathbf{R} \in \mathbb{R}^{p+1 \times r+1}$ such that $\mathbf{B}_R(t) = \mathbf{PRz}'(t)$ for $\mathbf{z}' : I \to \mathbb{R}^{r+1}$ satisfies $\mathbf{B}_R(t) = \mathbf{B}(t)$ for all $t \in I$.

Another property allows us to upper bound the path length of the curve via a function of the control points:

Property 2.49 (Path Length [121])**.** The path length of a Bézier curve $\mathbf{x}_d(\cdot)$ is upper bounded by the norm distance of its control points:

$$\int_0^T \|\dot{\mathbf{x}}_d(\tau)\|_2 d\tau \leq \sum_{j=0}^{p-1} \|\mathbf{P}_{j+1} - \mathbf{P}_j\|_2.$$

This upper bound is particularly useful for convex relaxations of path-length cost functions, since it is expressed as a sum of norms over discrete control points.

Finally, it will be useful to operate with the (column-wise) vectorized versions of $\mathbf{p}$ and $\mathbf{P}$, defined as $\vec{\mathbf{p}} \triangleq \text{vec}(\mathbf{p}) \in \mathbb{R}^{m(p+1)}$ and $\vec{\mathbf{P}} \triangleq \text{vec}(\mathbf{P}) \in \mathbb{R}^{n(p+1)}$. With these new representations, we have the following equivalences:

$$\begin{aligned} \vec{\mathbf{P}} &= \vec{\mathbf{H}}\vec{\mathbf{p}} \\ \vec{\mathbf{D}}\vec{\mathbf{p}} &= \text{vec}\left(\begin{bmatrix} \mathbf{x}_0^\top & \mathbf{x}_T^\top \end{bmatrix}\right), \end{aligned}$$

with $\vec{\mathbf{H}}$ and $\vec{\mathbf{D}}$ the vectorized versions of $\mathbf{H}$ and $\mathbf{D}$, respectively.

Chapter 3: Philosophy of Layered Architectures

Contents

Layered architectures naturally balance speed and flexibility trade-offs.

3.1 Problem Description

To motivate the use of layered architectures for the control of complex robotic systems, consider the nonlinear system:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{3.1}$$

with state $\mathbf{x} \in \mathbb{R}^n$, input $\mathbf{u} \in \mathbb{R}^m$, and dynamics $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ assumed to be continuously differentiable on their domain. Towards the deployment of autonomous systems, we would like to control (3.1) using high-level semantic commands such as "get me an apple," or "put away the dishes." Achieving such tasks presents two key challenges: understanding *what* the desired environment state should be and *how* the robot can achieve it [122].

Recent advances in large language models (e.g., ChatGPT) provide a means of addressing the first challenge by mapping abstract semantic tasks into goal states and environment representations:

$$\text{"Complete Task A"} \xrightarrow{\text{Language Model}} \mathbf{x}_G, \mathcal{X}$$

where $\mathbf{x}_G \in \mathbb{R}^n$ is a desired goal state and $\mathcal{X} \subset \mathbb{R}^n$ is a state constraint set, such as the free space in a cluttered environment. In general, this mapping may produce a sequence of intermediate goal states and constraints, but without loss of generality we will focus on a single instance.

Given this mapping, the remaining challenge is *how* to drive the system to the goal state while satisfying state constraints $\mathcal{X}$ and input constraints $\mathcal{U} \subset \mathbb{R}^m$. To formalize this, let $\mathbf{k} : \mathbb{R}^n \to \mathbb{R}^m$ be a controller mapping states to control actions. Given any initial condition $\mathbf{x}_0 \in \mathcal{X}$, applying $\mathbf{k}$ results in a continuously differentiable closed-loop trajectory $\mathbf{x}_{\text{cl}} : I \to \mathcal{X}$ over some interval $I \subset \mathbb{R}_{\geq 0}$:

$$\mathbf{x}_{\text{cl}}(t) \equiv \boldsymbol{\varphi}(t, \mathbf{x}_0; \mathbf{k}) \triangleq \mathbf{x}_0 + \int_0^t \mathbf{f}(\mathbf{x}_{\text{cl}}(\tau), \mathbf{k}(\mathbf{x}_{\text{cl}}(\tau)))d\tau.$$

Our goal is to design a controller that drives the system to the desired goal state while satisfying constraints, as summarized by the following problem statement:

Problem 3.1. Given an initial state $\mathbf{x}_0 \in \mathcal{X}$, a goal state $\mathbf{x}_G \in \mathcal{X}$, constraint sets $\mathcal{X} \subset \mathbb{R}^n$ and $\mathcal{U} \subset \mathbb{R}^m$, a horizon $T > 0$, and a tolerance $\varepsilon > 0$, design a controller $\mathbf{k} : \mathbb{R}^n \to \mathbb{R}^m$ such that the closed-loop trajectory $\mathbf{x}_{\text{cl}}$ of system (3.1) satisfies:

- $\|\mathbf{x}_{\text{cl}}(T) - \mathbf{x}_G\| \leq \varepsilon$

- $\mathbf{x}_{cl}(t) \in \mathcal{X} \subset \mathbb{R}^n$ for all $t \in [0, T]$
- $\mathbf{k}(\mathbf{x}_{cl}(t)) \in \mathcal{U} \subset \mathbb{R}^m$ for all $t \in [0, T]$.

Optimal control provides a constructive approach for solving Problem 3.1 by embedding task objectives and constraints into a single framework via the following optimization problem:

$$\begin{aligned} V(\mathbf{x}_0) = \min_{\mathbf{u}(\cdot)} \quad & \int_0^\infty c(\mathbf{x}(t), \mathbf{u}(t))dt \\ \text{s.t.} \quad & \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \;\; \mathbf{x}(0) = \mathbf{x}_0 \\ & \mathbf{x}(t) \in \mathcal{X}, \quad \mathbf{u}(t) \in \mathcal{U} \end{aligned} \tag{OCP}$$

where $c : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is a positive-definite cost function and $V : \mathcal{X} \to \mathbb{R}_+$ denotes the infinite horizon cost at that point. This formulation naturally incorporates constraints while providing a principled means of defining control objectives through the choice of the cost function. We will investigate solution methods for (OCP), discuss the associated trade-offs between optimality, feasibility, and computational efficiency, and demonstrate that layered architectures provide an effective means to balance these trade-offs.

Directly solving this infinite-dimensional, nonconvex optimization problem via monolithic policy design is intractable in practice. Instead, we split the problem into two *levels* of abstraction:

High Level	**Low Level**
$\min_{\mathbf{u}(\cdot)} \quad \sum_{k=0}^{\infty} C(\mathbf{x}_k, \mathbf{u}_k)$	$\min_{\mathbf{u}(\cdot)} \quad \int_0^\infty c(\mathbf{x}(t), \mathbf{u}(t))dt$
s.t. $\quad \mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k, \mathbf{u}_k)$	s.t. $\quad \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$
$\mathbf{x}_k \in \mathcal{X}$	$\mathbf{u}(t) \in \mathcal{U},$

where the high-level controller focuses on planning over a (discrete) state model $\mathbf{F}$ to navigate the nonconvex constraint space, and the low-level controller focuses on stabilizing any dynamic instabilities of the continuous-time dynamics $\mathbf{f}$ while satisfying the input constraints. Even within each level, however, we still face challenging optimal control problems. To address this, we further decompose each level into two *layers*: a planning layer that generates feasible trajectories or waypoints, and a tracking layer that stabilizes the system along the plan. By clearly distinguishing *levels* (high and low) as operating with different timescales

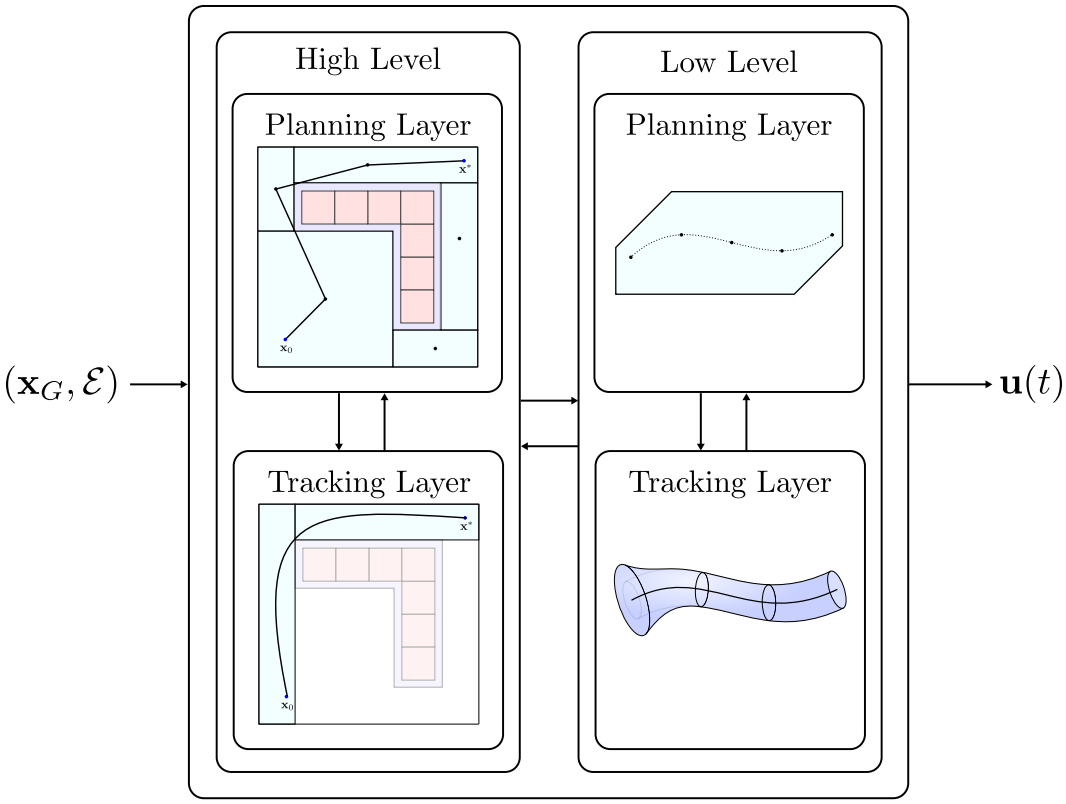


Figure 3.1: A solution to the abstract problem of "how" to drive systems to goal states that leverages layered control architectures. This will serve as an outline for the remaining sections of the thesis.

and system abstractions, and *layers* (planning and tracking) as feedforward and feedback components within each level, we organize the overall control problem into four interacting subproblems, as seen in Figure 5.14.

3.2 The Pitfalls of Purely Tracking Layers

The next sections will focus on the low level control problem, and motivate the further splitting of this problem into planning and tracking layers. Solving (OCP) *globally* over the domain $\mathcal{X}$ would yield a positive definite function $V : \mathcal{X} \to \mathbb{R}_+$ which satisfies the Hamilton-Jacobi-Bellman PDE. Such a solution is appealing — from this function, a controller which achieves Problem 3.1 is directly given by:

$$\mathbf{k}(\mathbf{x}) = \underset{\mathbf{u}}{\operatorname{arginf}} \frac{dV}{d\mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{u}) + c(\mathbf{x}, \mathbf{u}). \tag{3.2}$$

Solving the HJB equation to determine V, however, is generally regarded as intractable for high-dimensional nonlinear systems [123]. Therefore, alternative approaches approximate global solutions, such as Reinforcement Learning (RL), which

optimizes objectives stochastically through exhaustive simulation rollouts:

$$\min_{\mathbf{k}} \underset{\substack{\mathbf{u}(\cdot)\sim\mathbf{k}\\ \mathbf{x}_0\sim\mathcal{X}}}{\mathbb{E}} \quad \int_0^\infty c(\mathbf{x}(t),\mathbf{u}(t))dt$$
$$\text{s.t.} \quad \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u}), \;\; \mathbf{x}(0) = \mathbf{x}_0,$$

the solution of which would similarly produce a global feedback controller. Although RL has recently shown significant promise towards solving challenging robotic tasks [124]–[126], it remains highly data-intensive, often requiring millions of simulation samples, struggles with long-horizon tasks, and is often unable to generalize sufficiently beyond the training distribution. Moreover, these methods do not inherently guarantee constraint satisfaction, requiring additional mechanisms such as reward shaping or safety layers.

An alternative tracking layer approach is to decompose the goal tracking problem and the constraint satisfaction problem into separate entities within a real-time optimization framework. In this setting, Control Lyapunov functions (CLFs) can be used to ensure convergence to the goal, and Control Barrier Functions (CBFs) can be used to enforce state constraints, leading to the following CLF-CBF optimization program:

$$\begin{aligned} \min_{\mathbf{u}} \quad & \mathbf{u}^\top \mathbf{u} \qquad \text{(CLF-CBF)} \\ \text{s.t.} \quad & \dot{V}(\mathbf{x},\mathbf{u}) \leq -\gamma V(\mathbf{x}) \\ & \dot{h}(\mathbf{x},\mathbf{u}) \geq -\alpha h(\mathbf{x}). \end{aligned}$$

Splitting the problem up enables greater design flexibility, but such a formulation relies on finding valid functions V and h. Although these can be readily produced in certain scenarios, finding such functions in general often results in solving similarly complex problems to the HJB PDE [127]. Furthermore, if these functions are not compatible, such a program can often result in infeasibilities or local minima.

Regardless of the method, global solutions produced *a priori* suffer from a key limitation — rigidity. Any change in constraints, goals, or system dynamics requires re-solving the optimization problem from scratch, which is antithetical to the goal of real-time adaptability. Furthermore, solving (OCP) once for a specific system and environment offers little insight into how solutions generalize to new scenarios.

3.3 The Pitfalls of Purely Planning Layers

Given the complexity and rigidity of solving (OCP) globally, we next investigate the use of *local* methods for controller synthesis. Instead of solving for a global feedback

policy, we consider trajectory optimization methods that generate state and input control curves $\mathbf{x}_d : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ and $\mathbf{u}_d : \mathbb{R}_{\geq 0} \to \mathbb{R}^m$, respectively, for individual initial conditions $\mathbf{x}_0$. This shift from global to local solutions transforms the PDE in (3.2) into an ODE via the method of characteristics, which can be efficiently solved using numerical optimization tools. The simplest control strategy we could consider to solve Problem 3.1 follows as a feedforward application of the computed trajectory:

$$\mathbf{k}(\mathbf{x}, t) = \mathbf{u}_d(t). \tag{3.3}$$

This controller, however, is fragile — any model mismatch, external disturbance, or numerical imprecision will cause deviations from the planned trajectory. To analyze this issue, we make the following assumption about solve accuracy and time:

Assumption 3.2. Each solve of (OCP) requires a nonzero computation time $\tau > 0$ and will have numerical tolerance $\epsilon > 0$, meaning the implied state trajectory $\mathbf{x}_d(t)$ will only approximately satisfy the dynamics:

$$\|\dot{\mathbf{x}}_d(t) - \mathbf{f}(\mathbf{x}_d(t), \mathbf{u}_d(t))\| \leq \epsilon.$$

This error term captures the combined effects of numerical errors, solver inaccuracies, and model mismatches. In order to analyze the efficacy of the controller in (3.3), we define the state tracking error as $\mathbf{e}(t) = \mathbf{x}(t) - \mathbf{x}_d(t)$. We can bound the associated evolution of the error signal via:

$$\begin{aligned}
\|\mathbf{e}(t)\| &= \left\|\mathbf{e}(0) + \int_0^t \dot{\mathbf{e}}(\tau) d\tau\right\| \\
&\leq \|\mathbf{e}(0)\| + \int_0^t \|\mathbf{f}(\mathbf{x}, \mathbf{u}) - \dot{\mathbf{x}}_d\| d\tau \\
&\leq \|\mathbf{e}(0)\| + \int_0^t \epsilon + \|\mathbf{f}(\mathbf{x}, \mathbf{u}) - \mathbf{f}(\mathbf{x}_d, \mathbf{u}_d)\| d\tau \\
&= \|\mathbf{e}(0)\| + t\epsilon + \int_0^t L_f \|\mathbf{e}(\tau)\| d\tau.
\end{aligned}$$

Applying the Gronwall-Bellman Lemma 2.39, we obtain an upper bound on the deviation at the solver interval $t \in [0, T]$:

$$\|\mathbf{e}(t)\| \leq (\|\mathbf{e}(0)\| + t\epsilon) e^{tL_f} \triangleq \rho(t).$$

This bound reveals that even under ideal conditions and solving as quickly as possible, tracking errors grow exponentially over time without the introduction of a

feedback term. As a result, $\mathbf{x}_{cl}(t)$ deviates from the planned trajectory, leading to potential constraint violations or instability.

One way to mitigate this accumulated error is to ensure that the desired trajectory $\mathbf{x}_d$ is open-loop robust to disturbances. That is, we can require all possible states the system could reach under the worst-case tracking error to remain within the state and input constraints. This perspective, a variant of tube MPC [128], leads to a robust trajectory optimization problem:

$$\begin{aligned} \min_{\mathbf{u}(\cdot)} \quad & \int_0^\infty c(\mathbf{x}(t), \mathbf{u}(t))dt \\ \text{s.t.} \quad & \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \;\; \mathbf{x}(0) = \mathbf{x}_0 \\ & \mathbf{x}(t) \in \mathcal{X} \ominus B_{\rho(t)}(\mathbf{0}), \;\; \mathbf{u}(t) \in \mathcal{U} \ominus \mathbf{k}(B_{\rho(t)}(\mathbf{0})) \end{aligned}$$

where $\mathbf{k}(B_{\rho(t)}(\mathbf{0}))$ denotes all possible control actions taken over the set $B_{\rho(t)}(\mathbf{0})$, and $\ominus$ represents the Pontryagin difference. However, such a controller will result in overly conservative behavior [128] as the planner is always considering the worst case disturbance, and less conservative estimates of the disturbance reachable sets are generally intractable for nonlinear systems [129].

A natural conclusion from this development is that we must introduce a notion of feedback in a neighborhood of the nominal trajectory $\mathbf{x}_d$. To this end, we can modify the plan to compute a disturbance-feedback policy via:

$$\begin{aligned} \min_{\mathbf{k}} \max_{\mathbf{w} \in \mathcal{W}} \quad & \int_0^\infty c(\mathbf{x}(t), \mathbf{k}(\mathbf{x}(t))dt \\ \text{s.t.} \quad & \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{k}(\mathbf{x})) + \mathbf{w}, \;\; \mathbf{x}(0) = \mathbf{x}_0 \\ & \mathbf{x}(t) \in \mathcal{X}, \;\; \mathbf{k}(\mathbf{x}(t)) \in \mathcal{U} \end{aligned}$$

where $\mathcal{W} \subset \mathbb{R}^n$ is a differential form of the disturbance reachable set $B_{\rho(t)}(0)$. Although this formulation accounts for the disturbances induced by tracking error, it inherits the spatial complexity of the HJB equation and therefore significantly increases the problem complexity. Even for discrete-time linear systems with polytopic sets $\mathcal{W}$, such min-max formulations which optimize a control law over all possible state–disturbance pairs have been shown to be an NP hard problem [130], making it impractical for real-time application. The challenges with both global and local solutions to (OCP) suggest the need for a control architecture that allows for planning efficiently with real-time adaptability. In the next section, we introduce hierarchical control as a structured solution to balancing this trade-off.

3.4 Hierarchical Solutions

In light of the challenges demonstrated in the previous two sections, we have shown that neither feedforward nor feedback can solely provide satisfactory solutions to Problem 3.1. These observations point towards an inherent tension in control design: on one hand, we seek responsiveness and robustness to disturbance, and on the other, we desire long-horizon reasoning and satisfaction of global constraints. Attempting to encode both planning and feedback into a single monolithic policy quickly leads to high complexity and reduced tractability. A natural question then arises:

> **Question:** *Is it necessary to solve all aspects of the control problem at once, or are there benefits to decomposing it?*

To this end, consider the following combined feedforward and feedback controller:

$$\mathbf{k}(\mathbf{x}, t) = \mathbf{u}_d(t) + \mathbf{k}_{\text{fb}}(\mathbf{x}(t), \mathbf{x}_d(t)).$$

Here, $\mathbf{u}_d(t)$ is a feedforward input generated by solving a trajectory optimization problem, while $\mathbf{k}_{\text{fb}}$ is a local feedback controller tasked with rejecting disturbances and correcting deviations from the nominal state $\mathbf{x}_d$. Before proceeding, we need a way to coordinate the interaction between the planning and tracking layer. To achieve this, we formalize a key assumption about the interaction between the planning layer and the tracking layer: the tracking error remains within a bounded tracking invariant set, i.e., $\mathbf{x} - \mathbf{x}_d(t) \in \mathcal{E} \subset \mathbb{R}^n$. Although this assumption may initially seem strong, we will provide theoretical and empirical evidence supporting its existence in the subsequent sections. With this, we can now take a fixed tube MPC style approach [131], which modifies the desired trajectory to account for the tracking error while maintaining the desired guarantees:

$$\begin{aligned} \min_{\mathbf{u}(\cdot)} \quad & \int_0^\infty c(\mathbf{x}(t), \mathbf{u}(t))dt \\ \text{s.t.} \quad & \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \;\; \mathbf{x}(0) = \mathbf{x}_0 \oplus \mathcal{E} \\ & \mathbf{x}(t) \in \mathcal{X} \ominus \mathcal{E}, \;\; \mathbf{u}(t) \in \mathcal{U} \ominus \mathbf{k}(\mathcal{E}) \end{aligned}$$

where $\oplus$ represents the Mikowski sum.

This formulation immediately yields several important benefits. First, because the tracking layer performs disturbance rejection, the planning layer optimization problem is able to plan on the deterministic system. Unlike other planning layer solutions, this preserves computational efficiency while maintaining robustness to disturbances. Second, because constraints can be directly enforced at the planning

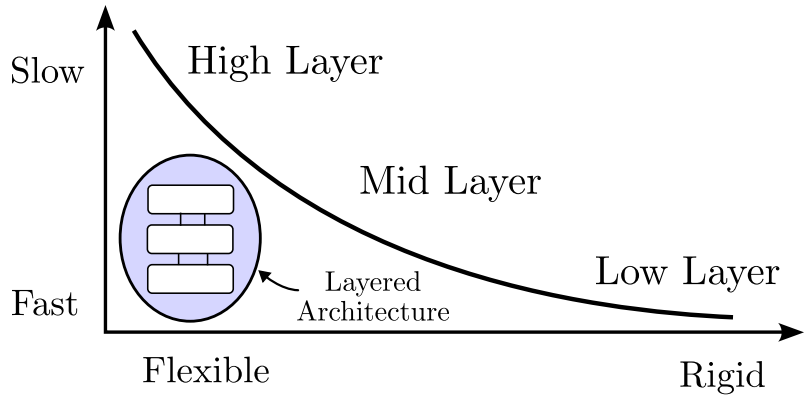


Figure 3.2: The Pareto front of controller design, which trades off speed and flexibility (a modernization of the trade-off discussed in [132]). Combining individual blocks into a hierarchy enables fast and flexible solutions to the control task.

stage, the resulting controller is guaranteed to be feasible. Third, this architecture is agnostic to the specific model implementation used at each layer. Therefore, any model, objective, time horizon, or solution method can be used at each layer (provided that the interface between the layers remains consistent) allowing for generalizable controller design and deployment. These benefits directly arise from coordinating layers through a carefully designed interface, exemplifying how a layered architecture achieves greater performance than its constituent parts individually, as illustrated in Figure 3.2. In summary, the layered control architecture offers three fundamental advantages:

1. **Efficiency**: Responsibility allocation across levels and layers of the architecture enables real-time control, even for complex systems and tasks.

2. **Feasibility**: Properly defined interfaces between levels and layers enable individual guarantees to compose into system-level guarantees.

3. **Generalizability**: Modularity enables component reuse across diverse tasks, platforms, and environments.

Although there is an elegance in this splitting and coordination, it is important to note that splitting up the problem is not without cost. Planning robust solutions is necessarily suboptimal as it strictly shrinks the feasible set of the optimization problem. This trade-off is central to layered architectures: what we give up in global optimality, we gain in modularity, scalability, and real-time feasibility. In many cases, this trade-off is not just acceptable, but necessary — in the next section,

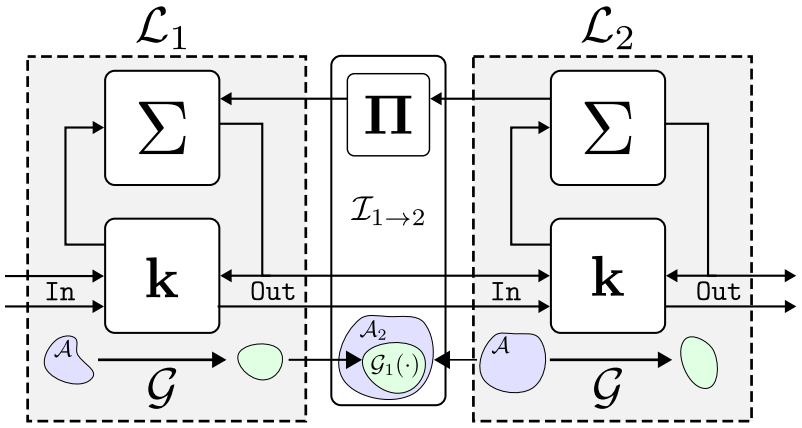


Figure 3.3: A depiction of the six components that make up a level, as well as an interface between the levels such that the assumptions of the lower level ($\mathcal{L}_2$) are satisfied by the guarantees of the level above ($\mathcal{L}_1$).

we formally introduce layered architectures and provide concrete instances in the context of solving Problem 3.1.

3.5 Layered Control Architecture Definition

To formalize the discussion in the previous section, we begin by introducing the notion of a system [133]:

Definition 3.3. Let $\mathcal{X}$ denote the state space, $\mathcal{U}$ the input space, and $\mathbf{f} \subseteq \mathcal{X} \times \mathcal{U} \times \mathcal{X}$ the transition relation. A *system* Σ is given by the tuple $\Sigma = (\mathcal{X}, \mathcal{U}, \mathbf{f})$.

One way to mitigate complexity within a layered architecture is for each controller block to leverage different system representations. To this end, we often rely on planning models which serve as template systems that enable desired behaviors to be constructed in a computationally tractable way. Next, letting $\mathcal{P}(X)$ denote the power set over X, we define a component of the architecture as:

Definition 3.4. A *component* $\mathcal{L} = (\texttt{In}, \texttt{Out}, \Sigma, \mathbf{k}, \mathcal{A}, \mathcal{G})$ is a tuple consisting of an input set $\texttt{In}$, an output set $\texttt{Out}$, a system Σ, a controller $\mathbf{k} : \mathcal{X} \times \texttt{In} \to \mathcal{U}$, assumptions $\mathcal{A} \subseteq \texttt{In}$, and guarantees $\mathcal{G} : \mathcal{A} \to \mathcal{P}(\texttt{Out})$.

Components can be categorized as *levels* (denoted $\mathcal{L}_j$) which can be further refined as *layers* within levels (denoted $\mathcal{L}_j^k$) depending on the semantic purpose they serve. In the context of this work, (high and low) levels will operate on distinct system abstractions and timescales, and (planning and tracking) layers within these levels will

be used as refinements of each level resulting from implementation conveniences. In other words, a level operates on a particular abstraction, while a layer is a finer subdivision within that level. Both levels and layers will be combined to produce layered control architectures.

To achieve the desired guarantee for each component, we must synthesize a controller $\mathbf{k}$ whose closed-loop behavior meets the guarantee under inputs from the assumption set. Given the differences in systems and controllers between components, care must be taken when interfacing components. To make this interconnection precise, we introduce the notion of an *interface* between components:

Definition 3.5. An *interface* exists between two components $\mathcal{L}_1$ and $\mathcal{L}_2$, denoted $\mathcal{L}_1 \leftrightarrows \mathcal{L}_2$ if $\texttt{In}_2 \subseteq \texttt{Out}_1$, there exists a surjective mapping $\mathbf{\Pi}_{1,2} : \mathcal{X}_2 \to \mathcal{X}_1$, and for all $y \in \mathcal{A}_1$ we have that $\mathcal{G}_1(y) \subseteq \mathcal{A}_2$.

The above definition requires that outputs from $\mathcal{L}_1$ must satisfy the assumptions of $\mathcal{L}_2$. During the design phase, $\mathcal{L}_1$ requires information regarding the assumptions of $\mathcal{L}_2$. Critically, this implies that the interactions between these levels are not strictly top-down: high-level plans must respect the performance capabilities of the low-level controllers. Control architectures are characterized by bidirectional information flow in which achievable references are passed downward and certificates of performance are passed upward, enabling compositionality of control components by design. In order to synthesize controllers at each level, we often rely on further breaking levels down into layers:

Definition 3.6. A level $\mathcal{L}_j$ is *refined* by an ordered collection of layers $\{\mathcal{L}_j^k\}_{k=1}^M$, denoted $\mathcal{L}_j \cong \mathcal{L}_j^1 \leftrightarrows \cdots \leftrightarrows \mathcal{L}_j^M$, if the highest layer satisfies $\texttt{In}^1 = \texttt{In}$ and $\mathcal{A}^1 \subseteq \mathcal{A}$, the lowest layer satisfies $\texttt{Out}^M = \texttt{Out}$ and $\mathcal{G}^M \supseteq \mathcal{G}$, and for each layer there exists interfaces such that $\mathcal{L}_j^k \leftrightarrows \mathcal{L}_j^{k+1}$ for $k = 1, \ldots, M-1$.

Given these constructions, we are now equipped to introduce the notion of layered control architectures:

Definition 3.7. A *layered control architecture* $\mathcal{H}$ is an ordered collection of levels $\{\mathcal{L}_j\}_{j=1}^N$, layers that refine each level $\mathcal{L}_j \cong \mathcal{L}_j^1 \leftrightarrows \cdots \leftrightarrows \mathcal{L}_j^M$ for $j = 1, \ldots, N$, and interfaces such that $\mathcal{L}_j \leftrightarrows \mathcal{L}_{j+1}$ for $j = 1, \ldots, N-1$.

3.6 Specialization to Robotic Path Planning

As a motivating example of this abstraction, consider a four-component layered architecture tailored to solving Problem 3.1 consisting of a high and low level, each with a planning and tracking layer. While other decompositions are possible, this formulation reflects common approaches in robotics. Next, we define the responsibilities of each component in the hierarchy. These specifications define the inputs, outputs, assumptions, and guarantees — setting the stage for the constructive synthesis developed in the following sections. For each component, the guarantee is the intersection of the numbered guarantees, i.e., $\mathcal{G}_{\mathrm{K}} = \cap_i \mathcal{G}_{\mathrm{K},i}$, and the map $\mathbf{\Pi}$ is a linear projection.

We begin at the lowest level of the hierarchy. As discussed in the previous section, the low-level controller is responsible for ensuring that the system state $\mathbf{x}(\cdot)$ tracks a continuous-time trajectory $\mathbf{x}_d(\cdot)$ within a bounded tracking error set $\mathcal{E}$, while satisfying input constraints. We now reintroduce the low level controller in terms of the formalization of Definition 3.4:

Low Level Tracking Layer $\mathcal{L}_{\text{Low}}^{\text{Tracking}}$

- `In` : $\iota_{\mathrm{L}} = (\mathbf{x}_d, \mathbf{u}_d) \in C^1(\mathbb{R}_+, \mathcal{X}_{\mathrm{L}}) \times C^1(\mathbb{R}_+, \mathcal{U}_{\mathrm{L}})$
- `Out`: $y_{\mathrm{L}} = (\mathbf{x}, \mathbf{u}) \in C^1(\mathbb{R}_+, \mathcal{X}_{\mathrm{L}}) \times C^1(\mathbb{R}_+, \mathcal{U}_{\mathrm{L}})$
- $\mathcal{G}_{\mathrm{L},1} : \iota_{\mathrm{L}} \mapsto \{y_{\mathrm{L}} \mid \mathbf{x} \in \mathbf{x}_d \oplus \mathcal{E}_{\mathrm{L}}\}$
- $\mathcal{G}_{\mathrm{L},2} : \iota_{\mathrm{L}} \mapsto \{y_{\mathrm{L}} \mid \mathbf{u} \in \mathcal{U}_{\mathrm{L}}\}$,

for $\mathcal{E}_{\mathrm{L}} \subset \mathcal{X}_L$ satisfying $\mathcal{E}_{\mathrm{L}} \subset B_\varepsilon(\mathbf{0})$ for the desired tolerance $\varepsilon > 0$ defined in Problem 3.1. The low level assumes, $\mathcal{A}_L$, that reference signal $(\mathbf{x}_d, \mathbf{u}_d)$ lies in the low level controller's region of attraction, i.e., the trajectory passed down can be stabilized.

Low Level Planning Layer $\mathcal{L}_{\text{Low}}^{\text{Planning}}$

- `In` : $\iota_{\mathrm{L}} = (\bar{\mathbf{x}}_d, \bar{\mathbf{u}}_d) \in C^1(\mathbb{R}_+, \mathcal{X}_{\mathrm{M}}) \times C^1(\mathbb{R}_+, \mathcal{U}_{\mathrm{M}})$
- `Out`: $y_{\mathrm{L}} = (\mathbf{x}_d, \mathbf{u}_d) \in C^1(\mathbb{R}_+, \mathcal{X}_{\mathrm{L}}) \times C^1(\mathbb{R}_+, \mathcal{U}_{\mathrm{L}})$
- $\mathcal{G}_{\mathrm{L},1} : \iota_{\mathrm{L}} \mapsto \{y_{\mathrm{L}} \mid \mathbf{\Pi}_{\mathrm{M,\,L}}(\mathbf{x}_d) \in \bar{\mathbf{x}}_d \oplus \mathcal{E}_{\mathrm{M}}\}$
- $\mathcal{G}_{\mathrm{L},2} : \iota_{\mathrm{L}} \mapsto \{y_{\mathrm{L}} \mid \mathbf{u}_d \in \mathcal{U}_{\mathrm{L}}\}$,

for $\mathcal{E}_{\mathrm{M}} \subset \mathcal{X}_M$ satisfying $\mathbf{\Pi}_{\mathrm{H,\,M}}(\mathcal{E}_{\mathrm{M}}) \subset B_\varepsilon(\mathbf{0})$. Next, we define the high level tracking layer, responsible for converting discrete waypoints into trajectories that robustly satisfy the constraints.

High Level Tracking Layer $\mathcal{L}^{\text{Tracking}}_{\text{High}}$

- `In` : $\iota_{\mathrm{M}} = (\{\bar{\mathbf{x}}_k\}, \{\mathcal{X}_k\}) \in \mathcal{X}_{\mathrm{H}} \times \mathcal{P}(\mathcal{X}_{\mathrm{H}})$
- `Out`: $y_{\mathrm{M}} = (\bar{\mathbf{x}}_d, \bar{\mathbf{u}}_d) \in C^1(\mathbb{R}_+, \mathcal{X}_{\mathrm{M}}) \times C^1(\mathbb{R}_+, \mathcal{U}_{\mathrm{M}})$
- $\mathcal{G}_{\mathrm{M},1} : \iota_{\mathrm{M}} \mapsto \{y_{\mathrm{M}} \,|\, \mathbf{\Pi}_{\mathrm{H,\,M}}(\bar{\mathbf{x}}_d(t)) \in \mathcal{X}_k \ominus \mathbf{\Pi}_{\mathrm{H,\,M}}(\mathcal{E}_L) \forall t \in [kT/N, (k{+}1)T/N)\}$
- $\mathcal{G}_{\mathrm{M},2} : \iota_{\mathrm{M}} \mapsto \{y_{\mathrm{M}} \mid \mathbf{\Pi}_{\mathrm{H,\,M}}\left(\bar{\mathbf{x}}_d\left(\frac{kT}{N}\right)\right) \in \bar{\mathbf{x}}_k \oplus \mathcal{E}_{\mathrm{M}}\}$.

This component assumes, $\mathcal{A}_M$, that each waypoint is reachable from the previous waypoint. It guarantees that the planned trajectories lie in the buffered free space, and that the plan reaches a neighborhood of each waypoint.

At the top of the hierarchy, the high level planning layer selects waypoints and constraint sets to pass down to the low level controller. It assumes, $\mathcal{A}_H$, that the goal point, $\mathbf{x}_G$ is reachable from the initial condition, $\mathbf{x}_0$ under the hierarchical controller composition.

High Level Planning Layer $\mathcal{L}^{\text{Planning}}_{\text{High}}$

- `In`: $\iota_{\mathrm{H}} = (\mathbf{x}_0, \mathbf{x}_G, \mathcal{X}) \in \mathcal{X}_{\mathrm{H}} \times \mathcal{X}_{\mathrm{H}} \times \mathcal{P}(\mathcal{X}_{\mathrm{H}})$
- `Out`: $y_{\mathrm{H}} = (\{\bar{\mathbf{x}}_k\}, \{\mathcal{X}_k\}) \in \mathcal{X}_{\mathrm{H}} \times \mathcal{P}(\mathcal{X}_{\mathrm{H}})$
- $\mathcal{G}_{\mathrm{H},1} : \iota_{\mathrm{H}} \mapsto \{y_{\mathrm{H}} \mid \bar{\mathbf{x}}_0 = \mathbf{x}_0 \text{ and } \bar{\mathbf{x}}_N = \mathbf{x}_G \}$
- $\mathcal{G}_{\mathrm{H},2} : \iota_{\mathrm{H}} \mapsto \{y_{\mathrm{H}} \mid \mathcal{X}_k \subseteq \mathcal{X} \text{ for } k = 0, \ldots, N\}$
- $\mathcal{G}_{\mathrm{H},3} : \iota_{\mathrm{H}} \mapsto \{y_{\mathrm{H}} \mid \bar{\mathbf{x}}_k \in \mathcal{X}_k \text{ for } k = 0, \ldots, N\}$,

where $\mathcal{X} \subseteq \mathcal{X}_{\mathrm{H}}$ is the free space.

The high level, if feasible, guarantees a path connecting the initial condition to the goal which traverses a sequence of collision free sets. Having defined each component and their interfaces, we arrive at the central result of this section: a properly designed hierarchy composes independent assumptions and guarantees into a system-level guarantee of satisfaction of Problem 3.1:

Theorem 3.8. *The hierarchical controller defined by the composition* $\mathcal{H} : \mathcal{L}_{\text{High}} \leftrightarrows \mathcal{L}_{\text{Low}}$ *solves Problem 3.1 if the high level planning layer is feasible at time* $t = 0$.

Proof. If the high level planning layer is feasible at time $t = 0$, then it can produce a sequence of points which connect $\mathbf{x}_0$ to $\mathbf{x}_G$. In order to show that this results in completion of the problem statement, observe that the each interface ensures the outputs of the layer above is contained within the assumptions of the layer below — as such, feasibility of the problem ensures that all assumptions and guarantees are satisfied. Define $\mathbf{\Pi}_{\text{H, L}} \triangleq \mathbf{\Pi}_{\text{H, M}} \circ \mathbf{\Pi}_{\text{M, L}}$ to map states from the low level to the high level. The high level tracking layer plans $\bar{\mathbf{x}}_d$ with sufficient margin that $\mathbf{\Pi}_{\text{H, L}}(\mathbf{x}_{cl}(t)) \in \mathcal{X}_k$ satisfies state constraints. As the high level planning layer guarantees that $\mathcal{X}_k \subset \mathcal{X}$, we can conclude that the hierarchy enforces state constraints as well. Furthermore, as the low level controller is able to enforce tracking of the guarantee, we have that $\mathbf{\Pi}_{\text{H, L}}(\mathbf{x}_{cl}(T)) \in \mathbf{x}_G \oplus \mathbf{\Pi}_{\text{H, M}}(\mathcal{E}_{\text{L}})$, as well as the input constraint $\mathbf{u} \in \mathcal{U}_{\text{L}}$. This achieves the first objective of getting in a neighborhood of the goal. Therefore, we conclude that we are able to solve Problem 3.1. □

This layered structure mirrors control architectures extremely common in practice: high-levels reason over global objectives, and low-levels generate constraint-respecting trajectories and track them with robustness margins. Furthermore, we have demonstrated that these components compose to solve Problem 3.1. Crucially, the specific models and solution methods inside each component have been left undefined — as long as the interfaces are honored, each component can implement any method. In the following sections, we constructively synthesize controllers which satisfy these contracts for legged robotic systems.

Chapter 4: Low Level Tracking Layer

Contents

Closed loop performance is the ultimate metric of success for feedback controllers.

As discussed in the previous chapter, the role of the low level controller is to track desired trajectories coming in from the high level. Specifically, we would like to design a controller such that:

$$\|\mathbf{x}_{\text{cl}}(t) - \mathbf{x}_d(t)\| \leq \varepsilon,$$

for the smallest possible $\varepsilon > 0$ (implying the best tracking performance).

4.1 PD Control

From here on, we will consider the vector relative degree $\gamma = 2$ and square output $\mathbf{y} : \mathcal{X} \to \mathbb{R}^m$, and associated error coordinates $\boldsymbol{\eta} = [\mathbf{y}\ \dot{\mathbf{y}}]$. These represent desired trajectories for the actuated coordinates (the joints) of the robot. A PD controller is of the form:

$$\mathbf{u} = -\mathbf{K}_p(\mathbf{y} - \mathbf{y}_d(t)) - \mathbf{K}_d(\dot{\mathbf{y}} - \dot{\mathbf{y}}_d(t)) + \mathbf{u}_{\text{ff}}(t),$$

for positive definite gain matrices $\mathbf{K}_p$ and $\mathbf{K}_d$, where $\mathbf{u}_{\text{ff}}(t)$ is a feedforward control input computed from a model of the robot. This is a model-free approach to stabilizing to a desired trajectory, and to this day is still the workhorse for stabilization in real-world systems. A natural place to begin our investigations into low level control is to see how well this basic controller works and our robots of interest. Where it fails, more sophisticated techniques will be required.

Quadruped

We start by investigating the use of a PD controller on Vision 60. The quadrupedal robot was first tested on a consistently graded 13° grassy slope with minimal surface variation, which was replicated in RaiSim by creating a plane of constant incline. The quadruped was next asked to traverse a grassy slope just after it had rained with inclination ranging from 20° to 25°. In order to emulate the varying slope in RaiSim, a terrain was created with a sinusoidally oscillating height varying between 20° and 25° with a frequency that approximated that of the outdoor environment. As can be seen in the gait tiles in Figure 4.1, the PD controller allows the robot to successfully amble across both the 13° slope and the 20° and 25° slope despite the unmodeled variation in slope and lowered friction effects. See [134] for a video demonstration of the experimental validation of the quadruped, in which we also showed all five gaits walking on slopes of $0^\circ, 13^\circ, 15^\circ, 20^\circ, 25^\circ$ in RaiSim.

In addition, we logged 20 seconds of experimental data, and compared them with the desired ambling gait designed by the optimization, as seen in Figure 4.2 with

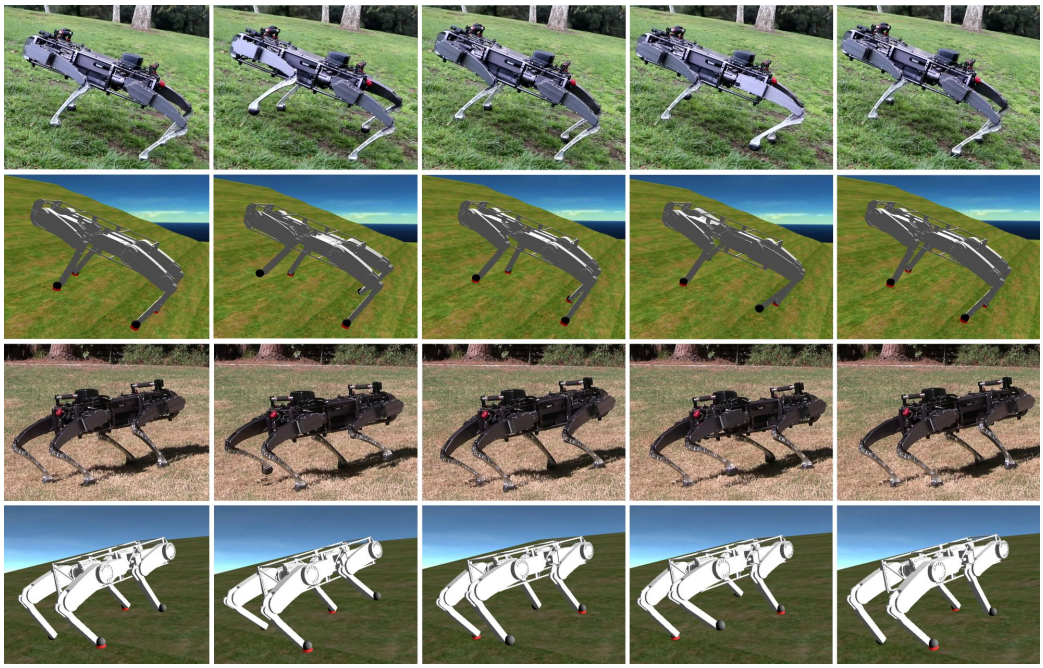


Figure 4.1: Full steps of gait tiles for the Vision 60 ambling in outdoor grasslands. **(Top)** Comparison between simulation and experiments for a 13° slope. **(Bottom)** Walking on a ramp with varying slopes of $20^\circ - 25^\circ$.

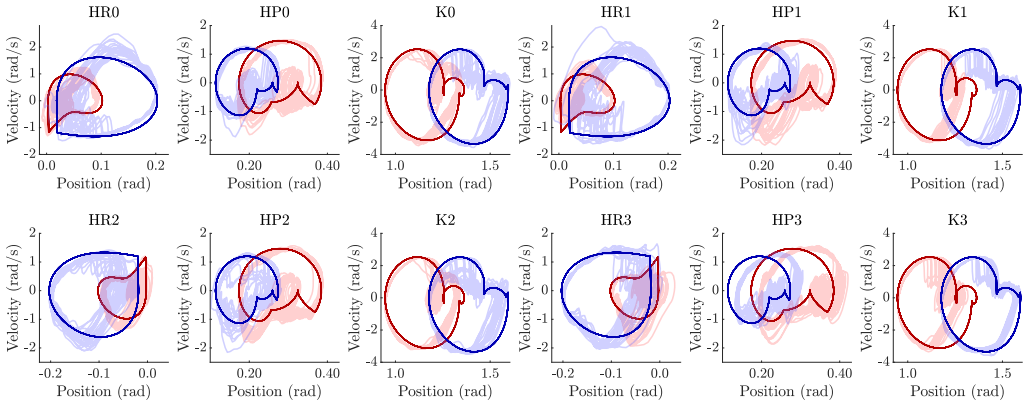


Figure 4.2: Phase portraits of the designed gaits (solid lines) vs. experimental data (transparent overlay) for quadrupedal walking on 13° (red) and $20^\circ - 25^\circ$ (blue) slopes. HR, HP, K are short for hip roll, hip pitch and knee, accordingly.

phase portraits. Note the difference in the desired behavior for the two terrains. These plots demonstrate that PD control produces an empirical tracking invariant around a desired trajectory for a quadruped. As we can see with the quadruped, simply tracking an *a priori* defined time varying trajectory is enough to result in

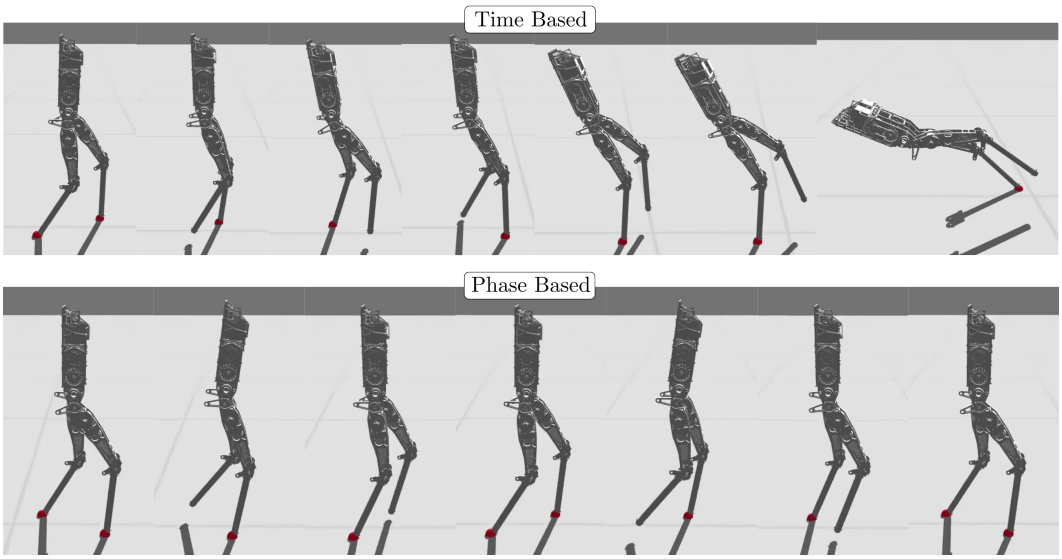


Figure 4.3: Time and state based control on AMBER. **(Top)** Time-based PD control on AMBER quickly leads to falling over due to instabilities in the underactuated center of mass state. **(Bottom)** Phase-based control incorporates feedback with respect to the underactuated states and enables stable walking.

full system stability, even on rough terrain. As alluded to earlier, this is due to the inherent stability in quadrupedal robots.

Biped

Unlike quadrupeds, tracking a single open loop trajectory is insufficient for stabilizing bipeds. This can be seen in Figure 4.3, where AMBER quickly becomes unstable and falls over. This is due to the fact that there is no feedback with respect to the underactuated coordinates, the hip position. As these dynamics are unstable and persist autonomously, they cause the system to be destabilized. This can also be analyzed through the lens of zero dynamics by noting that $\boldsymbol{\omega}(\boldsymbol{\eta}_d(t), \mathbf{z})$ is unstable.

For planer bipeds, there is a convenient way to parameterize the desired trajectory to add feedback to these underactuated coordinates. To do this, we introduce a *phasing variable*, $\tau : \mathcal{Q} \to [0, 1]$, given by:

$$\tau(\mathbf{q}) = \frac{\delta_{hip}(\mathbf{q}) - \delta_{hip}^{+}}{\delta_{hip}^{-} - \delta_{hip}^{+}}, \tag{4.1}$$

where $\delta_{hip} : \mathbb{R}^5 \to \mathbb{R}$ defined as $\delta_{hip}(\mathbf{q}) = [-l_t - l_f,\ -l_f,\ 0,\ 0,\ 0]\mathbf{q}$ is the linearized hip position with l_t and l_f the length of the robots tibia and femur, respectively. The constants δ_{hip}^{+} and δ_{hip}^{-} are the linearized hip positions at the beginning and the end

of a step, ensuring that $\tau(\mathbf{q})$ increases monotonically in time within a step. We are now well equipped to define the relative degree 2 ([109]) outputs $\mathbf{y} : \mathcal{Q} \to \mathbb{R}^4$ as the difference between the actual output $\mathbf{y}_a$ and the desired output trajectory $\mathbf{y}_d$:

$$\mathbf{y}(\mathbf{q}) \triangleq \mathbf{y}_a(\mathbf{q}) - \mathbf{y}_d(\tau(\mathbf{q})), \tag{4.2}$$

with $\boldsymbol{\alpha}$ being the coefficients of a Bézier polynomial coming from the trajectory generation step. The actual output is given by the actuated coordinates: $\mathbf{y}_a(\mathbf{q}) = \begin{bmatrix}\mathbf{0}_{4\times 1} & \mathbf{I}_{4\times 4}\end{bmatrix}\mathbf{q}$. The nominal controller for this system is then given by the PD controller

$$\mathbf{k}_{PD}(\mathbf{x}) \triangleq -\mathbf{K}_P\mathbf{y}(\mathbf{q}) - \mathbf{K}_D\dot{\mathbf{y}}(\mathbf{q}),$$

with proportional gain $\mathbf{K}_P \in \mathbb{S}^4_{++}$ and derivative gain $\mathbf{K}_D \in \mathbb{S}^4_{++}$. As seen in Figure 4.3, this feedback term is capable of stabilizing the system in a neighborhood of the desired periodic orbit. Unlike earlier, our output now becomes a function of the zero dynamics coordinate, and renders the zero dynamics $\boldsymbol{\omega}(\boldsymbol{\eta}_d(\tau(\mathbf{q})), \mathbf{z})$ stable. In fact, for this planar system, stability of the zero dynamics can be directly enforced as part of the trajectory optimization problem.

Note that in order to successfully do this reparameterization, we must ensure that the phasing variable function is monotonically increasing over the step, which ensures that τ is a bijection between $[\delta^+_{\text{hip}}\ \delta^-_{\text{hip}}]$ and $[0\ 1]$. This too can be added as a constraint to the trajectory optimization problem. Within this framework, the step length is fixed — rather, stability is derived from speeding up or slowing down the gait along its predefined trajectory based on where the underactuated state is.

Hopper

In the case of ARCHER, the control task is complicated by the fact that the actuated coordinates live on the manifold S^3. When the system states lie on a Lie group, we must be more careful about the computation of the error coordinates. Consider the quaternion $q \in \mathcal{S}^3$, the body frame angular rate as $\boldsymbol{\omega} \in \mathfrak{s}^3$, and a desired orientation $q_d \in \mathcal{S}^3$. Then, the torque is given by:

$$\mathbf{u} = -\mathbf{K}_{\text{p}} \log(q_d^{-1}q) - \mathbf{K}_{\text{d}}\boldsymbol{\omega} + \mathbf{u}_{\text{ff}}(t).$$

The $\log : \mathcal{S}^3 \to \mathfrak{s}^3$ map can also be approximated via $\mathbb{I}m : \mathcal{S}^3 \to \mathfrak{s}^3$, which takes only the imaginary component of the quaternion and can be used to simplify the computation. The log map is a distance preserving bijection, and the imaginary map can be thought of as a linear projection map onto the Lie algebra.

As with the biped, applying this controller to a fixed trajectory will cause the robot to almost immediately fall over. Unlike the biped, there is no easy way to produce a phasing variable to stabilize this system as this would require finding a bijection between the 2D underactuated center of mass position and the interval $[0\ 1]$, which is impossible due to the difference in dimension. As such, we will need to be more intelligent about the design of the desired trajectory that we try to stabilize to, which we will investigate in the next chapter. The remainder of this section will focus on the low level control of the planar biped AMBER, as the low level for quadrupeds can effectively be controlled with PD control, and we do not yet have the necessary tools to discuss how to stabilize the 3D hopping robot.

4.2 Data-Driven Performance via Preference Based Learning

As we saw in the previous section, simple controllers like PD can generate empirical tracking invariants given well tuned gains. In theory, switching to a controller that leverages a model of the system could provide a much tighter tracking invariant; however, in practice model based controllers are often difficult to tune because of modeling errors and an unintuitive relationship between the gains and the resulting performance. We specifically investigate the use of CLF based controllers at the low level in an attempt to improve the tracking error bound, which provide formal guarantees of exponential convergence under idealized conditions but are often sensitive to model mismatch and controller parameterization. Therefore, in this section, we investigate the use of a Bayesian Optimization method called *Preference Based Learning* [135], which leverages user preferences to maximize unmeasurable user utility functions as a means of model-based gain tuning, in the hopes that it will provide better tracking invariants for our systems of interest.

With the goal of constructing a CLF, recall that the feedback linearized error coordinates $\boldsymbol{\eta}$ for the output $\mathbf{y}$ from (4.2) have the following dynamics:

$$\dot{\boldsymbol{\eta}} = \mathbf{F}\boldsymbol{\eta} + \mathbf{G}\mathbf{v}. \tag{4.3}$$

Instead of solving the CTLE as we did in (2.4), we instead phrase this as an LQR problem whereby we evaluate the continuous time algebraic Riccati equation:

$$\mathbf{F}^\top\mathbf{P} + \mathbf{P}\mathbf{F} + \mathbf{P}\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^\top\mathbf{P} + \mathbf{Q} = \mathbf{0}, \tag{CARE}$$

which has a solution $\mathbf{P} \succ 0$ for any $\mathbf{Q} = \mathbf{Q}^\top \succeq 0$ and $\mathbf{R} = \mathbf{R}^\top \succ 0$. From (CARE),

we can construct a rapidly exponentially stabilizing CLF (RES-CLF) [136]:

$$V(\boldsymbol{\eta}) = \boldsymbol{\eta}^\top \mathbf{I}_\epsilon \mathbf{P} \mathbf{I}_\epsilon \boldsymbol{\eta}, \quad \mathbf{I}_\epsilon = \begin{bmatrix} \frac{1}{\epsilon}\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{4.4}$$

where $0 < \epsilon < 1$ is a tunable parameter that drives the (rapid) exponential convergence. Any feedback controller, $\mathbf{u}$, which can satisfy the convergence condition:

$$\dot{V}(\boldsymbol{\eta}) = L_f V(\boldsymbol{\eta}) + L_g V(\boldsymbol{\eta}) \mathbf{u} \leq -\frac{1}{\epsilon} \underbrace{\frac{\lambda_{\min}(\mathbf{Q})}{\lambda_{\max}(\mathbf{P})}}_{\gamma} V(\boldsymbol{\eta}), \tag{4.5}$$

will then render rapidly exponential stability for the output dynamics (4.3).

To enforce (4.5), a quadratic program (CLF-QP) [137], with (4.5) as an inequality constraint can be posed. Implementing this controller on physical systems, which are often subject to additional constraints such as torque bounds or friction limits, suggests that relaxation for the inequality constraint should be used. The introduction of relaxation and the need to reduce torque chatter on physical hardware lead to the following relaxed (CLF-QP) with incentivized convergence in the cost [138]:

$$\begin{aligned} \mathbf{u}^* = \underset{\mathbf{u} \in \mathbb{R}^m}{\operatorname{argmin}} \quad & \|L_{\mathbf{f}}^2 \mathbf{y}(\mathbf{x}) + L_{\mathbf{g}} L_{\mathbf{f}} \mathbf{y}(\mathbf{x}) \mathbf{u}\|^2 + w_{\dot{V}} \dot{V}(\mathbf{x}, \mathbf{u}) \\ \text{s.t.} \quad & \mathbf{u}_{min} \preceq \mathbf{u} \preceq \mathbf{u}_{max}. \end{aligned} \tag{CLF-QP$^+$}$$

Parameterization of CLF-QP

For the following discussion, let $\boldsymbol{a} = [a_1, ..., a_v] \in \mathbf{A} \subset \mathbb{R}^v$ be an element of a $v-$dimensional parameter space, termed an *action*. We let $\mathbf{Q} = \mathbf{Q}(\boldsymbol{a})$, $\epsilon = \epsilon(\boldsymbol{a})$, and $w_{\dot{V}} = w_{\dot{V}}(\boldsymbol{a})$ denote a parameterization of our control tuning variables, which will subsequently be learned. Each gain a_i for $i = 1, \ldots, v$ is discretized into d_i values, leading to an overall search space of actions given by the set $\mathbf{A}$ with cardinality $|\mathbf{A}| = \prod_{i=1}^{v} d_i$. For the AMBER robot, v is taken to be 6 with discretizations $d = [4, 4, 5, 5, 4, 5]$, resulting in the following parameterization:

$$\begin{aligned} \mathbf{Q}(\boldsymbol{a}) = \begin{bmatrix} \mathbf{Q}_1 & 0 \\ 0 & \mathbf{Q}_2 \end{bmatrix}, \quad & \mathbf{Q}_1 = \text{diag}([a_1, a_2, a_2, a_1]), \\ & \mathbf{Q}_2 = \text{diag}([a_3, a_4, a_4, a_3]), \\ \epsilon(\boldsymbol{a}) = a_5, \qquad & w_{\dot{V}}(\boldsymbol{a}) = a_6, \end{aligned}$$

which satisfies $\mathbf{Q}(\boldsymbol{a}) \succ 0$, $0 < \epsilon(\boldsymbol{a}) < 1$, and $w_{\dot{V}}(\boldsymbol{a}) > 0$ for the choice of bounds, as summarized in Table 4.1.

Learning Framework

As in [139], the preference-based learning framework leveraged in this section is aimed at regret minimization, defined as sampling N actions $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N\}$ such that:

$$\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N\} = \underset{\boldsymbol{a} \in \mathbf{A}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(U(\boldsymbol{a}^*) - U(\boldsymbol{a}_i) \right),$$

where $\mathbf{A} \subset \mathbb{R}^{|\boldsymbol{a}|}$ is the discretized set of all possible actions, $U : \mathbf{A} \to \mathbb{R}$ is the underlying utility function of the human operator mapping each action to a subjective measure of "good," and $\boldsymbol{a}^*$ is the action maximizing U. This iterative process of (1) querying the operator for feedback, (2) modeling the underlying utility function, and (3) sampling new actions, is repeated in each subsequent iteration. This paradigm of optimization is especially interesting, as it is not even a zeroth order method — we can never evaluate the function U; rather, we rely on comparisons of subsequent trials to estimate the gradient of the utility. Once the algorithm terminates, the best action after the completion of the experiment is given by $\hat{\boldsymbol{a}}^* = \operatorname{argmax}_{\boldsymbol{a} \in \mathbf{A}} \mu(\boldsymbol{a})$.

Learning to Walk in Experiments

Preference-based learning was applied to the realization of optimization-based control on two separate robotic platforms: the 5 DOF planar biped AMBER, and the 22 DOF 3D biped Cassie, as can be seen in the video [140]. As illustrated in Fig. 4.4, the experimental procedure had four main components: the physical robot (either AMBER or Cassie), the controller running on a real-time PC, a human operator providing feedback, and a secondary PC running the learning algorithm. Each action was tested for approximately one minute, during which the behavior of the robot was evaluated in terms of both performance and robustness. User feedback in the form of pairwise preferences and ordinal labels was obtained after testing each action via the respective questions: "Do you prefer this behavior more or less than the last behavior," and "Would you give this gait a label of very bad, neutral, or very good." After user feedback was collected for the sampled controller gains, the posterior was inferred over all of the uniquely sampled actions, which took up to

Table 4.1: Learned Parameters for AMBER.

AMBER				
	Pos. Bounds	Vel. Bounds		Bounds
Q Knees	a_1:[100, 1500]	a_3:[10, 300]	ϵ	a_5:[0.08, 0.2]
Q Hips	a_2:[100, 1500]	a_4:[10, 300]	$w_{\dot{V}}$	a_6:[1, 5]

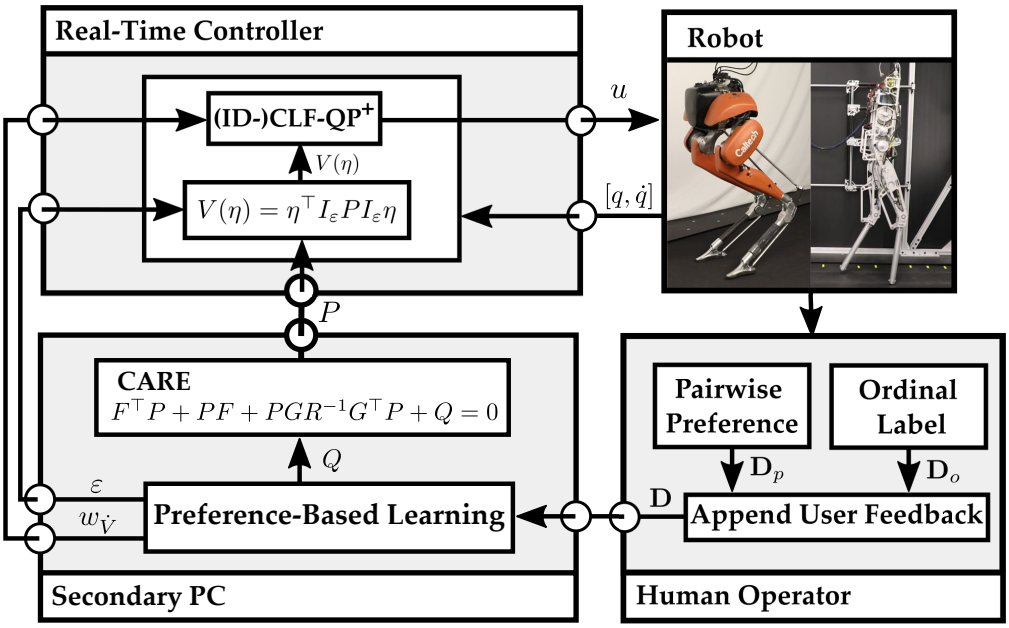


Figure 4.4: The experimental procedure for Preference Based Learning for gain tuning.

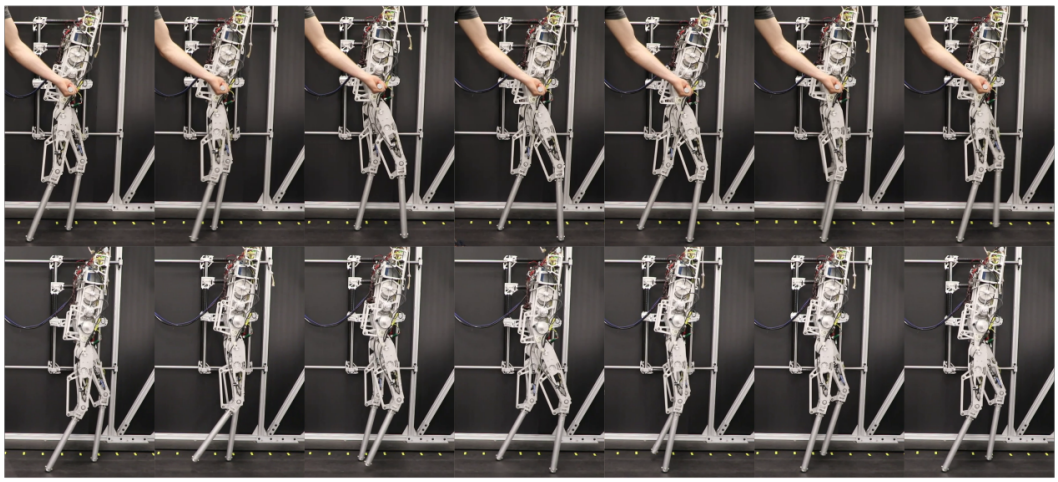


Figure 4.5: Very low utility (top) where the robot was unable to walk unassisted and maximum posterior utility (bottom) where stable walking was achieved.

0.5 seconds. The experiment with AMBER was conducted for 50 iterations, lasting one hour.

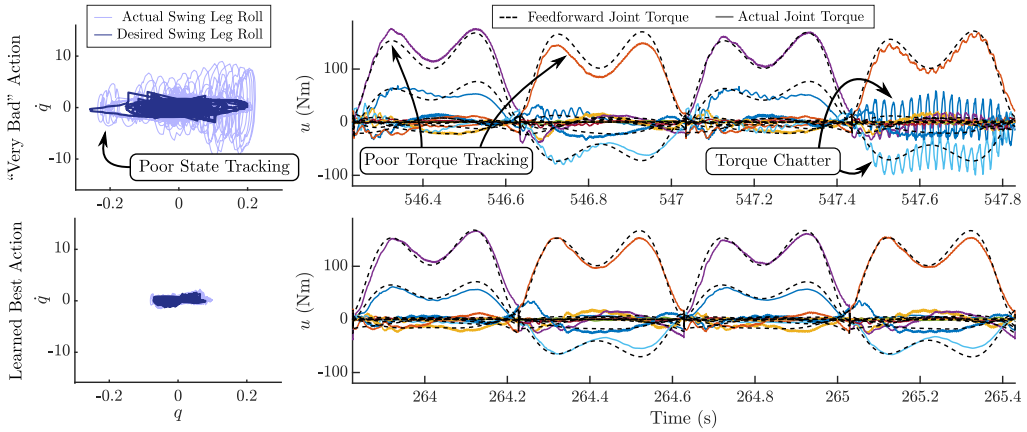


Figure 4.6: Phase plots and torques commanded. For torques, each colored line corresponds to a different joint, with the black dotted lines being the feedforward torque. The gains corresponding to a "very bad" action (top) yield torques that exhibit poor tracking on joints and torque chatter. On the other hand, the gains corresponding to the learned optimal action (bottom) exhibit much better tracking and no torque chatter.

Results with AMBER — CLF-QP$^+$

During the first half of the experiment, the algorithm sampled a variety of gains causing behavior ranging from instantaneous torque chatter to induced tripping due to inferior output tracking. It is important to note that none of the initial sampled values led to unassisted walking.

By the end of the experiment however, the algorithm had sampled 3 gains which were deemed "very good," and which resulted in stable walking. Gait tiles for an action deemed "very bad," as well as the learned best action are shown in Fig. 4.5. Additionally, tracking performance for the two sets of gains is seen in Fig. 4.6, where the learned best action tracks the desired behavior to a better degree.

Features of this optimal action, compared to a worse action sampled in the beginning of the experiments, are outlined in Fig. 4.6. In terms of quantifiable improvement, the difference in tracking performance is shown in Fig. 4.7. The magnitude of the tuned parameters, η_t, illustrates the improvement that preference-based learning attained in tracking the outputs it intended to. At the same time, the tracking error of the constant parameters, η_{nt}, shows that the outputs that were not tuned remained unaffected by the learning process. This quantifiable improvement is further illustrated by the commanded torques in Fig. 4.6, which show that the optimal gains result in much less torque chatter and better tracking as compared to the other gains.

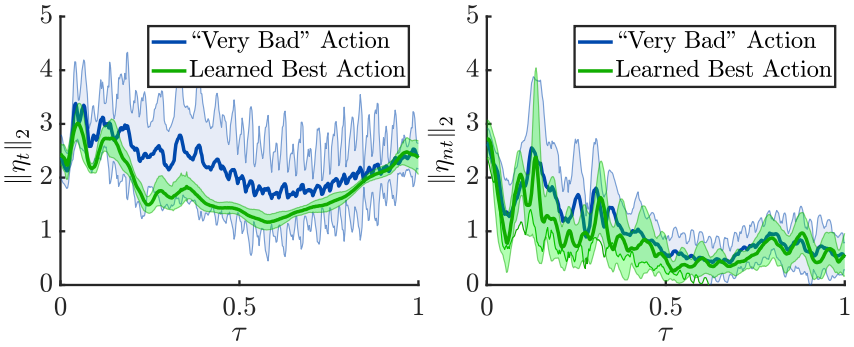


Figure 4.7: Output error of the tuned parameters η_t (left) and untuned parameters η_{nt} (right).

4.3 Data-Driven Safety

In the previous section, we leveraged learning to improve the performance of low-level controllers through better parameter tuning. We now shift focus from performance to safety, aiming to enforce an additional certificate—namely, forward set invariance—at the same control layer. In particular, we investigate how safety-critical constraints, encoded as control barrier functions (CBFs), can be robustly enforced despite modeling error by learning the projected disturbances that compromise the CBF condition.

In practice, the system dynamics (3.1) are not known during control design due to parametric error and unmodeled dynamics. Instead, a nominal model of the system is utilized:

$$\dot{\hat{\mathbf{x}}} = \hat{\mathbf{f}}(\mathbf{x}) + \hat{\mathbf{g}}(\mathbf{x})\mathbf{u}, \tag{4.6}$$

where $\hat{\mathbf{f}} : \mathbb{R}^n \to \mathbb{R}^n$ and $\hat{\mathbf{g}} : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ are assumed to be Lipschitz continuous on $\mathbb{R}^n$. By adding and subtracting the right hand side of (4.6) to (3.1), the dynamics of the system are:

$$\dot{\mathbf{x}} = \hat{\mathbf{f}}(\mathbf{x}) + \hat{\mathbf{g}}(\mathbf{x})\mathbf{u} + \overbrace{\underbrace{\mathbf{f}(\mathbf{x}) - \hat{\mathbf{f}}(\mathbf{x})}_{\mathbf{b}(\mathbf{x})} + \underbrace{(\mathbf{g}(\mathbf{x}) - \hat{\mathbf{g}}(\mathbf{x}))}_{\mathbf{A}(\mathbf{x})}\mathbf{u}}^{\mathbf{d}(\mathbf{x},\mathbf{u})}, \tag{4.7}$$

where the unknown disturbance $\mathbf{d}(\mathbf{x},\mathbf{u}) = \mathbf{b}(\mathbf{x}) + \mathbf{A}(\mathbf{x})\mathbf{u}$ is assumed to be time invariant, but depends on the state and input to the system. If the function $h : \mathbb{R}^n \to \mathbb{R}$ is a CBF for the nominal model (4.6) on $\mathcal{C}$, the uncertainty in the dynamics directly manifests in the time derivative of h:

$$\dot{h}(\mathbf{x},\mathbf{u}) = \underbrace{\nabla h(\mathbf{x})(\hat{\mathbf{f}}(\mathbf{x}) + \hat{\mathbf{g}}(\mathbf{x})\mathbf{u})}_{\hat{\dot{h}}(\mathbf{x},\mathbf{u})} + \underbrace{\nabla h(\mathbf{x})\mathbf{b}(\mathbf{x})}_{b(\mathbf{x})} + \underbrace{\nabla h(\mathbf{x})\mathbf{A}(\mathbf{x})}_{\mathbf{a}(\mathbf{x})^\top}\mathbf{u}. \tag{4.8}$$

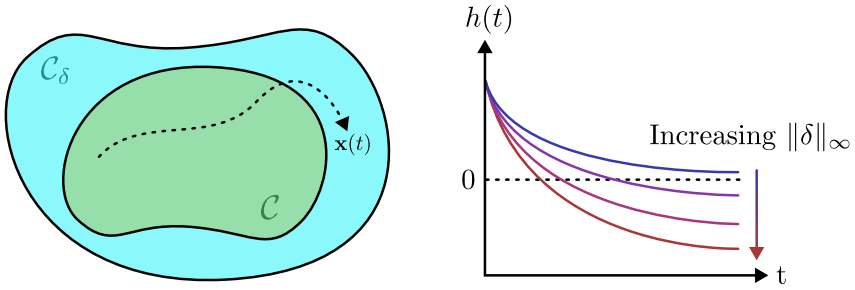


Figure 4.8: Geometric visualization of Projection-to-State Safety. The system is able to leave the safe set $\mathcal{C}$, but remains within the larger set $\mathcal{C}_\delta$. Maximum possible deviation from the safe set grows larger with $|\delta|_\infty$.

Given that h is a CBF for (4.6) on $\mathcal{C}$, let $\mathbf{k} : \mathbb{R}^n \to \mathbb{R}^m$ be a Lipschitz continuous state-feedback controller such that $\dot{\hat{h}}(\mathbf{x}, \mathbf{k}(\mathbf{x})) \geq -\alpha(h(\mathbf{x}))$. Defining the *projected disturbance* as:

$$\delta(\mathbf{x}) \triangleq \dot{h}(\mathbf{x}, \mathbf{k}(\mathbf{x})) - \dot{\hat{h}}(\mathbf{x}, \mathbf{k}(\mathbf{x})) = b(\mathbf{x}) + \mathbf{a}(\mathbf{x})^\top \mathbf{k}(\mathbf{x}), \tag{4.9}$$

yields:

$$\dot{h}(\mathbf{x}, \mathbf{k}(\mathbf{x})) \geq -\alpha(h(\mathbf{x})) - \delta(\mathbf{x}). \tag{4.10}$$

The projected disturbance δ appears in the time derivative of the barrier function $\dot{h}$, and potentially leads to unsafe behavior since it compromises the CBF condition. If an upper bound $\overline{\delta}$ on $|\delta|_\infty$ is known (or determined heuristically), it could be directly incorporated into the inequality enforced in the controller:

$$\begin{aligned} \mathbf{k}(\mathbf{x}) = \ & \underset{\mathbf{u} \in \mathbb{R}^m}{\operatorname{argmin}} \quad \frac{1}{2}\|\mathbf{u} - \mathbf{k}_d(\mathbf{x})\|_2^2 \\ & \text{s.t.} \quad \dot{\hat{h}}(\mathbf{x}, \mathbf{u}) - \overline{\delta} \geq -\alpha(h(\mathbf{x})). \end{aligned} \tag{$\overline{\delta}$-CBF-QP}$$

While this will enforce safety of the original set $\mathcal{C}$, it can be exceedingly conservative if $\overline{\delta}$ is larger than the actual projected disturbance. Furthermore, as the projected disturbance is a function of the state, its magnitude (and possibly sign) may change along a trajectory, leading to additional conservativeness in this approach.

Instead, we consider a learning approach to resolve the impact of δ. To motivate such an approach, consider the following setting: in an experiment, the system is allowed to evolve forward in time from a particular initial condition and under a given state-feedback controller. During this experiment, data is collected which provides a discrete-time history of the CBF, h. This time history is smoothed and

numerically differentiated to compute an approximate time history of the true value of the time derivative of the CBF, $\dot{h}$. This yields a collection of input-output pairs:

$$D_i = ((\mathbf{x}_i, \mathbf{k}(\mathbf{x}_i)), \dot{h}_i) \in (\mathbb{R}^n \times \mathbb{R}^m) \times \mathbb{R} \tag{4.11}$$

whereby a dataset $\mathfrak{D} = \{D_i\}_{i=1}^N$ can be constructed. Given a nonlinear function class $\mathcal{H} : \mathbb{R}^n \to \mathbb{R}$ and a loss function $\mathcal{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, a learning problem can be specified as finding a function $\widehat{\delta} \in \mathcal{H}$ to estimate δ via empirical risk minimization:

$$\inf_{\widehat{\delta} \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(\widehat{\dot{h}}(\mathbf{x}_i, \mathbf{k}(\mathbf{x}_i)) + \widehat{\delta}(\mathbf{x}_i), \dot{h}_i\right). \tag{ERM}$$

A controller can be synthesized which incorporates $\widehat{\delta}$ as follows:

$$\begin{aligned} \mathbf{k}(\mathbf{x}) = \ & \underset{\mathbf{u} \in \mathbb{R}^m}{\operatorname{argmin}} \quad \frac{1}{2}\|\mathbf{u} - \mathbf{k}_d(\mathbf{x})\|_2^2 \\ & \text{s.t.} \qquad \widehat{\dot{h}}(\mathbf{x}, \mathbf{u}) + \widehat{\delta}(\mathbf{x}) \geq -\alpha(h(\mathbf{x})). \end{aligned} \tag{$\widehat{\delta}$-CBF-QP}$$

Note that compared with the standard CBF formulation, the extended safe set with Equation ($\widehat{\delta}$-CBF-QP) shrinks from (2.17) to

$$\mathcal{C}_\delta = \left\{\mathbf{x} \in \mathbb{R}^n : h(\mathbf{x}) + \gamma(|\delta - \hat{\delta}|_\infty) \geq 0\right\}.$$

We directly build upon the episodic learning framework from [141], [142] by seeking to learn δ. Our approach is outlined in Algorithm 1. In each episode, the algorithm runs the current controller to collect data, learns a new $\widehat{\delta}$ using the newly collected data, and synthesizes a new controller. In this prior work, which was applied to less complex dynamical systems, the collected data was rich enough to determine a control affine structure. In many contexts, such as bipedal robots, such a degree of diversity is infeasible without damaging the system. We instead directly learn δ as a function of the previous controller $\mathbf{k}$ via a recursive relationship, as updating the estimator leads to the definition of a new projected disturbance $\delta' = b(\mathbf{x}) + \mathbf{a}(\mathbf{x})^\top \mathbf{k}'(\mathbf{x})$. This yields a projected disturbance δ learned iteratively by modifying $\widehat{\delta}$ over the course of multiple episodes. This episodic approach to safety-critical control is captured in Algorithm 1.

Control Barrier Functions for Stepping-Stones

The stepping-stone problem is captured through the use of *virtual* stepping-stones, which shrink over the course of a step to confine foot placement to a safe region

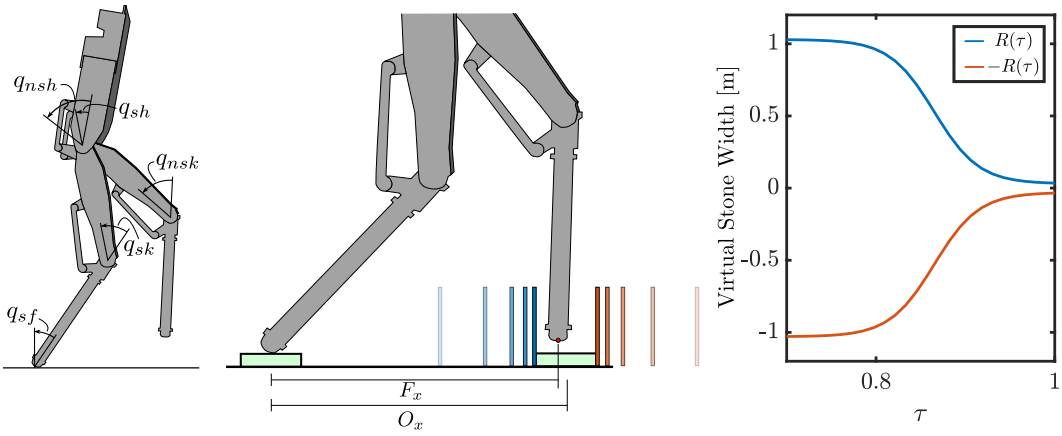


Figure 4.9: Barrier function stepping stones on AMBER. **(Left):** Schematic diagram of the AMBER-3M robot with position coordinates. **(Center):** Schematic of the foot placement in the stepping-stone problem. The boundaries of virtual stepping-stones are captured via the blue and orange vertical lines. **(Right):** Virtual stepping stone width as function of the phase variable $\tau(\mathbf{q})$.

defined on a targeted stone [143]. The CBFs used to specify these foot position constraints are given by:

$$h_1(\mathbf{q}) = R(\tau(\mathbf{q})) - (O_x - F_x(\mathbf{q})), \tag{4.12}$$

$$h_2(\mathbf{q}) = R(\tau(\mathbf{q})) + (O_x - F_x(\mathbf{q})), \tag{4.13}$$

where $F_x(\mathbf{q})$ is the horizontal position of the swing foot and $O_x > 0$ is the horizontal position of the center of stepping-stone. The virtual stone width is given by the function $R : \mathbb{R} \to \mathbb{R}$:

$$R(\tau(\mathbf{q})) = \frac{ar - 1}{1 + ar(e^{-m(\tau(\mathbf{q})-1)} - 1)} + 1 + r \tag{4.14}$$

Algorithm 1 Projected Disturbance Learning (PDL)

Input: CBF h, CBF derivative estimate $\widehat{\dot{h}}$, model class $\mathcal{H}$, loss function $\mathcal{L}$, nominal state-feedback controller $\mathbf{k}_0$, number of episodes T, initial condition $\mathbf{x}_0$
Output: Augmented Controller $\mathbf{k}_T$

1: **for** $j = 1, \ldots, T$ **do**
2: $\quad \mathfrak{D}_j \leftarrow$ EXPERIMENT($\mathbf{x}_0, \mathbf{k}_{j-1}$) ▷ Execute experiment
3: $\quad \widehat{\delta} \leftarrow$ ERM($\mathcal{H}, \mathcal{L}, \mathfrak{D}_j, \widehat{\dot{h}}_0$) ▷ Fit estimator
4: $\quad \widehat{\dot{h}}_j \leftarrow \widehat{\dot{h}}_0 + \widehat{\delta}$ ▷ Update derivative estimator
5: $\quad \mathbf{k}_j \leftarrow \widehat{\delta}$-CBF-QP($\widehat{\dot{h}}_j$) ▷ Synthesize new controller
6: **end for**

where $m > 0$ determines the decay rate of the barrier function, $(1+a)r$ is half of the targeted stone width, and $1+r$ defines the half the width of the virtual stepping-stone when $\tau = 0$. These functions are visualized in Figure 4.9. The safety constraints can be interpreted as keeping the swing foot horizontal position in an interval centered at the middle of the stepping-stone, where the interval shrinks as τ increases.

As this formulation of CBFs is position-based and therefore relative degree two, we employ the exponential control barrier function (ECBF) extension technique [115] to both CBFs to attain the relative degree 1 CBFs: $h_{e,i}(\mathbf{x}) \triangleq L_{\mathbf{f}} h_i(\mathbf{x}) + \alpha_e h_i(\mathbf{q})$.

The final Stepping Stone QP (SS-QP) controller combines the robustifying term of ($\widehat{\delta}$-CBF-QP) with the stepping-stone ECBF extensions of (4.12) and (4.13):

$$\mathbf{k}(\mathbf{x}) = \underset{\mathbf{u}\in\mathbb{R}^m}{\operatorname{argmin}} \quad \frac{1}{2}\|\mathbf{u} - \mathbf{k}_{PD}(\mathbf{x})\|_2^2 \tag{SS-QP}$$

$$\text{s.t.} \quad L_{\hat{\mathbf{f}}}^2 h_1(\mathbf{x}) + L_{\hat{\mathbf{g}}} L_{\hat{\mathbf{f}}} h_1(\mathbf{x})\mathbf{u} + \alpha_e L_{\hat{\mathbf{f}}} h_1(\mathbf{x}) + \widehat{\delta}_1(\mathbf{x}) \geq -\alpha(h_{e,1}(\mathbf{x})) \tag{4.15}$$

$$L_{\hat{\mathbf{f}}}^2 h_2(\mathbf{x}) + L_{\hat{\mathbf{g}}} L_{\hat{\mathbf{f}}} h_2(\mathbf{x})\mathbf{u} + \alpha_e L_{\hat{\mathbf{f}}} h_2(\mathbf{x}) + \widehat{\delta}_2(\mathbf{x}) \geq -\alpha(h_{e,2}(\mathbf{x})). \tag{4.16}$$

Simulation and Experimental Validation

In this section we apply our episodic learning framework (Algorithm 1) to the AMBER-3M platform in both simulation with injected model uncertainty and on hardware with the model error inherent to real-world systems. In each instance the estimator $\widehat{\delta}$ was implemented as a neural network with two hidden layers of 50 hidden units using the ReLU activation function. The network was trained minimizing mean absolute error using mini-batch gradient descent. Mean absolute error was chosen over other loss functions for its robustness to outliers. The same controller (SS-QP) was deployed in the RaiSim [144] simulation environment and on the AMBER-3M hardware platform, as seen in the supplementary video ([145]). The complete learning code can be found at [146].

Simulation

The controllers and learning algorithm were first validated in simulation. Model error was introduced by increasing the inertia of all limbs on the true model by a factor of ten while maintaining constant mass. Due to the underactuated nature of the robot and the relationship between step length and zero dynamics stability, not every set of stepping stones is navigable, even if safety is perfectly enforced with

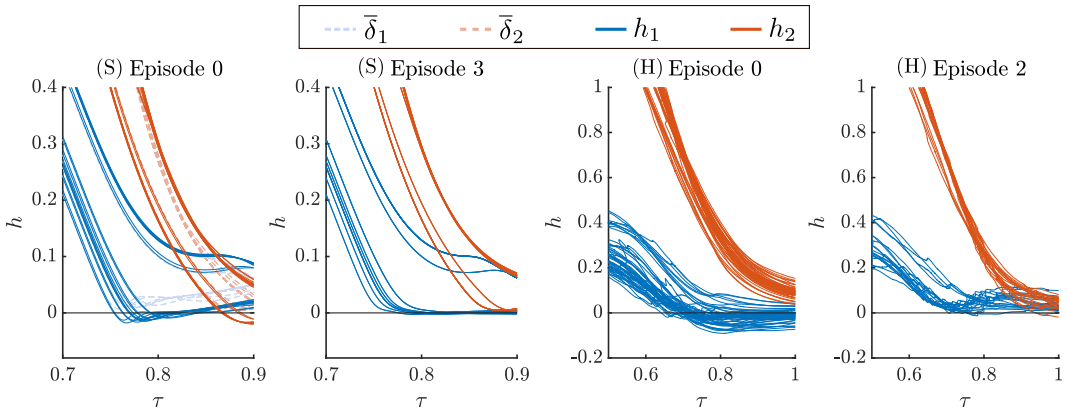


Figure 4.10: Simulation (S) and Hardware (H) data where model mismatch causes violations. **(Far-Left)**: Simulation where the barrier functions h_1 (solid blue) and h_2 (solid orange) are enforced via a CBF-QP. The ($\overline{\delta}$-CBF-QP) is also shown for $\overline{\delta}_1$ (dashed blue) and $\overline{\delta}_2$ (dashed orange), which results in more conservative behavior over many steps. **(Mid-Left)**: After three episodes of learning the (SS-QP) in simulation, the maximum barrier violation decreases from 2.0 to 0.3 cm. **(Mid-Right)**: Hardware where the barrier functions h_1 (blue) and h_2 (orange) enforced via a CBF-QP. **(Far-Right)**: After two episodes of learning on hardware, the maximum barrier violation decreases from 9.2 to 1.9 cm via the (SS-QP).

respect to the CBFs. Therefore, a feasible stepping stone configuration was first generated for the robot to traverse with stones of 4 cm in width. Without knowledge of the modified model ($\widehat{\delta}_1(\mathbf{x}) = \widehat{\delta}_2(\mathbf{x}) = 0$), the controller did not satisfy the CBF constraints (4.15-4.16), resulting in a maximum violation at foot placement of 2.0 cm, causing the robot to miss the stepping stone and fall over. Three episodes of the PDL algorithm were run, after which the maximum violation was reduced to be 0.3 cm, only 15% of the original violation. Additionally, the ($\overline{\delta}$-CBF-QP) controller was implemented, which ensured safety but resulted in extremely conservative behavior, resulting in poor qualitative walking, i.e., harsh foot strikes and an over-bending torso. A comparison of the barrier functions h_1 and h_2 over the steps with these controllers can be seen in Figure 4.10.

Hardware

The same nominal model for the robot was used in the hardware experiments as in simulation, with model uncertainty presenting itself as significant friction in the joints, as well as imperfect mass and inertia measurements. The PDL algorithm was implemented on the AMBER-3M robot across a sequence of two episodes.

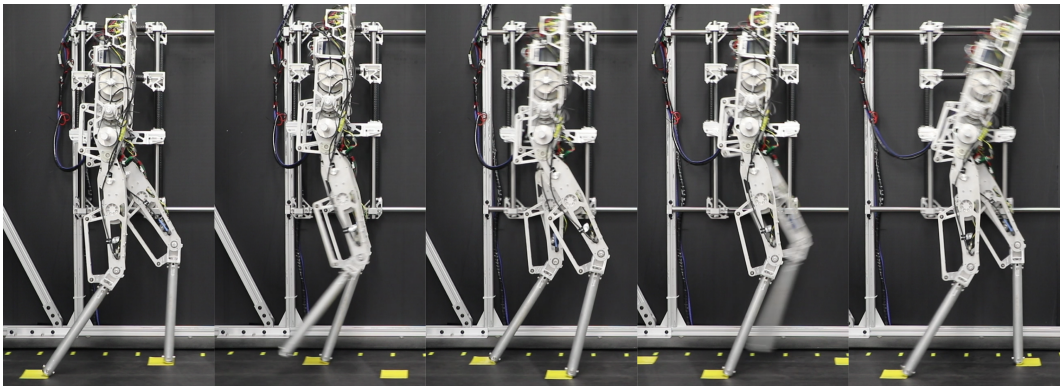


Figure 4.11: Gait tiles for Episode 2 of learning showing the AMBER-3M robot safely traversing a set of stepping stones. Notice the change in step width and added lean of the torso induced by the barrier functions.

The stepping stone configuration was specified to the controller with stones of 8 cm in width. As with simulation, the CBF-QP resulted in a maximum violation of the barriers of 9.2 cm due to model error. After running the PDL algorithm for two episodes, the maximum violation of the barriers was 1.9 cm, only 21% of the original violation, as depicted in Figure 4.10. Although learning improves our estimate of the safe set size, there is still uncertainty in the stone size. This was accommodated by utilizing physical stones which were 10 cm in width. The 7.3 cm reduction in stone size mismatch captures the change from $|\delta|_\infty$ to $|\delta - \widehat{\delta}|_\infty$. Gait tiles for this improved traversal of the stepping stones are shown in Figure 4.11.

Across both simulation and hardware, the learned controller in (SS-QP) remained feasible throughout all episodes, even in the presence of significant modeling error. This empirical observation supports our hypothesis that learning the projected disturbance enables more precise constraint satisfaction without introducing infeasibility. More broadly, these results demonstrate how data-driven techniques can complement model-based design to enforce safety certificates at the low level of the control stack.

4.4 Summary

This chapter demonstrated how low-level controllers can be tune and refined using learning to meet both performance and safety objectives, even in the presence of model uncertainty. The effectiveness of these controllers, however depends on the trajectories that they are tasked with tracking. In the next section, we discuss the design of trajectories that stabilize the underactuated states of the legged robots.

Chapter 5: Low Level Planning Layer

Contents

Intelligent trajectory design can stabilize underactuated robots.

As we turn our attention to the underactuated states of legged robots, we can no longer rely on myopic controllers like Control Lyapunov Functions or Control Barrier Functions for safety and stability. Instead, we require a notion of foresight: planning trajectories over a horizon to see how current actions will effect future states. In this section, we see that intelligent trajectory design (namely through some variant of optimal control) can stabilize underactuated robots.

5.1 Offline Trajectory Generation

All of the results in the previous section leverages a predefined trajectory $\mathbf{x}_d(\cdot)$. Therefore, we begin by discussing how to produce such a trajectory. The first method of producing a desired trajectory $\mathbf{x}_d(\cdot)$ is to perform trajectory optimization once, offline, prior to execution. To do this, we set up the following hybrid optimal control problem:

$$\begin{aligned} V(\mathbf{x}_0) = \min_{\mathbf{u}(\cdot)} \quad & \int_0^T c(\mathbf{x}(t), \mathbf{u}(t))dt \\ \text{s.t.} \quad & \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u}, \quad \mathbf{x}(t) \notin \mathcal{S} \\ & \mathbf{x}^+ = \boldsymbol{\Delta}(\mathbf{x}^-), \qquad\quad \mathbf{x}(t) \in \mathcal{S} \\ & \mathbf{x}(t) \in \mathcal{X}, \\ & \mathbf{u}(t) \in \mathcal{U} \end{aligned} \tag{5.1}$$

where the cost incentivizes efficient forward walking, and the constraints $\mathcal{X}$ and $\mathcal{U}$ ensure the robot picks up its feet and satisfies other physical constraints like joint and torque limits. A trajectory optimization problem as above could produce a single T second long behavior; however, if we want indefinite walking we would have to increase the problem size past what would be tractable. Instead, for periodic behaviors like nominal walking on flat ground, we can leverage the Hybrid Zero Dynamics (HZD) framework [147]–[149]. The HZD method reduces the infinite-time optimal control problem to a finite-time one exactly by enforcing the following periodicity constraint:

$$\mathbf{x}^* = \boldsymbol{\Delta} \circ \boldsymbol{\varphi}_T(\mathbf{x}^*)$$

where the step duration T can be a decision variable. For true HZD theory, we would like to enforce the HZD constraint $\boldsymbol{\Delta}(\mathcal{Z} \cap \mathcal{S}) \subset \mathcal{Z}$, but as trajectory optimization only occurs over a single path, we can only enforce a periodicity constraint (which enforces this impact invariance constraint at a point on $\mathcal{Z}$). Methods to enforce the HZD condition over the entire output zeroing manifold are explored in Section 5.5.

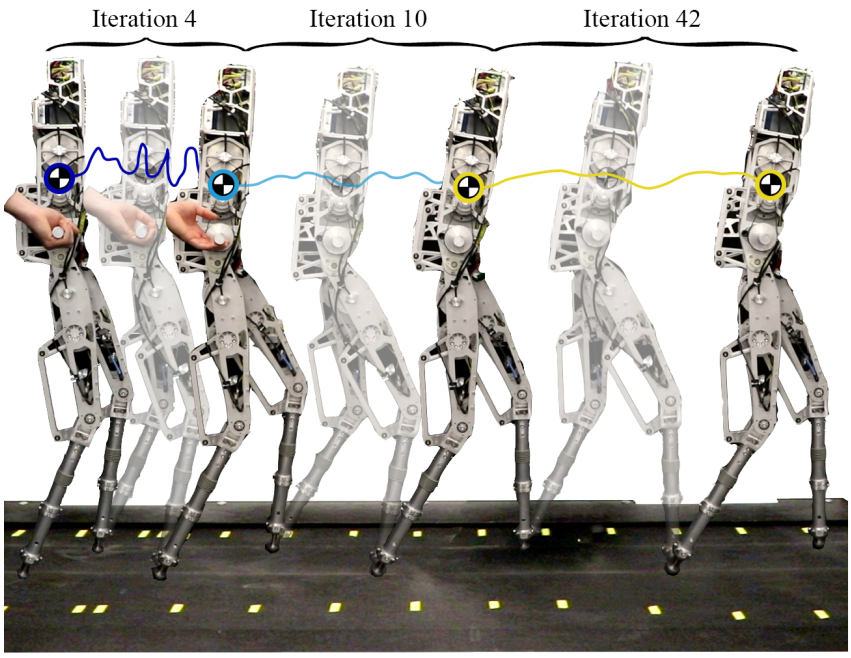


Figure 5.1: Through 50 iterations of experiments, the proposed combination of preference-based learning and HZD optimization transforms failed gaits into robust walking on the AMBER-3M robot with a pair of compliant legs.

Gait Tuning via Preference Based Learning

Given that predefined gaits worked to stabilize the planar biped AMBER, we ask:

> **Question:** *How can we produce desired trajectories that result in the most performant, robust closed loop behavior on hardware?*

As such, we transition to again using preference based learning, but instead of learning the controller gains, returning to PD control and learning the parameters for the trajectory optimization problem. Specifically, we parameterize the constraint set $\mathcal{X}$, and run trials to find the best constraints for walking. Each trial began by initializing AMBER-3M in a static double-support configuration, starting the treadmill, and attempting to push the robot into the designed periodic orbit. If the resultant dynamics were not stable, extra precaution was taken to give the gait the best chance at succeeding. Once the gait reached its orbit, the robot was released and the robustness of the gait to various disturbances was investigated. After both gaits were executed on the physical robot, a preference was collected from the human operator observing the physical realization of the walking. In some iterations, video footage was also reviewed before giving a preference. The criteria used to determine preferences between gaits were the following (in order of prioritization): capable of

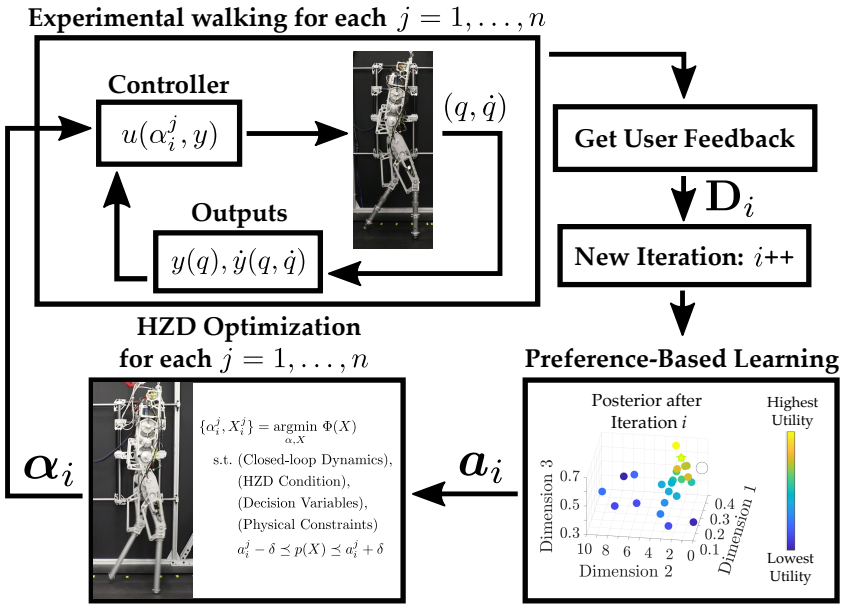


Figure 5.2: The experimental procedure is illustrated in terms of each iteration i with n denoting the number of gaits compared in each iteration. The experiments presented in this section used $n = 2$. Using this notation, the set of n actions given to the HZD optimization is denoted: $\mathbf{a}_i = \{a_i^1, \ldots, a_i^n\}$. The resulting n sets of Bézier coefficients given to the controller are denoted $\boldsymbol{\alpha}_i = \{\alpha_i^1, \ldots, \alpha_i^n\}$.

walking, robust to perturbations in treadmill speeds, robust to external disturbance, does not exhibit harsh noise (e.g., during impact), is visually appealing (intuitive judgment from operator).

In this section, we leverage two configurations of the robot: 1) the point-foot configuration, AMBER3M-PF and 2) the spring-foot configuration, AMBER3M-SF [106]. We first demonstrate the learning framework on AMBER3M-PF, with the corresponding rigid point-foot model used in the gait generation. To emphasize the scalability of our method, we repeat the exact procedure applied to AMBER3M-PF on AMBER3M-SF, but intentionally do not account for changes in the robot model and instead still generate gaits assuming the rigid-body model. Furthermore, we execute the gaits on hardware using the same controller with unmodified gains. Historically, robots with compliance are difficult to generate gaits for because of the resulting complexities which include: increased degrees of freedom of the system; the addition of a double support domain to the hybrid dynamics; and increased stiffness of the dynamics. Past success with compliant bipeds has relied on sophisticated models [150]. Therefore, the fact that our method yields stable walking despite the unmodeled compliance highlights its effectiveness.

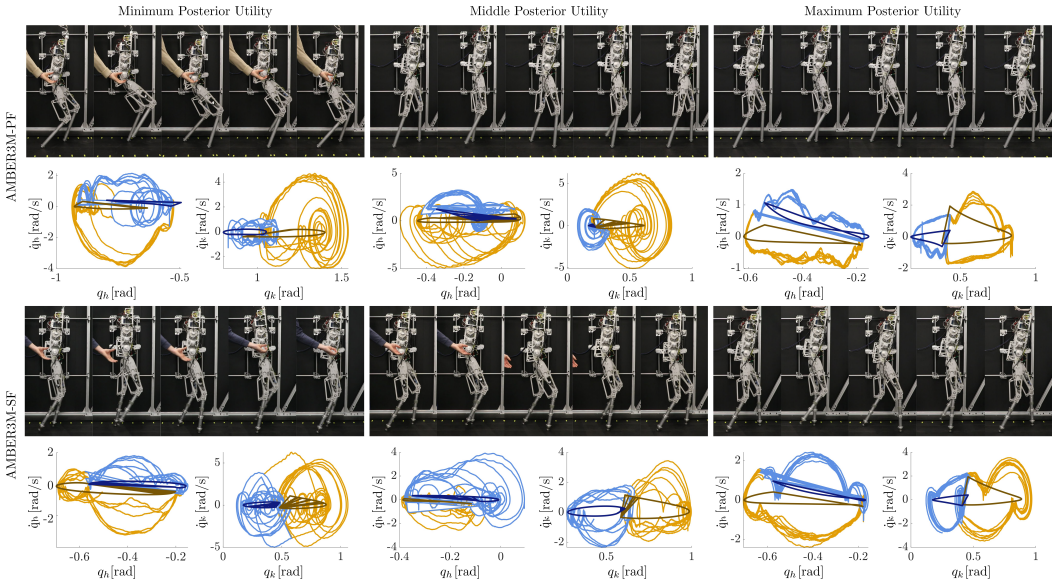


Figure 5.3: Gait tiles with increasing posterior utility values from left to right are shown for the the rigid model (top) and spring model (bottom). The phase portraits of the hip (q_h) and knee (q_k) of the stance leg (blue) and swing leg (yellow) are shown below each corresponding gait, plotted over 10 seconds of data. The phase portraits clearly indicate that for both AMBER3M-PF and AMBER3M-SF the gaits evolved to be more experimentally robust.

A summary of the experimental results is illustrated in the supplementary video [151], with additional videos and material available at [152]. The experiment with AMBER3M-PF was run for 30 iterations and sampled 27 unique gaits. Since gaits quickly met the first criterion of being able to walk, preferences were mainly dictated based on the robustness and appearance of the experimental walking. The initial gaits tried on hardware, although optimal subject to the imposed constraints, resulted in inferior trajectory tracking and power consumption. As the algorithm progressed, the gaits became significantly smoother, more robust to disturbance, and energy efficient. This is exemplified in Fig. 5.3 which illustrates the gaits corresponding to the minimum, a middle, and the maximum posterior utility; the iterations corresponding to when these gaits were first sampled is 1, 21, and 26, respectively. In Fig. 5.3, we note significantly lower velocity overshoot for all of the limbs and tighter tracking shown in the phase portraits for the gaits with higher posterior utility. It is also interesting to note the framework's success at improving the efficiency of the experimental walking: a latent property which is discernible to the human operator even though it is not immediately measured. When the procedure was repeated on AMBER3M-SF, many of the initial gaits were

unable to walk due to the unmodeled compliance. Thus, gaits exhibiting periodic walking were strongly preferred. This second experiment was conducted for 50 iterations and sampled 37 unique gaits. Again, three gaits are selected for further discussion corresponding to the minimum, a middle, and the maximum posterior utility values. Gait tiles and phase portraits for these are again shown in Fig. 5.3. These experiments demonstrate that preference-based learning provides an effective framework for tuning both control gains and trajectory parameters in complex, high-dimensional robotic systems. Across all platforms—ranging from planar bipeds with limited actuation to 3D compliant robots—the approach identified parameter settings that improved robustness, tracking performance, and subjective gait quality, despite significant model mismatch.

5.2 Whole Body Model Predictive Control

While in theory HZD-based gait generation produces provably stable periodic walking, it suffers from several limitations in practice. First, by design it yields a single, fixed trajectory — there is no mechanism for starting stopping, or modulating the behavior without enumerating and precomputing every possible gait offline.

While effective for steady-state behaviors, having a single predefined gait lacks flexibility in dynamic or uncertain environments where online adaptation is required. In the next section, we examine how to extend these ideas by embedding such trajectories within receding-horizon model predictive control (MPC) to enable online replanning and therefore produce stabilizing feedback. Second, tracking an offline generated gait with a naive tracking controller cannot enforce essential constraints such as input bounds or footstep constraints while retaining stability guarantees. In fact, the enforcement of these constraints for the closed loop system needs to be dealt with at design time. Finally, as seen in Section 4.1, HZD relies on the existence of a phasing variable that is a bijection between the center of mass state and time and monotonically increases over time. Such a phasing variable only exists in planar settings, as the lateral motion of walking is monotonic, and, more critically, the center of mass state is two dimensional, barring a bijection from existing with the scalar interval of time. As time-based trajectory tracking is unstable for underactuated systems, as seen in Section 4.1, heuristic regulators are always needed to achieve 3D walking, which once again voids the theoretical guarantees of the method.

Given the complexities of underactuated control, we now transition now from offline gait generation to online gait generation and investigate the following question:

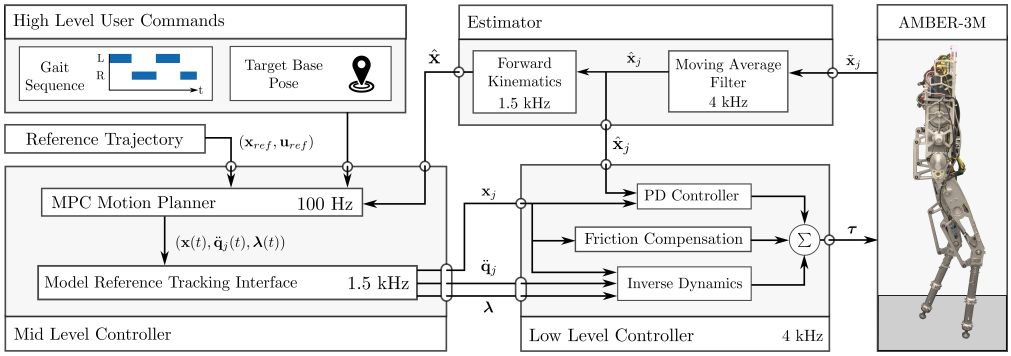


Figure 5.4: Multi-rate control architecture incorporating whole-body planning via MPC and low level tracking controller.

> **Question:** *Does optimal control provide a constructive means of stabilizing underactuated systems?*

We begin by considering full-model model predictive control (MPC) on the AMBER platform, where the planning layer optimizer directly computes reference trajectories for the tracking layer to track.

Nonlinear Model Predictive Control

Nonlinear MPC solves an optimization problem in a receding horizon manner by solving the following finite time nonlinear optimal control problem, similar to that discussed in Section 2.5:

$$\underset{\mathbf{u}(\cdot)}{\text{minimize}} \quad \phi(\mathbf{x}(t_H)) + \int_0^{t_H} l(\mathbf{x}(t), \mathbf{u}(t), t)\mathrm{d}t, \tag{5.2a}$$

$$\text{subject to:} \quad \mathbf{x}(0) = \mathbf{x}_0, \tag{5.2b}$$

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u}, \tag{5.2c}$$

$$\mathbf{x}(t_i^+) = \Delta_c(\mathbf{x}(t_i)), \tag{5.2d}$$

$$\mathbf{h}_{eq}(\mathbf{x}, \mathbf{u}, t) = \mathbf{0}, \tag{5.2e}$$

$$\mathbf{h}_{in}(\mathbf{x}, \mathbf{u}, t) \geq \mathbf{0}, \tag{5.2f}$$

where t_H is the length of the horizon, $\phi : \mathcal{X} \to \mathbb{R}$ is the terminal cost, $l : \mathcal{X} \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}$ is the time-varying running state-input cost, and t_i are times of contact mode transitions. Note that for the following discussion we assume that the contact times t_i are fixed *a priori* based on the desired gait. The optimal control problem is solved in real-time by updating the initial conditions (5.2b) with the measured state of the system. Eq. (5.2c) describes the system dynamics. $\mathbf{h}_{eq} : \mathcal{X} \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^{eq}$ and $\mathbf{h}_{in} : \mathcal{X} \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^{in}$ are generalized path equality

and inequality constraints, respectively. There exist various approaches to solve this problem as outlined in [153]. We take a direct-multiple shooting transcription of the problem together with a sequential quadratic programming approach to handle nonlinearities [154]. Inequality constraints (5.2f) are implemented through relaxed-barrier penalty functions:

$$p(\mathbf{x}) = \begin{cases} -\mu \log(\mathbf{h}_{\text{in}}(\mathbf{x})) & \text{if } \mathbf{h}_{\text{in}}(\mathbf{x}) > \delta \\ -\mu \ln(\delta) + \frac{\mu}{2}\left(\left(\frac{\mathbf{h}_{\text{in}}(\mathbf{x}) - 2\delta}{\delta}\right)^2 - 1\right) & \text{o.w.} \end{cases} \tag{5.3}$$

for parameters $\delta, \mu \geq 0$ as in [155].

Whole-Body Motion Planning & Control

Our nonlinear MPC problem will be constructed using the *OCS2* toolbox [156], which provides convenient interfaces to the *Pinocchio* [157] rigid body library and *CppAd* [158] automatic differentiation tools. Our formulation assumes that the contact schedule associated with a given locomotion mode (standing, stepping in place, walking) is given by the user. The fixed contact schedule assumption simplifies the optimization problem as the sequence of domains and timing of contact mode transitions does not need to be optimized [159], [160]. The position of the foot at contact is captured in the optimization problem through its kinematic relationship with joint coordinates. Moreover, we assume the user provides a desired base pose and velocity to the MPC. In this section we discuss the formulation of bipedal locomotion planning as an MPC problem as posed in (5.2).

System Dynamics

Due to the affine relationship between generalized accelerations $\ddot{\mathbf{q}}$, torques $\boldsymbol{\tau}$, and contact forces $\boldsymbol{\lambda}$ and assuming the torques do not directly impact the floating-base equations of motion, the system dynamics may be rewritten to interpret the joint accelerations $\ddot{\mathbf{q}}_j$ and contact forces $\boldsymbol{\lambda}$, instead of the torques $\boldsymbol{\tau}$, as inputs. The computational benefit of this reparameterization has been shown for reactive whole-body control [161] and offline trajectory optimization [162]. To see this, we write the dynamics in terms of non-actuated base coordinates and fully actuated joint coordinates:

$$\begin{bmatrix} \mathbf{D}_{bb}, & \mathbf{D}_{bj} \\ \mathbf{D}_{bj}^\top & \mathbf{D}_{jj} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}}_b \\ \ddot{\mathbf{q}}_j \end{bmatrix} + \begin{bmatrix} \mathbf{h}_b \\ \mathbf{h}_j \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{B}_j \end{bmatrix} \boldsymbol{\tau} + \begin{bmatrix} \mathbf{J}_{c,b}^\top \\ \mathbf{J}_{c,j}^\top \end{bmatrix} \boldsymbol{\lambda}. \tag{5.4}$$

The base acceleration may be expressed as:

$$\ddot{\mathbf{q}}_b = -\mathbf{D}_{bb}^{-1}\left(\mathbf{h}_b + \begin{bmatrix}\mathbf{D}_{bj} & -\mathbf{J}_{c,b}^{\top}\end{bmatrix}\begin{bmatrix}\ddot{\mathbf{q}}_j \\ \boldsymbol{\lambda}\end{bmatrix}\right), \tag{5.5}$$

and assuming the legs are fully actuated ($\mathbf{B}_j$ is invertible), the corresponding joint torques may be expressed as:

$$\boldsymbol{\tau} = \mathbf{B}_j^{-1}\left(\mathbf{D}_{bj}^{\top}\ddot{\mathbf{q}}_b + \mathbf{h}_j + \begin{bmatrix}\mathbf{D}_{jj} & -\mathbf{J}_{c,j}^{\top}\end{bmatrix}\begin{bmatrix}\ddot{\mathbf{q}}_j \\ \boldsymbol{\lambda}\end{bmatrix}\right), \tag{5.6}$$

maintaining an affine dependence on $\ddot{\mathbf{q}}_j$ and $\boldsymbol{\lambda}$. The base dynamics in (5.5) fully encode the challenge of underactuation and encapsulate the core of the floating-base dynamics. Equation (5.6) plays a secondary role and is only required when formulating torques constraints. We may view the control inputs to optimize over as:

$$\mathbf{u} = \begin{bmatrix}\ddot{\mathbf{q}}_j^{\top}, & \boldsymbol{\lambda}^{\top}\end{bmatrix}^{\top}, \tag{5.7}$$

with the corresponding system dynamics defined as:

$$\dot{\mathbf{x}} = \begin{bmatrix}\dot{\mathbf{q}} \\ \mathbf{D}_{bb}^{-1}\left(-\mathbf{h}_b - \mathbf{D}_{bj}\ddot{\mathbf{q}}_j + \mathbf{J}_{c,b}^{\top}\boldsymbol{\lambda}\right) \\ \ddot{\mathbf{q}}_j\end{bmatrix}. \tag{5.8}$$

In order to avoid large discontinuities in the optimized trajectory, the contact transition maps in (5.2d) have been set to identity maps for the online MPC program, with exponential damping of the contact point velocity after impact being regulated through the stance foot constraint in (5.12), defined in Section 5.2. Due to the assumption of a fixed contact schedule, inclusion of the contact transition map does not fundamentally change the complexity of the optimization problem [160].

Cost Functions

The cost function is formulated as a nonlinear least square cost around a given state and input reference trajectory. To that end we define the set of tracking errors as follows:

$$\boldsymbol{\epsilon}_x = \mathbf{x} - \mathbf{x}_{\text{ref}}(t),\ \boldsymbol{\epsilon}_u = \mathbf{u} - \mathbf{u}_{\text{ref}}(t),\ \boldsymbol{\epsilon}_i = \begin{bmatrix}\mathbf{p}_i - \mathbf{p}_{i,\text{ref}}(t) \\ \mathbf{v}_i - \mathbf{v}_{i,\text{ref}}(t) \\ \mathbf{a}_i - \mathbf{a}_{i,\text{ref}}(t),\end{bmatrix},$$

where $\mathbf{x}_{\text{ref}}$ is the state reference, $\mathbf{u}_{\text{ref}}$ is the input reference, and $\mathbf{p}_i, \mathbf{v}_i, \mathbf{a}_i \in \mathbb{R}^3$ with $i \in \{1, 2\}$ are the Cartesian position, velocity, and accelerations of the i^{th} foot, with

corresponding references $\mathbf{p}_{i,\text{ref}}, \mathbf{v}_{i,\text{ref}}, \mathbf{a}_{i,\text{ref}}$. The references $\mathbf{x}_{\text{ref}}$ and $\mathbf{u}_{\text{ref}}$ are defined heuristically (see Section 5.2) or via a walking gait synthesized offline using HZD. The running state-input cost l is given by:

$$l(\mathbf{x}, \mathbf{u}, t) = \frac{1}{2}\boldsymbol{\epsilon}_x^\top \mathbf{Q} \boldsymbol{\epsilon}_x + \frac{1}{2}\boldsymbol{\epsilon}_u^\top \mathbf{R} \boldsymbol{\epsilon}_u + \frac{1}{2}\sum_i \boldsymbol{\epsilon}_i^\top \mathbf{W} \boldsymbol{\epsilon}_i, \tag{5.9}$$

where $\mathbf{Q}, \mathbf{R}$, and $\mathbf{W}$ are positive definite weighting matrices.

To pick an appropriate weighting for the terminal cost, we approximate the infinite horizon cost by solving an unconstrained Linear Quadratic Regulator (LQR) problem using a linear approximation of the dynamics and a quadratic approximation of the running costs (5.9) around the nominal stance configuration of the robot. The positive definite Riccati matrix $\mathbf{P}_{\text{LQR}}$ of the cost-to-go is used to define the quadratic cost around the terminal reference state:

$$\phi(\mathbf{x}) = \frac{\rho}{2}\boldsymbol{\epsilon}_x(T)^\top \mathbf{P}_{\text{LQR}} \boldsymbol{\epsilon}_x(T), \tag{5.10}$$

where $\rho > 0$ is a hyperparameter. Setting $\rho = 1.0$ would express approximately equal importance of the integrated running cost and terminal cost, and $\rho \to \infty$ would make the terminal cost behave as an equality constraint. We found good performance for the heuristic reference at $\rho = 1.0$ and for the HZD reference at $\rho = 10.0$. Note that this cost does not penalize deviation from the stance configuration used to produce the LQR solution, but rather provides a systematic way to scale the relative importance of all of the degrees of freedom of the robot in the cost.

Reference Trajectories

A key component in establishing closed loop stability and recursive feasibility is the choice of terminal components [163], either as terminal cost in (5.2a) or as constraints on the terminal state, $\mathbf{x}(t_H)$, to lie in a control invariant set. In practice, for nonlinear complex systems, it is challenging to prove that such conditions hold. Extending the prediction horizon is a common choice to reduce the relative importance of the terminal components [164]. However, for systems where long prediction horizons are not feasible due to computational limits, careful choice of terminal components directly translates to the overall performance of the controller, as we will empirically show in this work.

HZD Trajectory HZD state and input reference trajectories, $\mathbf{x}_{\text{ref}}(t)$ and $\mathbf{u}_{\text{ref}}(t)$, are found offline for the whole-body nonlinear dynamics using the FROST toolbox

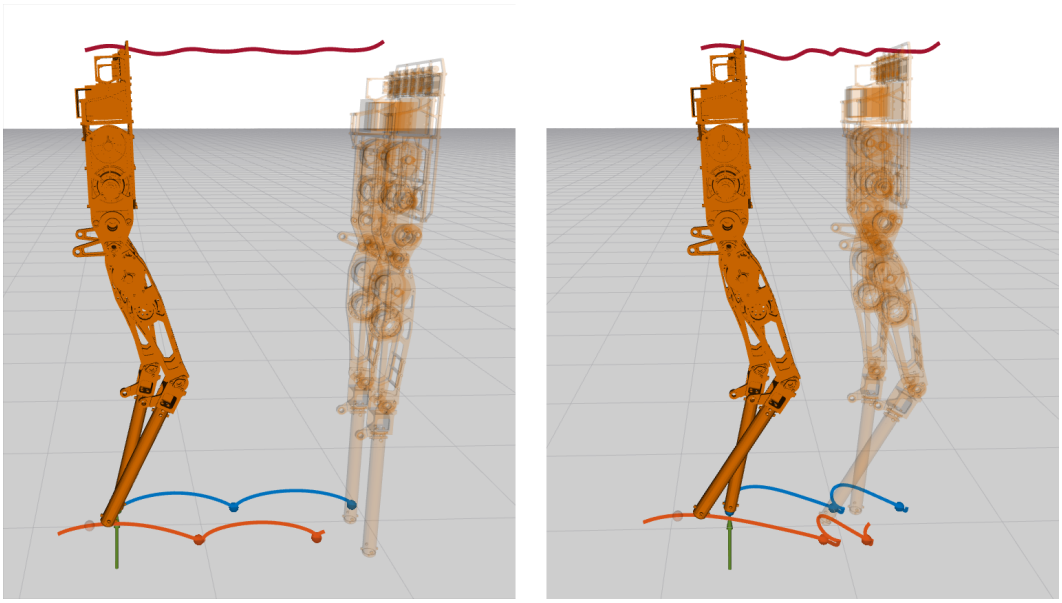


Figure 5.5: Heuristic (left) and HZD (right) terminal states.

[165] and stored as Bézier polynomials. This process is completed by fixing a target gait sequence and a forward velocity, and adding various other state and input constraints to a nonlinear trajectory optimization program which ensure the underactuated dynamics of the system display stable periodic behavior. For planar systems, stability can be enforced directly in the optimization program [166], and for general systems it can be verified *a posteriori* via the Poincaré return map [147]. The foot references $\mathbf{p}_{i,\text{ref}}(t)$, $\mathbf{v}_{i,\text{ref}}(t)$, and $\mathbf{a}_{i,\text{ref}}(t)$ are entirely determined by $\mathbf{x}_{\text{ref}}(t)$.

Heuristic Trajectory To evaluate the relative impact of using a gait synthesized offline via HZD in the cost function, we produce a heuristic reference trajectory to be compared against. In particular, the state trajectory $\mathbf{x}_{\text{ref}}(t)$ is composed of a user-commanded base pose and velocity, and a static nominal joint configuration. The input reference $\mathbf{u}_{\text{ref}}(t)$ is defined with zero joint accelerations and contact forces that are evenly distributed among each foot in contact in the nominal joint configuration such that the weight of the robot is compensated. The foot references $\mathbf{p}_{i,\text{ref}}(t)$, $\mathbf{v}_{i,\text{ref}}(t)$, and $\mathbf{a}_{i,\text{ref}}(t)$ are designed by extracting the nominal touchdown and liftoff locations below the hip at the middle of the contact phase and fitting a smooth hand-designed swing reference trajectory. The heuristic and HZD-based terminal state are visualized in Fig. 5.5.

By construction, both the HZD gaits and the gait sequences defined by MPC have

an associated *phasing variable*, i.e., a parameter in the interval $[0, 1]$ which monotonically increases over the step. During execution, the phasing variable is attained from the current MPC gait sequence that the robot is in, and is used to construct $\mathbf{x}_{\text{ref}}$ and $\mathbf{u}_{\text{ref}}$ over the MPC horizon. In this section, we use a time-based phasing variable from MPC to interpolate the HZD trajectory.

Remark: *In phase-based HZD control, stability is achieved via implicit modification of the contact times, whereas in MPC, stability is achieved via explicit modification of the footstep locations.*

Constraints

The following constraints are imposed in problem (5.2). All inequality constraints are implemented as soft constraints with the relaxed log barrier functions in (5.3).

Gait-Dependent Constraints These constraints capture the different modes of each leg at any given point in time determined by the specified gait sequence. We enforce the user-defined gait and avoid foot scuffing of a swing leg by constraining the swing foot motion in the orthogonal direction to the ground surface, $\mathbf{n} \in \mathbb{R}^3$, to follow the Cartesian reference trajectory:

$$\mathbf{n}^\top(\mathbf{a}_i - \mathbf{a}_{i,\text{ref}}(t) + k_d(\mathbf{v}_i - \mathbf{v}_{i,\text{ref}}(t)) + k_p(\mathbf{p}_i - \mathbf{p}_{i,\text{ref}}(t))) = \mathbf{0} \tag{5.11}$$

where $k_d, k_p \in \mathbb{R}_{\geq 0}$ are feedback gains chosen to achieve asymptotic tracking. The foot position in the direction parallel to the ground is not directly constrained; rather, tracking is enforced via the cost function described in Section 5.2. For a stance leg we enforce a stationarity constraint in Cartesian space through:

$$\mathbf{a}_i + k_d\mathbf{v}_i = \mathbf{0}. \tag{5.12}$$

Contact Force Constraints The following constraints require the contact forces at each foot to match the designation of swing and stance legs:

$$\begin{cases} \boldsymbol{\lambda}_i = \mathbf{0}, & i \text{ is a swing leg} \\ \boldsymbol{\lambda}_i \in \mathcal{C}(\mathbf{n}, \mu_c), & i \text{ is a stance leg.} \end{cases} \tag{5.13}$$

The first constraint requires no contact force from a swing leg, as it does not contact the ground. The second constraint requires the contact force of a stance leg to lie in the friction cone $\mathcal{C}(\mathbf{n}, \mu_c)$ defined by the surface normal $\mathbf{n}$ and the friction coefficient $\mu_c = 0.6$. This second-order cone constraint is expressed in the local surface-aligned frame:

$$\mu_c\lambda_{i,3} - \sqrt{\lambda_{i,1}^2 + \lambda_{i,2}^2} \geq 0. \tag{5.14}$$

Only linear forces are present as the robot has point feet, and that the friction constraint also enforces a unilateral contact constraint as it requires $\lambda_{i,3} \geq 0$. Note that the complementarity constraint $\boldsymbol{\lambda}^\top \mathbf{p}_{\text{sw}} = 0$ requiring the product of the reaction forces and height of the swing foot to be zero (which is bilinear and challenging to optimize over) gets resolved by enforcing a fixed contact schedule.

Joint and Torque Limit Constraints The joint coordinates and joint coordinate velocities are enforced to lie in the set of minimum and maximum joint positions and velocities through state inequality constraints: $\mathbf{x} \in [\mathbf{x}_{\text{min}}, \mathbf{x}_{\text{max}}]$. Similarly, the joint torques can be computed by Eq. (5.6) and should lie within joint torque limits $\boldsymbol{\tau} \in [\boldsymbol{\tau}_{\text{min}}, \boldsymbol{\tau}_{\text{max}}]$.

Low Level Controller

As shown in Fig. 5.4, the state and input trajectories generated by MPC are interpolated at a high frequency and converted to a feed-forward control torques, $\boldsymbol{\tau}_{\text{MPC}}$, via (5.6). As the feed-forward torque is model-based, we compensate for model errors when executing the controller on hardware by adding a PD torque, $\boldsymbol{\tau}_{\text{PD}}$, and a friction compensation torque, $\boldsymbol{\tau}_{\text{FC}}$, to the feed-forward torque:

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{\text{MPC}} + \boldsymbol{\tau}_{\text{PD}} + \boldsymbol{\tau}_{\text{FC}}. \tag{5.15}$$

The friction compensated torque $\boldsymbol{\tau}_{\text{FC}}$ was a simple regression model fit to data collected from sinusoidal motions of the legs, which helped overcome static friction in the gearbox. The total torque $\boldsymbol{\tau}$ is send to the open loop torque controlled BLDC motors.

The time discretization in the multiple shooting scheme is set to 15ms and we allow for a maximum of 10 SQP iterations per MPC problem. All planning, control, and estimation loops were done on separate threads on an offboard Ryzen 9 5950x CPU @ 3.4GHz. Benchmarks of the maximum obtainable MPC frequency for different horizon lengths can be seen in Table 5.1. To isolate how the system's behavior depends on horizon length, all experiments were conducted with a consistent MPC frequency of 100Hz.

Table 5.1: MPC Planning Frequency (10 SQP Iterations)

Horizon Length [s]	2.0	1.0	0.5	0.2
MPC Frequency [Hz]	270	480	670	850

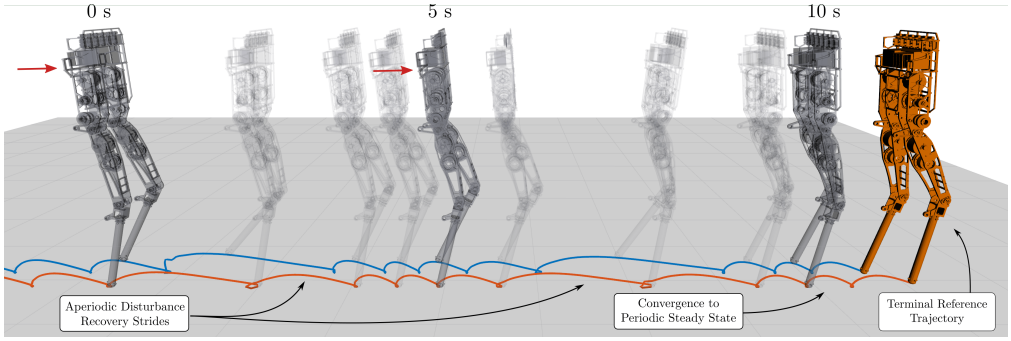


Figure 5.6: Push recovery using the proposed method under a disturbance — notice the aperiodic stepping that was planned online in order to reject the disturbance, something that is not possible with traditional HZD based methods.

As can be seen in the supplementary video [167], the proposed MPC formulation is capable of simultaneously stabilizing the underactuated system dynamics and synthesizing valid motion trajectories for a broad range of gait pattern and target velocities both in simulation and on hardware. To evaluate the effect of changing reference signals on the feasibility and robustness of the full control pipeline, a sequence of disturbances of increasing magnitude was applied in simulation with the following MPC configurations:

- *MPC with No Terminal*: The proposed whole-body MPC with heuristic references for the running cost (refer to Sec.5.2) and no terminal cost.
- *MPC with Heuristic Terminal*: Same as above, but with heuristic references included as a terminal cost.
- *MPC with HZD Reference*: The proposed whole-body MPC with HZD-based references for the running and terminal cost (refer to Sec. 5.2).
- *Lumped Mass MPC*: Uses a simplified dynamics model for the planning stage by moving leg inertia to the torso, otherwise identical to *MPC with Heuristic Terminal*.
- *HZD with PD*: An offline generated HZD trajectory tracked by a joint level PD controller.

The results of these simulations are summarized in Table 5.2. First, we remark that the Lumped Mass MPC model was introduced to highlight the effects of planning over the full system dynamics for the given platform. The particular structure of

this model was chosen to resemble some properties of the simplified models while allowing for an implementation independent comparison. Although the Lumped Mass MPC could withstand similar disturbances to the whole-body MPC for a specified standing position, it was observed to have only a marginal ability to reject disturbances during dynamic motions like stepping in place and walking, no matter the horizon length. This is likely attributable to the fact that the AMBER-3M platform has approximately 40% of its mass in the legs, and confirms the need for whole-body online planning methods, especially for robots like AMBER-3M which have a non-negligible mass distribution concentrated in the legs.

Next, note that the MPC approach fails quickly when no terminal cost is present. When a heuristic terminal component is added, the robustness of the system dramatically increases. Furthermore, when the proposed MPC approach is combined with an HZD-based reference trajectory for running and terminal costs, the horizon length can be shortened to as low as 0.2 seconds, which drastically reduces the computational complexity. This could be essential to enable the whole body NMPC approach to be applied to a 3D biped with a larger number of degrees of freedom. These results emphasize the importance of the careful design of reference components, as their construction is tightly coupled with the performance of the overall system. Finally, it is important to note that at a disturbance of 22N during walking the foot begins to slip, causing all of the MPC based methods to fail. The HZD with PD method exhibits more robustness to foot slipping and is therefore able to endure larger disturbances, as it does not model the disturbances. Note that the HZD and PD method is limited to periodic motions, and during disturbance rejection it heavily restricts the allowable stepping range, as reported in Table 5.2. On the other hand, the MPC methods naturally have a large variability in footstep locations in

Table 5.2: Maximum disturbance rejection and step adaption range (difference between smallest and largest observed step length). MPC planning frequency clamped at 100 Hz.

	Disturbance Rejection			Step Range
Horizon Length	2 s	0.5 s	0.2 s	2 s
Lumped Mass MPC	2 N	-	-	-
MPC + No Terminal	22 N	-	-	0.63 m
MPC + Heuristic	22 N	22 N	-	0.67 m
MPC + HZD	22 N	22 N	20 N	1.10 m
HZD + PD	30 N			0.14 m

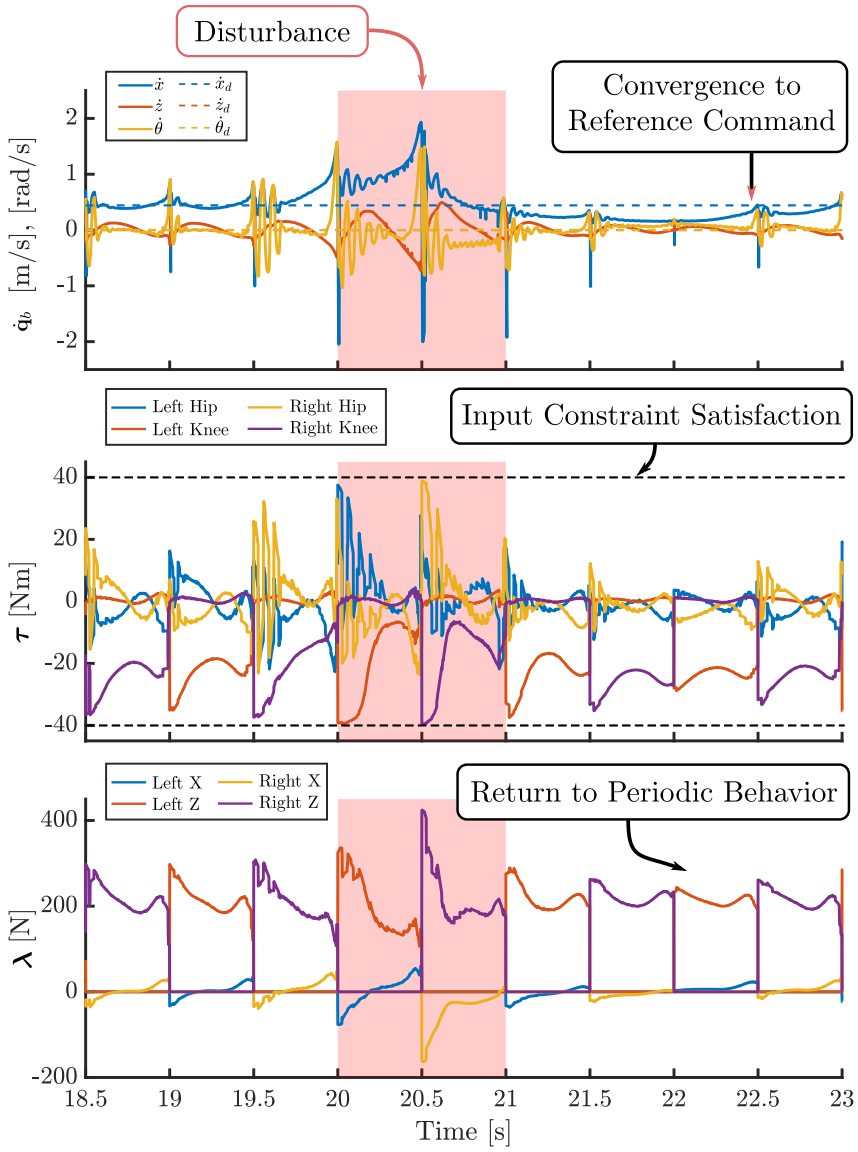


Figure 5.7: Simulation results for the MPC on AMBER with Heuristic Terminal controller under a disturbance of 20N applied in the forward (X) direction during the marked time of 1s, including states (top), torques (middle), and contact forces (bottom). The commanded forward walking velocity is 0.5m/s.

order to stabilize the system. We believe that the ability to modulate step width and exhibit aperiodic motions will be critical for bipedal robots operating on real-world terrain.

As seen in the supplementary video [167], the various proposed approaches react differently to disturbances. Specifically, the phase-based HZD with PD control achieves stability via implicit modification of the contact times, where the limbs

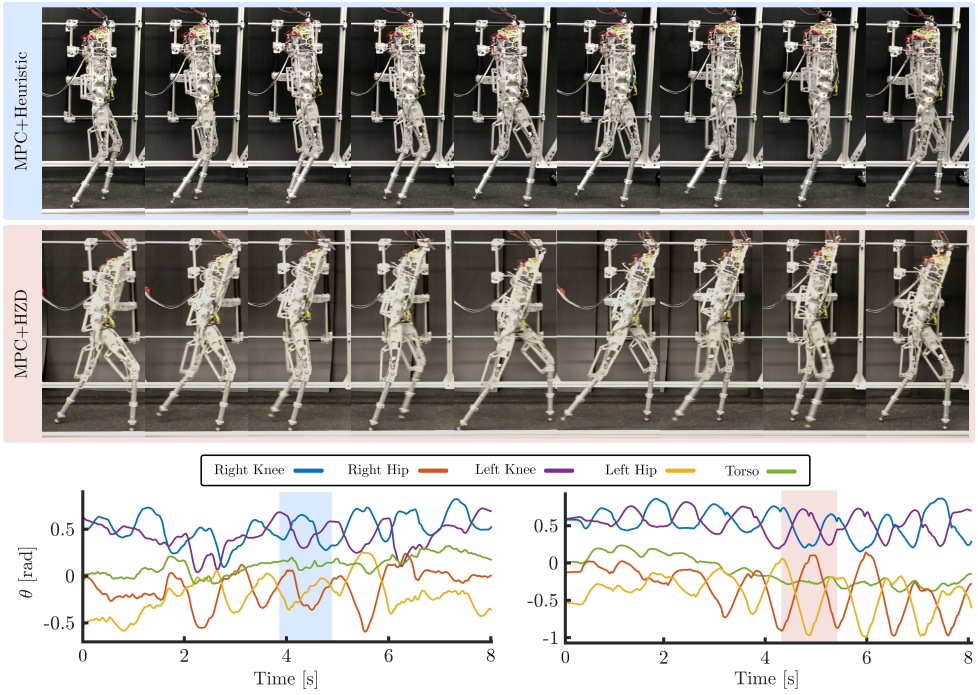


Figure 5.8: Gait tiles and joint angle trajectories for forward walking behavior of the whole body MPC at a horizon length of 1 second (top, left), and the whole body MPC+HZD at a horizon length of 0.5 seconds (bottom, right). The HZD reference induces stronger periodic behaviors in the joint coordinates, correlated with the periodic nature of an HZD gait.

are accelerated along the predefined reference trajectory. While this allows for significant disturbance rejection, it leads to the inputs being saturated for non-negligible amounts of time. On the other hand, in this MPC-based formulation, stability is achieved via explicit modification of the footstep locations, and is able to converge back to the desired reference trajectory in one to two steps while still satisfying state and input constraints. A depiction of the disturbance rejection behavior of the MPC method can be seen in Fig. 5.7.

The MPC with a heuristic reference trajectory and a horizon length of 1.0 second, and the MPC with an HZD trajectory and a horizon length of 0.5 seconds were then deployed on the AMBER hardware. As seen in Fig. 5.8, both methods produce forward walking and have a visually distinct gait. We see in the joint angle trajectory data that the MPC with HZD method displays strong periodic behavior, similar to periodic motions expected with an HZD approach.

5.3 Model Predictive Control on Manifolds

A key challenge in applying MPC to floating-base legged robots is that their configuration space is not Euclidean. In the previous section, we were able to ignore this complexity by planning over Euler angles, which are a convenient parameterization as long as the robot remains away from the singularity. In this section, we explicitly operate on the underlying manifold structure to avoid these singularities, and address the question:

> **Question:** *How do we perform optimal control when the states are manifold valued?*

In particular, the robot's orientation evolves on the manifold $\mathcal{S}^3$, requiring special care in defining both the system dynamics and the optimization variables. Directly including quaternion or rotation matrix representations in an MPC formulation introduces nonlinearities that can make the problem intractable in real time, and conversely ignoring the underlying manifold constraint can produce trajectories that are not dynamically feasible.

To address this, we introduce a change of coordinates that lifts the orientation dynamics into the Lie algebra $\mathfrak{s}^3$ — a vector space — enabling locally linear approximations using Taylor approximations to be preformed. This representation allows us to formulate a convex quadratic program for trajectory optimization over short horizons, while still respecting the underlying manifold structure.

Linearized Dynamics

There are a number of potential options to produce solutions for the orientation variable $q \in \mathcal{S}^3$. The first option is to take the continuous time dynamics as in Section 2.1:

$$\dot{q} = q\boldsymbol{\omega},$$

and linearize them directly in the ambient space $\mathbb{R}^4$. Once these dynamics have been integrated, the solution $q(t)$ could be renormalized to bring it back to the constraint surface $\mathcal{S}^3$. Depending on the integration time, however, this will likely cause extremely poor dynamics approximations as a significant portion of the path length will be lost to the projection operation.

For the next option, recall from Section 2.1 that a Lie Euler step over a time discretization h is of the form:

$$q_{k+1} = q_k \exp(\boldsymbol{\omega}_k h). \tag{5.16}$$

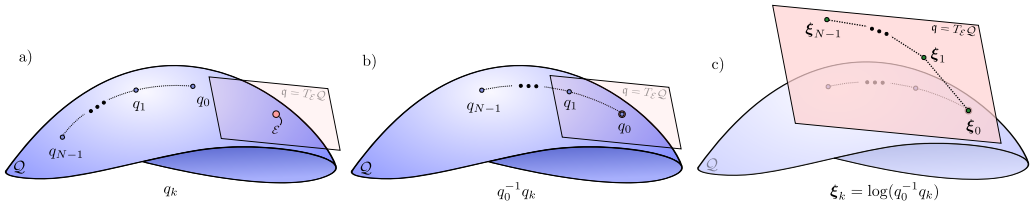


Figure 5.9: A depiction of Lie groups, Lie algebras, and the $\log$ operation. a) The trajectory q_k, b) pulling the trajectory back to the identity element via q_0^{-1}, and c) taking the $\log$ map near identity to obtain elements in the Lie algebra.

These dynamics could be linearized directly whereby producing perturbation vectors in the tangent space at q_k would yield trajectories which satisfy the underlying manifold structure. Unfortunately, the $\exp$ operation is complex, and its linearization can be difficult.

For the final option, the one used for the remainder of this section, in order to avoid the nonlinearity present in (5.16), we propose the following change of coordinates:

$$\boldsymbol{\xi}_k = \log(q_0^{-1} q_k), \tag{5.17}$$

which first pulls the variables back to the vicinity of the identity element q_ε, and then to the Lie algebra as shown in Figure 5.9. Substituting in the Lie-Euler step from (2.10) into the above expression for the first few values of k, we have:

$$\begin{aligned}
\boldsymbol{\xi}_0 &= \mathbf{0} \\
\boldsymbol{\xi}_1 &= \log(q_0^{-1} q_1) = \log(q_0^{-1} q_0 \exp(\boldsymbol{\omega}_0 h)) \\
&= \boldsymbol{\omega}_0 h \\
\boldsymbol{\xi}_2 &= \log(q_0^{-1} q_2) = \log(q_0^{-1} q_1 \exp(\boldsymbol{\omega}_1 h)) \\
&= \log(q_0^{-1} q_0 \exp(\boldsymbol{\omega}_0 h) \exp(\boldsymbol{\omega}_1 h)) \\
&= \boldsymbol{\omega}_0 h + \boldsymbol{\omega}_1 h + \frac{1}{2}[\boldsymbol{\omega}_0 h, \boldsymbol{\omega}_1 h] + O(h^3)
\end{aligned}$$

where $[\cdot, \cdot]$ represents the Lie bracket, the last line follows from the Campbell-Baker-Hausdorff theorem [168], and the higher order terms consist of linear combinations of iterated Lie brackets, which due to linearity are multiplied by terms of h^3 or higher[1]. This means that, neglecting terms of $O(h^2)$ or higher, we are able to write our dynamics update law as:

$$\boldsymbol{\xi}_{k+1} = \sum_{i=0}^{k} \boldsymbol{\omega}_k h = \boldsymbol{\xi}_k + \boldsymbol{\omega}_k h,$$

[1] If quaternion multiplication commuted, then the iterated Lie brackets would be zero — but alas, no such pleasantries exist.

which is linear and will therefore be straightforward to include in the MPC program.

The next challenge concerns the construction of local approximations of the acceleration dynamics. As most of the coordinates lie in Euclidean space and therefore have straightforward Taylor approximations, we will limit our attention on the manifold valued variables. Specifically, in a continuous domain v consider a function $f : S^3 \times \mathfrak{s}^3 \times \mathbb{R}^m \to \mathfrak{s}^3$ satisfying:

$$\dot{\boldsymbol{\omega}} = f(q, \boldsymbol{\omega}, \mathbf{u}).$$

Given a perturbation $\boldsymbol{\eta} \in \mathfrak{s}^3$, a local representation of f in exponential coordinates $\tilde{f} : S^3 \times \mathfrak{s}^3 \times \mathbb{R}^m \times \mathfrak{s}^3 \to \mathbb{R}^3$ can be defined as:

$$\tilde{f}(q, \boldsymbol{\omega}, \mathbf{u}, \boldsymbol{\eta}) \triangleq f(q \exp(\boldsymbol{\eta}), \boldsymbol{\omega}, \mathbf{u}).$$

Given $q \in S^3$, $\boldsymbol{\omega} \in \mathfrak{s}^3$, and $\mathbf{u} \in \mathbb{R}^m$, as well as additional perturbations $\Delta\boldsymbol{\omega} \in \mathbb{R}^3$ and $\Delta\mathbf{u} \in \mathbb{R}^m$, we can compute a Taylor expansion of $\tilde{f}$ about the point $(q, \boldsymbol{\omega}, \mathbf{u}, \mathbf{0})$ at a perturbed point $(q, \boldsymbol{\omega} + \Delta\boldsymbol{\omega}, \mathbf{u} + \Delta\mathbf{u}, \boldsymbol{\eta})$ via:

$$\begin{aligned} &f(q \exp(\boldsymbol{\eta}), \boldsymbol{\omega} + \Delta\boldsymbol{\omega}, \mathbf{u} + \Delta\mathbf{u}) \\ &\qquad \approx f(q, \boldsymbol{\omega}, \mathbf{u}) + \frac{\partial \tilde{f}}{\partial \boldsymbol{\eta}} \cdot \boldsymbol{\eta} + \frac{\partial f}{\partial \boldsymbol{\omega}} \cdot \Delta\boldsymbol{\omega} + \frac{\partial f}{\partial \mathbf{u}} \cdot \Delta\mathbf{u}. \end{aligned}$$

Then, we can write the continuous-time linearized dynamics of $\boldsymbol{\omega}$ about the point $(q, \boldsymbol{\omega}, \mathbf{u})$ as:

$$\frac{\mathrm{d}}{\mathrm{dt}}\delta\boldsymbol{\omega} = \frac{\partial \tilde{f}}{\partial \boldsymbol{\eta}}(q, \boldsymbol{\omega}, \mathbf{u}, \mathbf{0}) \cdot \boldsymbol{\eta} + \frac{\partial f}{\partial \boldsymbol{\omega}} \cdot \delta\boldsymbol{\omega} + \frac{\partial f}{\partial \mathbf{u}} \cdot \delta\mathbf{u}. \tag{5.18}$$

Next, we consider the dynamics of the variables $\boldsymbol{\xi}_k$ around a reference trajectory $\bar{q}_k \in S^3$, $\bar{\boldsymbol{\omega}}_k \in \mathfrak{s}^3$, and $\bar{\mathbf{u}}_k \in \mathbb{R}^m$. Define $\bar{\boldsymbol{\xi}}_k = \log(\bar{q}_0^{-1}\bar{q}_k)$ with $\bar{\boldsymbol{\xi}}_0 = \mathbf{0}$ and suppose the reference trajectory satisfies the Lie-Euler step, i.e.:

$$\bar{\boldsymbol{\xi}}_{k+1} = \bar{\boldsymbol{\xi}}_k + \bar{\boldsymbol{\omega}}_k h.$$

For a trajectory $\boldsymbol{\xi}_k \in \mathfrak{s}^3$ similarly satisfying the Lie-Euler step, and vectors $\boldsymbol{\omega}_k \in \mathfrak{s}^3$, and $\mathbf{u}_k \in \mathbb{R}^m$, we have:

$$(\boldsymbol{\xi}_{k+1} - \bar{\boldsymbol{\xi}}_{k+1}) = (\boldsymbol{\xi}_k - \bar{\boldsymbol{\xi}}_k) + (\boldsymbol{\omega}_k - \bar{\boldsymbol{\omega}}_k)h + \underbrace{\bar{\boldsymbol{\xi}}_k + \bar{\boldsymbol{\omega}}_k h - \bar{\boldsymbol{\xi}}_{k+1}}_{=\mathbf{0}},$$

yielding continuous-time linearized dynamics:

$$\frac{\mathrm{d}}{\mathrm{dt}}\delta\boldsymbol{\xi} = \delta\boldsymbol{\omega}. \tag{5.19}$$

Combining expressions (5.18) and (5.19), we obtain:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\delta\boldsymbol{\xi}\\ \delta\boldsymbol{\omega}\end{bmatrix} = \underbrace{\begin{bmatrix}\mathbf{0} & \mathbf{I}\\ \frac{\partial \tilde{f}}{\partial \boldsymbol{\eta}}(q,\boldsymbol{\omega},\mathbf{u},\mathbf{0}) & \frac{\partial f}{\partial \boldsymbol{\omega}}\end{bmatrix}}_{\mathbf{A}}\begin{bmatrix}\delta\boldsymbol{\xi}\\ \delta\boldsymbol{\omega}\end{bmatrix} + \underbrace{\begin{bmatrix}\mathbf{0}\\ \frac{\partial f}{\partial \mathbf{u}}\end{bmatrix}}_{\mathbf{B}}\delta\mathbf{u}. \tag{5.20}$$

We can similarly construct local approximations of the impact maps. On an edge e, consider a function $\Delta : \mathcal{S} \to \mathfrak{s}^3$ which satisfies:

$$\boldsymbol{\omega}^+ = \Delta(q^-, \boldsymbol{\omega}^-). \tag{5.21}$$

Here, $\mathcal{S}$ represents the guard as a submanifold of $S^3 \times \mathfrak{s}^3$ (though the guard is actually a submanifold of $\mathcal{Q}$). As defined, this reset map is the restriction of the momentum transfer of the system at impact to the guard. Therefore, we can naturally extend the domain of the reset map by considering the same momentum transfer applied anywhere in the state space, yielding $\Delta_{\text{ext}} : S^3 \times \mathfrak{s}^3 \to \mathfrak{s}^3$. This is needed because in our Taylor expansion of the discrete dynamics, we consider perturbations of the full system state (not just perturbations tangent to the guard)[2]. As before, we can locally approximate the function Δ_{ext} via:

$$\begin{aligned}\Delta_{\text{ext}}(q^- \exp(\boldsymbol{\eta}), \boldsymbol{\omega}^- + \Delta\boldsymbol{\omega}^-) \approx{}& \Delta_{\text{ext}}(q^-, \boldsymbol{\omega}^-)\\ &+ \frac{\partial \tilde{\Delta}_{\text{ext}}}{\partial \boldsymbol{\eta}}(q^-, \boldsymbol{\omega}^-, \mathbf{0}) \cdot \boldsymbol{\eta} + \frac{\partial \Delta_{\text{ext}}}{\partial \boldsymbol{\omega}^-} \cdot \Delta\boldsymbol{\omega}^-,\end{aligned}$$

where again we can represent Δ_{ext} locally as:

$$\tilde{\Delta}_{\text{ext}}(q^-, \boldsymbol{\omega}^-, \boldsymbol{\eta}) \triangleq \Delta_{\text{ext}}(q^- \exp(\boldsymbol{\eta}), \boldsymbol{\omega}^-).$$

Noting that the $q^+ = q^-$, we can represent the linearization of the discrete map as:

$$\begin{bmatrix}\delta\boldsymbol{\xi}^+\\ \delta\boldsymbol{\omega}^+\end{bmatrix} = \underbrace{\begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \frac{\partial \tilde{\Delta}_{\text{ext}}}{\partial \boldsymbol{\eta}}(q^-, \boldsymbol{\omega}^-, \mathbf{0}) & \frac{\partial \Delta_{\text{ext}}}{\partial \boldsymbol{\omega}^-}\end{bmatrix}}_{\mathbf{D}}\begin{bmatrix}\delta\boldsymbol{\xi}^-\\ \delta\boldsymbol{\omega}^-\end{bmatrix}. \tag{5.22}$$

Geometric Model Predictive Control for ARCHER

This section represents the planning layer of the level, as shown in Figure 5.10. High level target base positions $\mathbf{p}_{\text{ref}} \in \mathbb{R}^3$ are provided by the user, and MPC produces reference trajectories to pass to the low layer. This architecture maintains the benefit

[2]This extension allows us to abscond from having to only consider perturbations along the guard, which is an interesting area of future work.

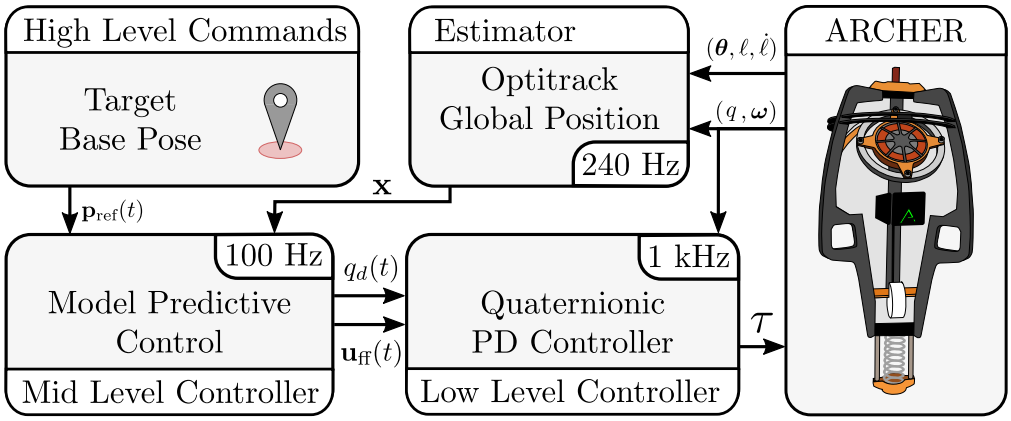


Figure 5.10: The multi-rate architecture employed for the robot.

of having a horizon and is paired with a low layer feedback controller which adds robustness to model error and delays induced by computation time.

The configuration of the hopping robot is given by $\mathbf{q} = (\mathbf{p}, q, \boldsymbol{\theta}, \ell) \in \mathcal{Q}$, where $\mathbf{p} \in \mathbb{R}^3$ is the Cartesian position, $q \in S^3$ is the unit quaternion representing the orientation, $\boldsymbol{\theta} \in \mathbb{R}^3$ represents the flywheel angles, and $\ell \in \mathbb{R}$ is the foot deflection. Next, let $\mathbf{v} = (\dot{\mathbf{p}}, \boldsymbol{\omega}, \dot{\boldsymbol{\theta}}, \dot{\ell}) \in \mathcal{V} \triangleq \mathbb{R}^3 \times \mathfrak{s}^3 \times \mathbb{R}^3 \times \mathbb{R}$, where $\boldsymbol{\omega} \in \mathfrak{s}^3$ is a purely imaginary quaternion representing the angular rate of the body.

For a given MPC horizon $N \in \mathbb{N}$, we begin by constructing a vertex sequence $v_k \in V$ for $k = 0, \ldots, N-1$ describing the continuous modes that the robot will be in at various points along the horizon. These are defined *a priori* by estimating the time to impact of the robot. We also construct a sequence of $e_k = v_k \to v_{k+1} \in E \cup \{0\}$, where $e_k = 0$ if no discrete transition is expected. Consider a discrete (manifold valued) state trajectory $\bar{\mathbf{x}}_k$ and input trajectory $\bar{\mathbf{u}}_k$. We introduce the variables:

$$\bar{\mathbf{z}}_k = (\bar{\mathbf{p}}_k, \bar{\boldsymbol{\xi}}_k, \bar{\boldsymbol{\theta}}_k, \bar{\ell}_k, \bar{\mathbf{v}}_k) \in \mathbb{R}^{20},$$

with $\bar{\boldsymbol{\xi}}_k$ defined as in (5.17), and whereby $\mathbf{z}_k$ will represent our decision variables in the MPC program. At each index k, compute the linearizations of the dynamics in the vertex v_k:

$$\begin{aligned} \dot{\mathbf{z}}_k = \mathbf{A}_{v_k}\mathbf{z}_k + \mathbf{B}_{v_k}\mathbf{u}_k \\ + \underbrace{\mathbf{f}_{v_k}(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k) - \mathbf{A}_{v_k}\bar{\mathbf{z}}_k - \mathbf{B}_{v_k}\bar{\mathbf{u}}_k}_{\triangleq \mathbf{C}_{v_k}}, \end{aligned} \tag{5.23}$$

where the q_k and $\boldsymbol{\omega}_k$ elements are linearized as in (5.20), the Euclidean elements are linearized in the standard way. From (5.23), we can produce a discrete-time linear

system over a time interval $h \in \mathbb{R}_{>0}$ by taking an Euler step (in Euclidean space), or by using the matrix exponential in $\mathbb{R}^{20\times 20}$ to produce the discrete time dynamics:

$$\mathbf{z}_{k+1} = \mathbf{A}^d_{v_k}\mathbf{z}_k + \mathbf{B}^d_{v_k}\mathbf{u}_k + \mathbf{C}^d_{v_k}, \tag{5.24}$$

$$\mathbf{z}^+_k = \mathbf{D}_{e_k}\mathbf{z}^-_k + \underbrace{\boldsymbol{\Delta}_{e_k}(\bar{\mathbf{x}}^-_k) - \mathbf{D}_{e_k}\bar{\mathbf{z}}_k}_{\triangleq \mathbf{E}_{e_k}}. \tag{5.25}$$

We now introduce the finite-time optimal control problem (FTOCP), i.e., the *geometric model predictive controller*:

$$\begin{aligned}
\min_{\mathbf{u}_k,\mathbf{z}_k} \quad & \sum_{k=0}^{N-1} (\mathbf{z}_k - \mathbf{z}_{\text{ref}})^\top \mathbf{Q}(\mathbf{z}_k - \mathbf{z}_{\text{ref}}) + \mathbf{u}_k^\top \mathbf{R}\mathbf{u}_k + \mathbf{z}_N^\top \mathbf{V}\mathbf{z}_N \\
\text{s.t.} \quad & \mathbf{z}_{k+1} = \mathbf{A}^d_{v_k}\mathbf{z}_k + \mathbf{B}^d_{v_k}\mathbf{u}_k + \mathbf{C}^d_{v_k}, \quad \text{if } e_k = 0 && \text{(5.26a)} \\
& \mathbf{z}_{k+1} = \mathbf{D}_{e_k}\mathbf{z}_k + \mathbf{E}_{e_k}, \quad \text{if } e_k \neq 0 && \text{(5.26b)} \\
& \mathbf{z}_0 = \mathbf{z}(t), \;\; \boldsymbol{\xi}_0 = \mathbf{0} && \text{(5.26c)} \\
& \mathbf{u}_k \in \mathcal{U} && \text{(5.26d)}
\end{aligned}$$

where $\mathbf{Q} \in \mathbb{S}^{2n}_{>0}$ and $\mathbf{R} \in \mathbb{S}^{2n}_{>0}$ are symmetric, positive definite state and input gain matrices, respectively, $\mathbf{V} \in \mathbb{S}^{2n}_{>0}$ is a quadratic approximation of the cost-to-go, $\mathcal{U}$ is an input constraint set, and where the initial condition $\boldsymbol{\xi}_0 = \mathbf{0}$ is enforced. The above optimal control problem is solved in an SQP fashion, where the solution from the previous iteration is used to produce the linearizations for the next. Specifically, we can take $(\bar{\mathbf{z}}_k, \bar{\mathbf{u}}_k) = (\mathbf{z}^*_k, \mathbf{u}^*_k)$ where the asterisk indicates the optimal solution, and $\bar{\mathbf{x}}_k$ can be produced from $\bar{\mathbf{z}}_k$ via inverting (5.17).

Quaternionic Feedback

Once MPC produces a solution, a desired trajectory and feedforward input can be produced as:

$$q_d(\tau) = \bar{q}_0 \exp(\boldsymbol{\xi}^*_1), \qquad \boldsymbol{\omega}_d(\tau) = \boldsymbol{\omega}^*_1, \qquad \mathbf{u}_{\text{ff}}(\tau) = \mathbf{u}^*_0,$$

for $\tau \in [0, dt) \subset \mathbb{R}$. An interesting area of future work is using the MPC signals to produce dynamically admissible trajectories in the inter-MPC times, but this was not explored due to WiFi communication bandwidth limitations. The FTOCP is implemented in a receding horizon fashion, where the low layer controller only ever receives the first control input and desired trajectory.

Given the measured quaternion q_a, measured angular rate $\boldsymbol{\omega}_a$, desired quaternion q_d, and desired angular rate $\boldsymbol{\omega}_d$ of the robot, we can construct our actuation as:

$$\mathbf{u}(\mathbf{x}, t) = -\mathbf{K}_p \mathbb{I}\text{m}(q_d(t)^{-1} q_a) - \mathbf{K}_d(\boldsymbol{\omega}_a - \boldsymbol{\omega}_d(t)) + \mathbf{u}_{\text{ff}(t)},$$

where $\mathbf{K}_p, \mathbf{K}_d \in \mathbb{S}^3_{>0}$ are positive definite gain matrices. The product $q_d(t)^{-1}q_a$ represents a "difference" between elements of S^3; if q_a is in a small neighborhood of q_d, then the product is in a small neighborhood of the identity element $q_\mathcal{E}$. The map $\mathbb{I}\text{m} : S^3 \to \mathfrak{s}^3$ takes the purely imaginary component of the error signal, and can be viewed as the Euclidean projection of the Lie group onto the Lie algebra, allowing us to base the control input over a vector valued error. Alternatively, the $\log$ operation could be used instead of $\mathbb{I}\text{m}$, but the $\mathbb{I}\text{m}$ operator was empirically found to work more reliably.

Implementation and Results

Hardware

The MPC program runs at 100 Hz on an Ubuntu 20.04 machine with AMD Ryzen 5950x @ 3.4 GHz and 64 Gb RAM and communicates over WiFi at 100 Hz via an ESP module to the onboard Teensy microcontrollers running the low level feedback control. The Pinocchio C++ library [157] is used, specifically the pinocchio3 preview branch, to produce fast evaluation of the system dynamics (constrained, unconstrained, continuous, and discrete), as well as their associated Jacobians, and the manif C++ library [169] is used to handle all Lie group operations (such as $\log$ and $\exp$). As seen in Figure 5.11 as well as the supplemental video [170], the robot was successfully able to hop stably in place, demonstrating the first instance of 3D hopping using online motion planning.

Simulation

In order to thoroughly test the method, the torque limits of the robot were increased in a Mujoco [171] simulation environment from 1.5 Nm to 15 Nm. First, the tracking of various global reference signals, including a square and a Lissajous trajectory, were evaluated as seen in Figure 5.12. Note the constant global velocity in the flight phase due to the lack of control inputs when the robot is in the air. As such, the robot must carefully plan its interactions with the ground in order to track the desired reference signals. Specifically, it is interesting to see how MPC is implicitly able to control the actuated coordinates of the robot in order to stabilize the underactuated ones. Also note the discontinuities in the MPC planned trajectories around the impact events due to the hybrid nature of the system dynamics. Next, disturbance rejection and more dynamic maneuvers like flipping were tested on the system, as seen in the accompanying video [170] and Figure 5.13. Due to the geometrically consistent structure of the planning algorithm, the robot is able to explore a variety of states on

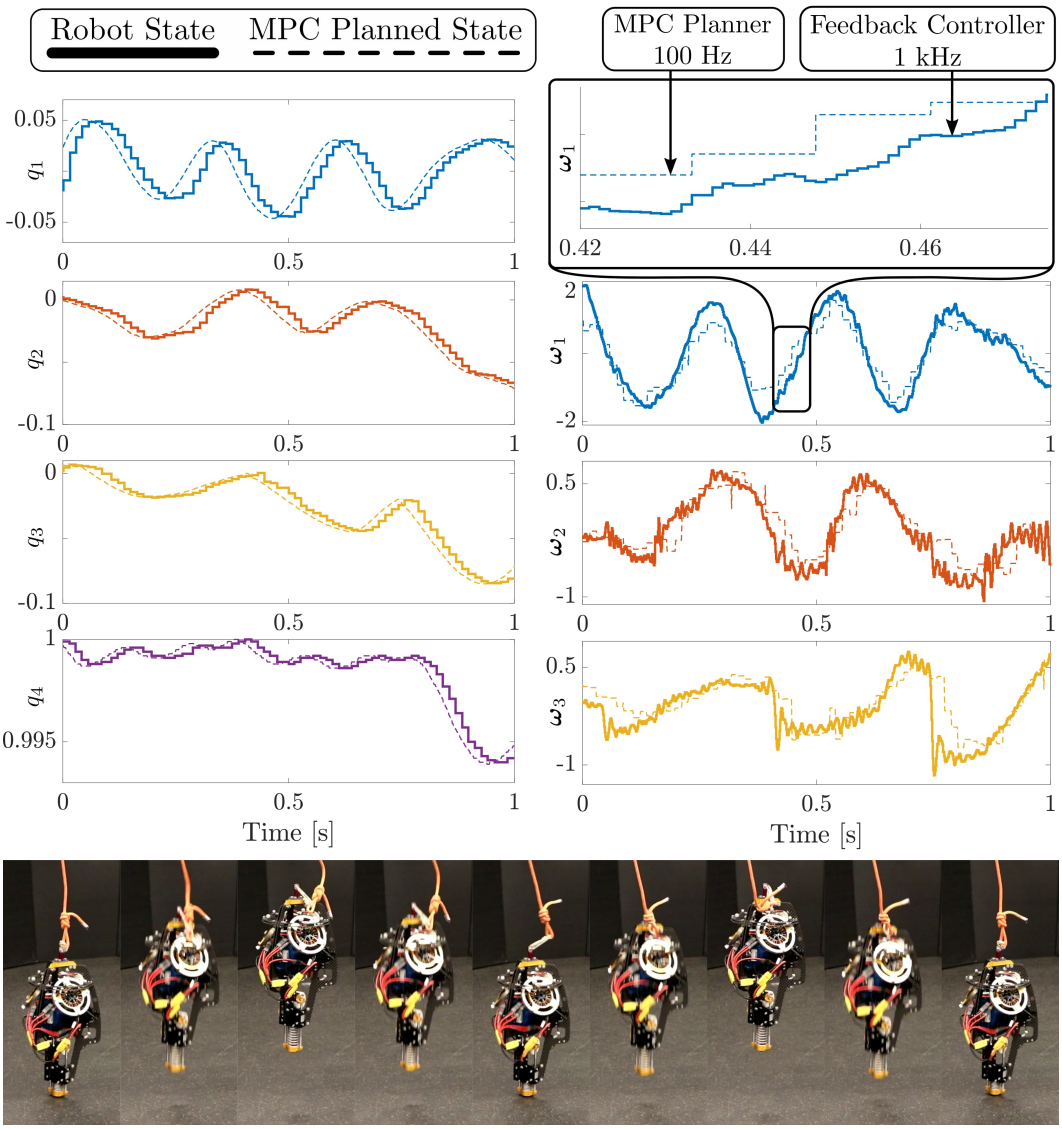


Figure 5.11: The planned elements $q \in S^3$ and $\boldsymbol{\omega} \in \mathfrak{s}^3$, as well as the low level feedback controller. The multi-rate nature of the methodology can be seen in the difference of time scales between when MPC produces trajectories and when the low level controller updates.

its orientation manifold, and exhibits exceptional robustness to disturbances. Note that the torque limitations, but not limitations of the methodology, prohibit such demonstrations on the hardware platform.

Implementation Details

The complete list of parameters used in the MPC program are detailed in Table 5.3. There is an inherent trade-off between tracking global position and maintaining a

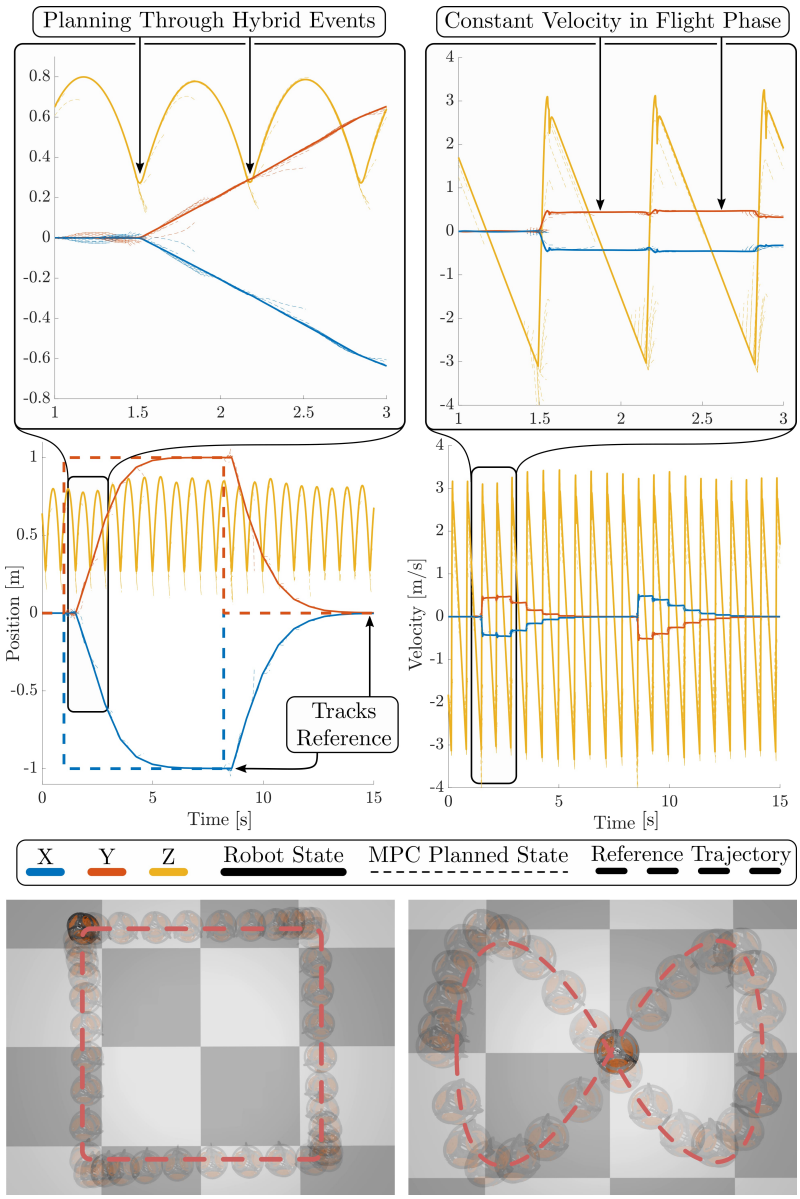


Figure 5.12: Trajectory tracking on ARCHER. **(Above)** Positions and velocities of the robot tracking a global reference setpoint in simulation using geometric MPC. Note the planning through hybrid events and constant velocity in flight phase due to underactuation. **(Below)** Two reference trajectories.

vertical orientation — as such the associated gains need to be appropriately tuned. To avoid adding a nonlinear and mixed integer constraint in to the optimization program, the impact time was calculated as though the hopper was exactly vertical via solving for the ballistic trajectory in the z-direction. As the spring dynamics add significant stiffness to the optimization problem, the foot torque was set to zero in the optimizer, and instead the MPC program plans as though it is a passive degree of

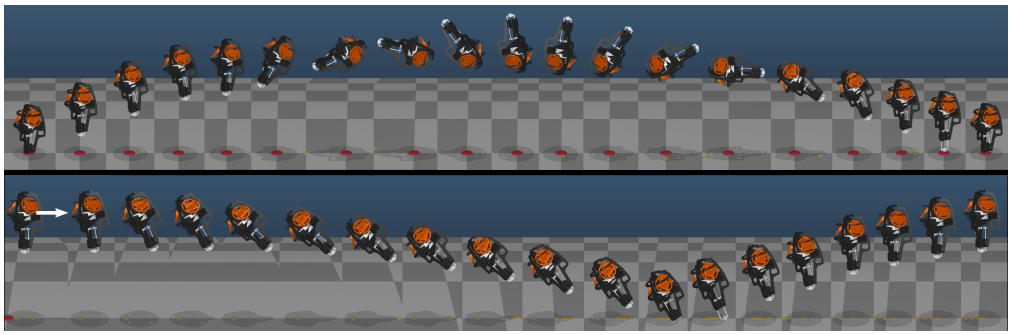


Figure 5.13: Dynamic motions explored in simulation using geometric MPC, including flipping (above) and disturbance rejection (below).

freedom. Instead, the low level controller runs its own feedback controller to regulate foot compression between impact events. As the dynamics of the ground phase are more challenging than the flight phase, the system was discretized more finely in that domain. This means that the look-ahead time shrank whenever impact came into view in the horizon, as the horizon length was kept constant. We found that using the matrix exponential instead of the Euclidean Euler step aided in performance, likely due to the reduced one step prediction error. Finally, the quadratic approximation of the cost to go was simply taken to be $\mathbf{Q}$. The complete code can be found at [172].

5.4 Learning Walking Behaviors by Enforcing Set Invariance

In the previous chapter, we enforced set invariance for staying on a stepping stone while tracking a predefined walking gait, which we have discussed how to generate in this chapter. In this section, we drop the notion of a predefined gait and ask the question:

> **Question:** *Can walking behaviors be generated simply by enforcing set invariance?*

This perspective is termed Neural Gaits, and is depicted in Figure 5.14. To answer

Table 5.3: MPC parameters

Horizon Length	20	SQP Iterations	2
$\mathbf{p}$ Weight	10	$\mathbf{v}$ Weight	1
q Weight	10	$\boldsymbol{\omega}$ Weight	0.01
$\mathbf{u}$ Weight	0.001	$\mathbf{u}_{\text{max}}$	1.5 Nm
K_p Roll/Pitch Gain	120	K_d Roll/Pitch Gain	4
K_p Yaw Gain	15	K_d Yaw Gain	1
dt_{flight}	0.01	dt_{ground}	0.001

this question, denote a parameter in a parameter space $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, whereby we can define a collection of k outputs $\mathbf{y} : \mathcal{X} \times \boldsymbol{\Theta} \to \mathbb{R}^k$ parameterized by $\boldsymbol{\theta}$ that we would like to converge to zero as:

$$\mathbf{y}(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{y_a}(\mathbf{x}) - \mathbf{y_d}(\mathbf{x}; \boldsymbol{\theta}), \tag{5.27}$$

where $\mathbf{y_a} : \mathcal{X} \to \mathbb{R}^k$ are the measured outputs, and $\mathbf{y_d} : \mathcal{X} \times \boldsymbol{\Theta} \to \mathbb{R}^k$ are the desired outputs. We take the outputs to be joint angles, and $\boldsymbol{\theta}$ to be the parameters of a neural network. Next, fix a controller structure $\mathbf{u}(\mathbf{x}; \boldsymbol{\theta})$, which is parameterized by $\boldsymbol{\theta}$ through the definition of $\mathbf{y}(\mathbf{x}; \boldsymbol{\theta})$. The neural gaits method relies on the assumption that good walking can be characterized as a forward invariant set. Thus, the first step of the method requires us to define a set of barrier functions that imply good walking. In the following discussion, we will only consider barrier functions defined on the zero dynamics surface, i.e., $h : \mathcal{Z} \to \mathbb{R}$ as defined when the error coordinates are zero ($\boldsymbol{\eta} = \mathbf{0}$). Importantly, the guarantees made on $\mathcal{Z} \subset \mathcal{X}$ have relevance to the full state space, as is made precise in Section 5.4.

After constructing a collection of barrier functions, we train a policy $\mathbf{y_d}$ that ensures the system stays safe by minimizing the violation of the barrier function conditions over regions of the state space. The resulting policy renders the intersection of the safe set for all barriers forward invariant. Finally, to mitigate model mismatch, we train a residual term $\varepsilon(\mathbf{z})$ on the zero dynamics. These corrected zero dynamics are then used to refine the existing policy episodically until the desired walking performance is achieved.

Learning the Policy $\mathbf{y_d}$

This learning approach builds upon and unifies two lines of work. The first studies how to characterize good walking behavior as set invariance via a collection of barrier function candidates [173]. The second studies how to train neural ODEs to satisfy control-theoretic properties such as Lyapunov stability [174], which we extend to the barrier setting.

Recall from Section 2.4 that the error and zero dynamics coordinates are computed from the states using the diffeomorphism $\boldsymbol{\Phi}$, which only depends on the policy through $\boldsymbol{\Phi}_\eta$. We thus parameterize the policy as a function of the projection of the state onto the zero dynamics manifold and parameters $\boldsymbol{\theta}$, i.e., $\mathbf{y_d}(\mathbf{x}) = \mathbf{y_d}(\boldsymbol{\Phi}_\mathbf{z}(\mathbf{x}); \boldsymbol{\theta})$. In other words, $\mathbf{y}_d$ only depends on the unactuated degrees of freedom of the system rather than the full state. Therefore, when there is no error (i.e., $\boldsymbol{\eta} = 0$) we have that $\mathbf{z} \in \mathcal{Z}$ with dynamics $\dot{\mathbf{z}} = \boldsymbol{\omega}(\mathbf{0}, \mathbf{z}; \boldsymbol{\theta})$. Note, importantly, that even when the error

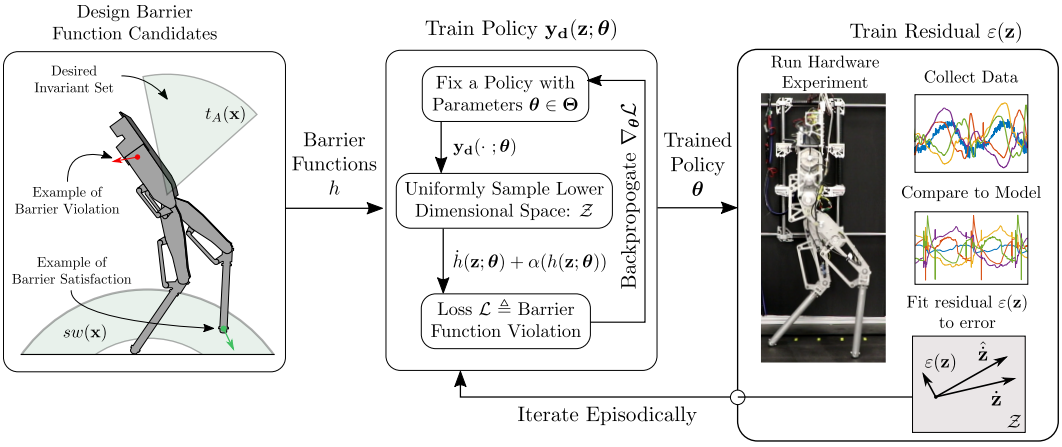


Figure 5.14: A depiction of the Neural Gaits framework. **(Left)** Designing barrier function candidates that we use to formally describe walking. **(Middle)** Training a policy capable of satisfying all the barrier condition in the zero dynamics state space where the constraint is enforced. **(Right)** Collecting hardware data to train a residual zero dynamics model. We then refine the policy episodically using the augmented model.

coordinates are zero, the zero dynamics are still a function of $\mathbf{y_d}$ and therefore $\boldsymbol{\theta}$. This implies that the zero dynamics are influenced by the parameters of the policy even though the control inputs are not present in $\boldsymbol{\omega}$.

Learning to Satisfy Barrier Conditions. Taking inspiration from [175] and [173], we assume that walking can be characterized as set invariance via a collection of barrier function candidates $\mathcal{H} = \{h_i\}_{i=1}^N$ (see Table 5.4 discussed later in Section 5.4). To each h_i, we associate a *region at risk* $\overline{\mathcal{S}}_i \subseteq \mathcal{Z}$ where the barrier function is enforced. We define a set of neural network parameters that render the region at risk safe under the barrier definition:

$$\boldsymbol{\Theta}_i = \{\boldsymbol{\theta} \in \boldsymbol{\Theta} : \forall_{\mathbf{z}\in\overline{\mathcal{S}}_i} \quad \dot{h}_i(\mathbf{z};\boldsymbol{\theta}) \geq -\alpha(h_i(\mathbf{z};\boldsymbol{\theta}))\}. \tag{5.28}$$

In other words, each $\boldsymbol{\Theta}_i$ corresponds to the set of policy parameters that render the set $\overline{\mathcal{S}}_i$ safe.

Thus, our learning problem is equivalent to finding a set of parameters $\boldsymbol{\theta} \in \bigcap_{i=1}^N \boldsymbol{\Theta}_i$ that render the system safe in all regions at risk. Similar to the Lyapunov Loss studied in [174], we introduce the concept of *Barrier Loss* as a learning signal for training.

Definition 5.1 (Barrier Loss)**.** For a set of barrier function candidates $\mathcal{H} = \{h_i\}_{i=1}^{N}$ and corresponding regions at risk $\overline{\mathcal{S}}_i \subset \mathcal{Z}$ on the zero dynamics, a Barrier Loss, $\mathcal{L} : \boldsymbol{\Theta} \to \mathbb{R}_{\geq 0}$, is defined as:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \int_{\overline{\mathcal{S}}_i} \max\{0, -\dot{h}_i(\mathbf{z}; \boldsymbol{\theta}) - \alpha(h_i(\mathbf{z}; \boldsymbol{\theta}))\} d\mathbf{z}. \tag{5.29}$$

When a choice of parameters $\boldsymbol{\theta}$ achieves zero Barrier Loss, then the safety of the zero dynamics is guaranteed by satisfying the forward invariance condition of the barrier functions:

Theorem 5.2 (Zero Barrier Loss Implies Safety of Zero Dynamics)**.** *The zero dynamics is guaranteed to be safe in all its regions at risk if and only if we find a* $\boldsymbol{\theta}^*$ *that attains* $\mathcal{L}(\boldsymbol{\theta}^*) = 0$*.*

Proof. Notice that for all $i \in \{1 \dots N\}$ both $\dot{h}_i$ and $\alpha \circ h_i$ are continuous functions. This implies that for all $\mathbf{z} \in \mathcal{Z}$ and $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, $\max\{0, -\dot{h}_i(\mathbf{z}; \boldsymbol{\theta}) - \alpha(h_i(\mathbf{z}; \boldsymbol{\theta}))\}$ is a continuous non-negative real function. It is well known that a continuous non-negative real function will have zero integral if and only if it is the zero function. We specialize this statement for the terms in our loss as follows:

$$\begin{aligned} &\forall_{\mathbf{z} \in \overline{\mathcal{S}}_i} \max\{0, -\dot{h}_i(\mathbf{z}; \boldsymbol{\theta}) - \alpha(h_i(\mathbf{z}; \boldsymbol{\theta}))\} = 0 \\ &\qquad \Leftrightarrow \int_{\overline{\mathcal{S}}_i} \max\{0, -\dot{h}_i(\mathbf{z}; \boldsymbol{\theta}) - \alpha(h_i(\mathbf{z}; \boldsymbol{\theta}))\} d\mathbf{z} = 0. \end{aligned} \tag{5.30}$$

It is clear that the sum in $\mathcal{L}(\boldsymbol{\theta})$ will be zero if an only if each integral term is zero since each integral is a non-negative function. Thus we can conclude that $\mathcal{L}(\boldsymbol{\theta}^*) = 0$ if and only if

$$\forall_{i \in \{1 \dots N\}, \mathbf{z} \in \overline{\mathcal{S}}_i} \max\{0, -\dot{h}_i(\mathbf{z}; \boldsymbol{\theta}^*) - \alpha(h_i(\mathbf{z}; \boldsymbol{\theta}^*))\} = 0.$$

For any barrier h_i and $\mathbf{z} \in \mathcal{Z}$ you can see that $\max\{0, -\dot{h}_i(\mathbf{z}; \boldsymbol{\theta}^*) - \alpha(h_i(\mathbf{z}; \boldsymbol{\theta}^*))\} = 0$ implies that:

$$-\dot{h}_i(\mathbf{z}; \boldsymbol{\theta}^*) - \alpha(h_i(\mathbf{z}; \boldsymbol{\theta}^*)) \leq 0 \implies \dot{h}_i(\mathbf{z}; \boldsymbol{\theta}^*) \geq -\alpha(h_i(\mathbf{z}; \boldsymbol{\theta}^*)),$$

i.e., the safety condition for the barrier is satisfied. □

Instantiation for Bipedal Walking

Table 5.4 describes the barrier functions used in our experiments, which take inspiration from [173]. We depict some of these conditions on the robot in Figure 5.15a. As all barrier functions $h_i : \mathcal{X} \to \mathbb{R}$ are enforced on the zero dynamics surface, we will write them implicitly as $h_i \circ \mathbf{\Phi}^{-1}(\mathbf{0}, \cdot\,; \boldsymbol{\theta}) : \mathcal{Z} \to \mathbb{R}$ for $i \in \{1 \dots N\}$ with $N = 5$ in this instantiation. In Table 5.4, t_A represents the torso angle, and p_x and p_z represent the x and z position of the swing foot, respectively. In addition to continuous time conditions, various conditions needed to be enforced on the guard, namely enforcing the location of the guard, symmetry of the model before and after impact, and a guard mapping condition. Interestingly, although these barriers would be relative degree two on the full state dynamics, they are directly enforceable as relative degree one barriers on the zero dynamics. This can be seen by treating $\mathbf{y_d}$ as the input to the zero dynamics, and observing that the zero dynamics themselves are functions of $\mathbf{y_d}$.

Note that these barrier functions h are defined over the space $\mathcal{Z}$, as, given a policy $\mathbf{y_d}(\,\cdot\,; \boldsymbol{\theta}) : \mathcal{Z} \to \mathbb{R}^4$, the mapping $\mathbf{\Phi}^{-1} : \mathcal{N} \times \mathcal{Z} \to \mathcal{X}$ is uniquely defined. We take inspiration from reduced order models, and specifically the notion of orbital energy [176] to define a set $\mathcal{Z}_\mathcal{O} \subset \mathcal{Z}$ with reasonably bounded orbital energies as our first region at risk. We also define the set $\mathcal{S}_\epsilon \subset \mathcal{Z}_\mathcal{O}$ which contains the part of the guard in $\mathcal{Z}_\mathcal{O}$ as well as a small region around it where discrete-time guard conditions are enforced. We learn policies that satisfy the barrier conditions on these regions of the zero dynamics by penalizing the violation of the constraints shown in Table 5.4. Notice that penalizing guard constraints over a region results in policies that are robust to impact modeling error since the policy must be prepared to change stance foot at any point in $\mathcal{S}_\epsilon$ rather than just the guard $\mathcal{S}$.

Torso Angle	$\{\mathbf{z} \in \mathcal{Z}_\mathcal{O}\}$	$-\frac{\pi}{10} \leq \theta_t(\mathbf{z}) \leq 0.05$
Swing Foot Clearance	$\{\mathbf{z} \in \mathcal{Z}_\mathcal{O}\}$	$0 \leq (p_x(\mathbf{z}) - c_x)^2 + (p_z(\mathbf{z}) - c_z)^2 - r^2 \leq 0.3$
Impact Mapping	$\{\mathbf{z} \in \mathcal{S}_\epsilon\}$	$-0.15 \leq \Delta(\mathbf{z}) + \mathbf{z} \leq 0.15$
Symmetry	$\{\mathbf{z} \in \mathcal{S}_\epsilon\}$	$\mathbf{y}(\mathbf{z}) = \mathbf{y}(\Delta(\mathbf{z}))$
Foot on Guard	$\{\mathbf{z} \in \mathcal{S}_\epsilon\}$	$p_z(\mathbf{z}) = 0$

Table 5.4: Barrier functions used to characterize bipedal walking, and the associated regions at risk in which they are enforced. The first two are enforced over the continuous dynamics, and the bottom three in a buffered region of the guard. The strict equality on symmetry and the foot on guard conditions were also enforced as a training loss.

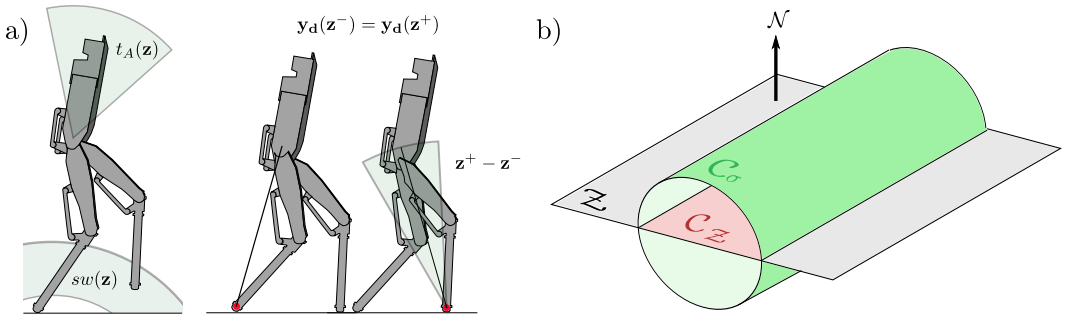


Figure 5.15: Barrier functions and output stability. **a)** A depiction of the barrier functions used to enforce walking as set invariance. On the left are the two continuous time barrier conditions, and on the right the three barrier conditions enforced at the guard. The red dot on the foot indicates the stance foot, and **b)** The safe set on the zero dynamics $\mathcal{C}_\mathcal{Z}$, as certified by the proposed learning method, and the combined safe set $\mathcal{C}_\sigma$, and described in Theorem 5.

Learning Optimization Details

Evaluating the Barrier Loss in Equation (5.29) requires solving an integral that is in general intractable. Instead, we use Monte Carlo sampling to approximate the integral. Since our approach follows Algorithm 1 of [174] we refer to it for more details while noting that we optimize for the Barrier Loss rather than the Lyapunov Loss. A key ingredient in the Monte Carlo sampling approach in [174] is defining a compact support set to sample from (i.e., where the barrier condition should be satisfied). In our work this compact support set directly corresponds to the region at risk for each barrier condition.

Learning the Residual Zero Dynamics $\varepsilon(\mathbf{z})$

As outlined in Figure 5.14, we can improve upon the nominal zero dynamics model by collecting trajectories of the robot executing the resulting policy in hardware. We can then use those trajectories to learn a residual error term on the zero dynamics $\hat{\dot{\mathbf{z}}} = \omega(0, \mathbf{z}; \boldsymbol{\theta}) + \varepsilon(\mathbf{z})$ where ε is the learned residual term. We model this residual term using Neural ODEs [177], which are naturally compatible with our policy learning approach. We can iterate this process multiple times, alternating between learning $\boldsymbol{\theta}$ and ε until the resulting policy achieves the desired behavior.

Providing Guarantees in the Full State Space

Assuming a controller which exponentially converges the outputs $\mathbf{y}(\mathbf{x})$, for example a feedback linearizing or control Lyapunov function based controller, the converse Lyapunov theorem allows us to construct a Lyapunov function $V_\eta : \mathcal{N} \to \mathbb{R}$ verifying the exponential convergence of the outputs. Along with a certificate of safety

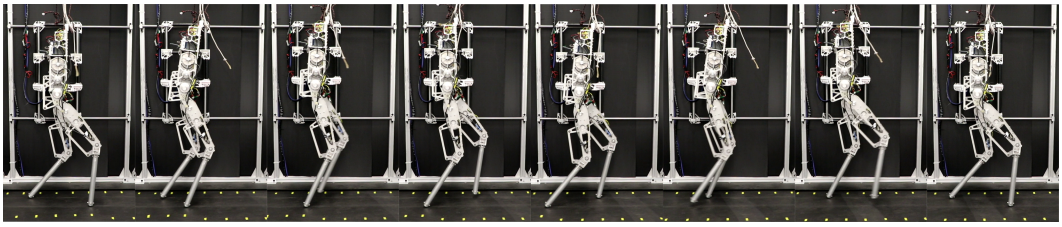


Figure 5.16: Gait tiles of the neural network encoding the final trained policy running in real time on the AMBER-3M robot. For a video discussing the methodology and summarizing the hardware results, please refer to [178].

$h_Z : \mathcal{Z} \to \mathbb{R}$ on the zero dynamics space, we can construct a set in the combined space $\mathcal{N} \times \mathcal{Z}$ which is safe, and has a barrier function certificate. This is described in the following theorem.

Theorem 5.3. *Let $V_{\boldsymbol{\eta}} = \boldsymbol{\eta}^\top \mathbf{P} \boldsymbol{\eta} : \mathcal{N} \to \mathbb{R}$ be an exponential control Lyapunov function for the output dynamics with $\dot{V}_{\boldsymbol{\eta}} \leq -\gamma V_{\boldsymbol{\eta}}$ and $h_Z : \mathcal{Z} \to \mathbb{R}$ be a barrier function on the zero dynamics with safe set $\mathcal{C}_Z$. Then, there exists a constant $\sigma \geq 0$ and $c \geq 0$ such that if $\dot{h}_Z(z) \geq -\alpha h_Z(z) + c$ with $\alpha \leq \frac{\gamma}{2}$, the barrier function $h(\boldsymbol{\eta}, \mathbf{z}) = h_Z(\mathbf{z}) - \sigma V_{\boldsymbol{\eta}}(\boldsymbol{\eta})$ is safe with set $\mathcal{C}_\sigma$.*

Proof. First note that the derivative of the function is given by:

$$\begin{aligned}
\dot{h} &= \frac{\partial h_Z}{\partial \mathbf{z}}(\mathbf{z}) \mathbf{w}(\boldsymbol{\eta}, \mathbf{z}) - \sigma \dot{V}_{\boldsymbol{\eta}}(\boldsymbol{\eta}) \\
&\geq -\alpha h_Z(\mathbf{z}) + c - \left| \frac{\partial h_Z}{\partial \mathbf{z}}(\mathbf{z}) \left(\mathbf{w}(\boldsymbol{\eta}, \mathbf{z}) - \mathbf{w}(0, \mathbf{z}) \right) \right| + \sigma \gamma V_{\boldsymbol{\eta}}(\boldsymbol{\eta}) \\
&\geq -\alpha h(\boldsymbol{\eta}, \mathbf{z}) + c - L_{h_Z} L_{\boldsymbol{\omega}_{\boldsymbol{\eta}}} \|\boldsymbol{\eta}\|_2 + \frac{\sigma \gamma}{2} \lambda_{min}(\mathbf{P}) \|\boldsymbol{\eta}\|_2^2,
\end{aligned} \tag{5.31}$$

where the third line follows from Cauchy Schwartz, the fact that h_Z and $\boldsymbol{\omega}(\boldsymbol{\eta}, \mathbf{z})$ are locally Lipschitz with Lipschitz constants L_{h_Z} and $L_{\boldsymbol{\omega}_{\boldsymbol{\eta}}}$, respectively, converse Lyapunov, and the assumption that $\alpha \leq \frac{\gamma}{2}$. Taking $\beta_1 = L_{h_Z} L_{\boldsymbol{\omega}_{\boldsymbol{\eta}}}$, and $\beta_2 = \frac{\gamma}{2} \lambda_{min}(\mathbf{P})$, we observe that $-\beta_1 \|\boldsymbol{\eta}\|_2 + \sigma \beta_2 \|\boldsymbol{\eta}\|_2^2 \geq -\frac{\beta_2^2}{4 \sigma \beta_3} \triangleq c$. By taking c defined as such, we achieve the desired result. □

The above theorem motivates the perspective of this work: satisfying barrier function certificates in the zero dynamics enables reasoning about safe sets in the complete state space. Note that the hybrid case is not addressed here, and is an interesting direction for future theoretical work.

Simulation and Experimental Results

The hardware platform used in this work was the planar underactuated biped AMBER-3M, which has actuators on the hips and knees, and point contact feet. Both in simulation, where the RaiSim [179] environment was used, and on hardware, the pipeline went as follows: the zero dynamics coordinate $\mathbf{z}$ was estimated, the Neural Network policy $\mathbf{y}_d(\mathbf{z};\boldsymbol{\theta})$ was evaluated, and the desired output values were passed to a PD controller running at 1kHz. The policy $\mathbf{y}_d(\mathbf{z};\boldsymbol{\theta})$ was randomly initialized and was trained for 1000 epochs. The AdamW optimizer was used in PyTorch [180] with an initial learning rate of 10^{-2}, weight decay of 10^{-4}, with a learning rate decay schedule at epochs 100, 400, and 800. Initially, the "gait" had the robots leg flailing randomly in the air, and when integrated resulted in the robot falling over. Once the loss converged, the policy had a loss in the order of 5×10^{-3}, and was able to walk stably in the simulation. The neural network ran in closed loop on the hardware platform and was called at approximately 500 Hz to produce desired outputs for the system to track. Unlike simulation, once tested on hardware, the policy resulted in the robot stumbling forward, unable to walk without falling. Data was collected over various trials, after which the methodology proposed in Section 5.4 was used to learn the residual of the model uncertainty, as projected to the zero dynamics space. During this process, Adam and other SGD methods were numerically unstable even with gradient clipping, so Nero [181] was used instead.

Once a residual term was learned, a new policy $\mathbf{y}_d(\mathbf{z},\boldsymbol{\theta})$ was trained with the updated dynamics (warm started with the policy from the previous iteration). After convergence, the gait was again tried on hardware. The gait was significantly more stable, and able to walk without assistance; however, the gait was not robust to walking speeds. Therefore, the process was repeated, and again a new policy was learned. When testing that policy, the robot was able to walk on its own, and was robust to different walking speeds. A sample gait is shown on Figure 5.16. The complete code can be found here [182].

5.5 Zero Dynamics Policies

In the last two sections, we have developed a powerful and constructive method for stabilizing underactuated systems. What we have not yet used, however, is the beautiful mathematical construction of zero dynamics that we developed in Section 2.4. Therefore, the focus of this section will be to address the following question:

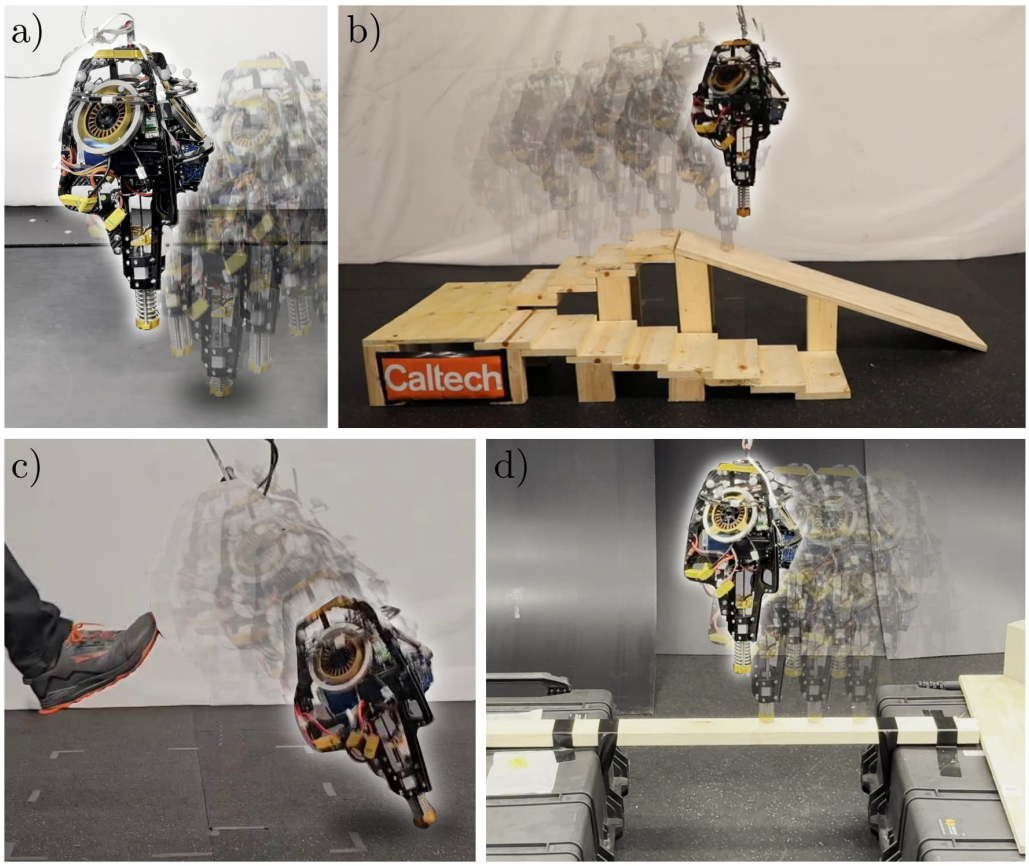


Figure 5.17: Experiments run with Zero Dynamics Policies: **a)** treadmill hopping with disturbances up to 1 mile per hour, **b)** 1.5" stair climbing and 20° ramp descending, **c)** disturbance rejection, and **d)** hopping across a 2x4.

> **Question:** *Can we leverage the zero dynamics decomposition in the paradigm of optimal control, and if so, wat are the benefits?*

In this section, we attempt to answer this question, and in doing so provide a means of imitation learning optimal control on the zero dynamics manifold, a method we term Zero Dynamics Policies (ZDP). For the diligent and thorough reader, this section is the best section of the entire thesis, and the result I am proudest of.

From Hybrid Dynamics to Discrete-Time Dynamics

The starting point for our discussion is the hybrid dynamical section from Section 2.3:

$$\mathscr{H} = \begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u} & \mathbf{x} \notin \mathcal{S} \\ \mathbf{x}^+ = \boldsymbol{\Delta}(\mathbf{x}^-) & \mathbf{x}^- \in \mathcal{S} \end{cases} \tag{5.32}$$

where $\mathcal{S} \subset \mathcal{X}$ is a guard that defines when the robot impacts the ground. We will be interested in modeling the hybrid system $\mathscr{H}$ as a discrete-time dynamical system via its impact-to-impact dynamics. To this end, let $\mathbf{x}_k \in \mathcal{X}$ denote the robot state just before impact, P denote an admissible parameter set for $\mathbf{v}_k \in P$, a discrete parameterization of the control input over a single continuous phase, and $t_k \in \mathbb{R}_{\geq 0}$ be the duration of the continuous phase. We reformulate our hybrid control system into discrete dynamics via:

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k; \mathbf{v}_k) \triangleq \boldsymbol{\varphi}(t_k, \boldsymbol{\Delta}(\mathbf{x}_k); \mathbf{v}_k). \tag{5.33}$$

where $\mathbf{F} : \mathcal{X} \times P \to \mathcal{X}$ composes the impact map with the flow of the continuous system under a parameterized feedback controller $\mathbf{u} = \mathbf{k}(\mathbf{x}(t), \mathbf{v}_k) \in \mathcal{K}$. In the context of hopping, we take $\mathbf{v}_k$ to be the desired impact angle. This parameterization of control input allows us to reason about the effect of impact conditions on the resulting system dynamics, which are the primary means of stabilizing legged systems. We assume that $\mathbf{F}$ is well defined (which implicitly requires a lower bound on the time between impacts) and that the origin is an unactuated equilibrium position, i.e., that $\mathbf{F}(\mathbf{0}; \mathbf{0}) = \mathbf{0}$. For a complete discussion of how to achieve this representation from the underlying hybrid dynamics, see [183].

Outputs and Zero Dynamics

Understanding the structure of underactuation provides key insight into constructing stabilizing controllers for these systems. To analyze the states that actuation directly impacts, consider the following coordinate change from Section 2.4:

$$\boldsymbol{\eta} = \boldsymbol{\Phi}_{\boldsymbol{\eta}}(\mathbf{x}) \triangleq \begin{bmatrix} \mathbf{B}^\top \mathbf{q} \\ \mathbf{B}^\top \dot{\mathbf{q}} \end{bmatrix}, \quad \mathbf{z} = \boldsymbol{\Phi}_{\mathbf{z}}(\mathbf{x}) \triangleq \begin{bmatrix} \mathbf{N}\mathbf{q} \\ \mathbf{N}\mathbf{D}(\mathbf{q})\dot{\mathbf{q}} \end{bmatrix} \tag{5.34}$$

for $\boldsymbol{\eta} \in \mathcal{N} \subset \mathcal{X}$ and $\mathbf{z} \in \mathcal{Z} \subset \mathcal{X}$, where $\mathbf{N} \in \mathbb{R}^{(n-m)\times n}$ is chosen to be a basis for the left nullspace of $\mathbf{B}$. As seen in Section 2.4, we know $\boldsymbol{\Phi}(\mathbf{x}) \triangleq (\boldsymbol{\Phi}_{\boldsymbol{\eta}}(\mathbf{x}), \boldsymbol{\Phi}_{\mathbf{z}}(\mathbf{x}))$ is a diffeomorphism between $\mathcal{X}$ and $\mathcal{N} \times \mathcal{Z}$; therefore, $\boldsymbol{\Phi}^{-1}$ exists and any conclusions

of stability of $(\boldsymbol{\eta}, \mathbf{z})$ are directly transferable back to $\mathbf{x}$. In these coordinates, the hybrid dynamics are given by:

$$\mathscr{N} = \begin{cases} \dot{\boldsymbol{\eta}} = \hat{\mathbf{f}}(\boldsymbol{\eta}, \mathbf{z}) + \hat{\mathbf{g}}(\boldsymbol{\eta}, \mathbf{z})\mathbf{u} & \boldsymbol{\Phi}^{-1}(\boldsymbol{\eta}, \mathbf{z}) \notin \mathcal{S} \\ \dot{\mathbf{z}} = \boldsymbol{\omega}(\boldsymbol{\eta}, \mathbf{z}) & \\ \boldsymbol{\eta}^+ = \boldsymbol{\Delta}_{\boldsymbol{\eta}}(\boldsymbol{\eta}^-, \mathbf{z}^-) & \boldsymbol{\Phi}^{-1}(\boldsymbol{\eta}, \mathbf{z}) \in \mathcal{S} \\ \mathbf{z}^+ = \boldsymbol{\Delta}_{\mathbf{z}}(\boldsymbol{\eta}^-, \mathbf{z}^-) & \end{cases} \tag{5.35}$$

termed the *actuated* dynamics and the *unactuated* dynamics, respectively. Note that these coordinates were exactly chosen such that $\hat{\mathbf{g}}(\boldsymbol{\eta}, \mathbf{z})$ is full rank and $\frac{d\mathbf{z}}{d\mathbf{x}}\mathbf{g}(\mathbf{x}) \equiv \mathbf{0}$; as such, this mapping decomposes the state space into coordinates which can directly be controlled, and those which cannot.

Assuming the continuous time input does not effect the impact map or impact time[3], applying $\boldsymbol{\Phi}$ to the discrete dynamics (5.33) results in:

$$\boldsymbol{\eta}_{k+1} = \hat{\mathbf{F}}(\boldsymbol{\eta}_k, \mathbf{z}_k, \mathbf{v}_k), \quad \mathbf{z}_{k+1} = \boldsymbol{\Omega}(\boldsymbol{\eta}_k, \mathbf{z}_k). \tag{5.36}$$

Now, consider a mapping $\boldsymbol{\psi} : \mathcal{Z} \to \mathcal{N}$ and associated discrete-time error $\mathbf{e}_k = \boldsymbol{\eta}_k - \boldsymbol{\psi}(\mathbf{z}_k)$. The goal will be to design $\boldsymbol{\psi}$ such that driving $\mathbf{e}_k$ to zero results in stability of the overall system. This choice of error parameterization is inspired by other successful results in robotics; the Raibert Heuristic [36], reduced order models [184], and regulators for HZD gaits [185] all reason about where to place a robot's feet (the actuated state) as a function of their center of mass state (the underactuated state). We aim to generalize these methods and reason explicitly about constructive methods to generate provably stable behaviors. The construction of the mapping $\boldsymbol{\psi}$ induces an associated manifold $\mathcal{M}_{\boldsymbol{\psi}} \subset \mathcal{X}$ via:

$$\mathcal{M}_{\boldsymbol{\psi}} \triangleq \{(\boldsymbol{\eta}_k, \mathbf{z}_k) \mid \boldsymbol{\eta}_k = \boldsymbol{\psi}(\mathbf{z}_k)\}. \tag{5.37}$$

Paralleling the continuous time notion of controlled invariance in Section 2.1, we will be interested in enforcing conditions such that $\mathcal{M}_{\boldsymbol{\psi}}$ is discretely controlled invariant, defined as:

Definition 5.4. The manifold $\mathcal{M}_{\boldsymbol{\psi}}$ is *controlled invariant* if for all $(\boldsymbol{\eta}_k, \mathbf{z}_k) \in \mathcal{M}_{\boldsymbol{\psi}}$ there exists a $\mathbf{v}_k \in P$ such that the next state remains on the manifold, i.e.:

$$\Big(\mathbf{F}(\boldsymbol{\eta}_k, \mathbf{z}_k, \mathbf{v}_k),\ \boldsymbol{\Omega}(\boldsymbol{\eta}_k, \mathbf{z}_k)\Big) \in \mathcal{M}_{\boldsymbol{\psi}}.$$

[3]This assumption is needed so that $\boldsymbol{\Omega}$ is not a function of $\mathbf{v}_k$ and is well justified on ARCHER as impact angle weakly effects impact time.

Assuming a controlled invariant manifold $\mathcal{M}_\psi$, we now have the notion of discrete-time zero dynamics:

Definition 5.5. The *discrete-time zero dynamics* associated with a controlled invariant manifold $\mathcal{M}_\psi$ are given by:

$$\mathbf{z}_{k+1} = \mathbf{\Omega}(\boldsymbol{\psi}(\mathbf{z}_k), \mathbf{z}_k).$$

These dynamics are autonomous but determined by choice of $\boldsymbol{\psi}$; therefore, the goal of this work will be to design $\boldsymbol{\psi}$ such that the zero dynamics are stable.

Discrete-Time Zero Dynamics Policies

We propose a discrete-time mapping from the underactuated state, $\mathbf{z}_k$, to a desired actuated state, $\boldsymbol{\eta}_k$. This mapping, $\boldsymbol{\psi} : \mathcal{Z} \to \mathcal{N}$, will encode the desired position of the actuated coordinates given the location of the unactuated coordinates at impact. The job of the continuous time controller is to drive $\boldsymbol{\eta}(t)$ to the desired pre-impact location, $\boldsymbol{\psi}(\mathbf{z}_{k+1})$.

In this section, we will first reason about the ability of continuous time controllers to render $\mathcal{M}_\psi$ attractive and invariant by driving the error $\mathbf{e}$ to zero. Second, we demonstrate that if the manifold has stable zero dynamics (trajectories on the manifold converge to the origin), then stabilizing the manifold stabilizes the entire system. Finally, we propose a learning pipeline which leverages optimal control to find a manifold with the desired properties.

Constructive Stabilization of the Zeroing Manifold

We first show that the structure of the proposed manifold allows constructive stabilization techniques:

Lemma 5.6. *Consider a controlled invariant manifold $\mathcal{M}_\psi$. There exists a continuous-time control law $\mathbf{k} \in \mathcal{K}$ which results in exponential stabilization of $\|\boldsymbol{\eta}_k - \boldsymbol{\psi}(\mathbf{z}_k)\|$.*

Proof. Consider a point $(\boldsymbol{\eta}_k, \mathbf{z}_k)$ and the evaluation of the current and next states on the manifold: $\boldsymbol{\psi}(\mathbf{z}_k)$ and $\boldsymbol{\psi}(\mathbf{z}_{k+1})$, respectively. As the $\boldsymbol{\eta}(t)$ dynamics are feedback linearizeable, there exists a dynamically feasible trajectory $\boldsymbol{\eta}_d(t)$ such that $\boldsymbol{\eta}_d(0) = (\boldsymbol{\psi}(\mathbf{z}_k))^+$, and $\boldsymbol{\eta}_d(t_k) = \boldsymbol{\psi}(\mathbf{z}_{k+1})$, where t_k is the impact time and $(\cdot)^+$ denotes a post-impact state. For example, $\boldsymbol{\eta}_d(t)$ can be constructed using Bézier

polynomials [79]. Using a controller $\mathbf{k} \in \mathcal{K}$, i.e., satisfying the RES-CLF condition (4.5), we can obtain exponential convergence to this trajectory in continuous time:

$$\|\boldsymbol{\eta}(t) - \boldsymbol{\eta}_d(t)\| \leq Me^{-\frac{\lambda}{\varepsilon}t}\|\boldsymbol{\eta}_k^+ - (\boldsymbol{\psi}(\mathbf{z}_k))^+\|,$$

for $M, \lambda > 0$. Taking $T_* > 0$ to be the lower bound between impact times, the impact states are uniformly bounded by:

$$\|\boldsymbol{\eta}_{k+1} - \boldsymbol{\psi}(\mathbf{z}_{k+1})\| \leq Me^{-\frac{\lambda}{\varepsilon}T_*}\|\boldsymbol{\eta}_k^+ - (\boldsymbol{\psi}(\mathbf{z}_k))^+\|.$$

Then, using the properties of the impact map we have:

$$\begin{aligned}\|\boldsymbol{\eta}_k^+ - (\boldsymbol{\psi}(\mathbf{z}_k))^+\| &= \|\boldsymbol{\Delta}_{\boldsymbol{\eta}}(\boldsymbol{\eta}_k, \mathbf{z}_k) - \boldsymbol{\Delta}_{\boldsymbol{\eta}}(\boldsymbol{\psi}(\mathbf{z}_k), \mathbf{z}_k)\| \\ &\leq L_\Delta\|\boldsymbol{\eta}_k - \boldsymbol{\psi}(\mathbf{z}_k)\|,\end{aligned}$$

substituting into the bound above, and choosing $\varepsilon > 0$ sufficiently small that $\alpha = ML_\Delta e^{-\frac{\lambda}{\varepsilon}T_*} \in (0, 1]$, we have:

$$\|\boldsymbol{\eta}_{k+1} - \boldsymbol{\psi}(\mathbf{z}_{k+1})\| \leq \alpha\|\boldsymbol{\eta}_k - \boldsymbol{\psi}(\mathbf{z}_k)\|,$$

proving exponential stability to the manifold, as desired. □

Remark: *The desired trajectory $\boldsymbol{\eta}_d(t)$ is being implicitly replanned at impact via $\boldsymbol{\psi}$ as a function of the underactuated state $\mathbf{z}_k$.*

Composite Stability

The previous subsection demonstrated a method for constructing a controller to exponentially stabilize the system to a controlled invariant manifold $\mathcal{M}_{\boldsymbol{\psi}}$. We now show that exponentially stabilizing the system to a manifold with stable zero dynamics results in composite exponential stability of the entire system:

Theorem 5.7. *Consider a controlled invariant manifold $\mathcal{M}_{\boldsymbol{\psi}}$ whose zero dynamics are exponentially stable. Any control law exponentially stabilizing $\|\boldsymbol{\eta}_k - \boldsymbol{\psi}(\mathbf{z}_k)\|$ stabilizes the discrete-time composite system $(\boldsymbol{\eta}_k, \mathbf{z}_k)$ to the origin.*

Proof. Define $\mathbf{e}_k = \boldsymbol{\eta}_k - \boldsymbol{\psi}(\mathbf{z}_k)$. By Lemma 1, there exists a continuous-time controller $\mathbf{k} \in \mathcal{K}$ rendering the discrete error dynamics exponentially stable. As such, converse Lyapunov theory guarantees the existence of a Lyapunov function $V_{\mathbf{e}} : \mathcal{E} \to \mathbb{R}$ satisfying:

$$\begin{aligned}k_1\|\mathbf{e}_k\|^2 \leq V_{\mathbf{e}}(\mathbf{e}_k) &\leq k_2\|\mathbf{e}_k\|^2 \\ \Delta V_{\mathbf{e}}(\mathbf{e}_k) &\leq -k_3\|\mathbf{e}_k\|^2.\end{aligned}$$

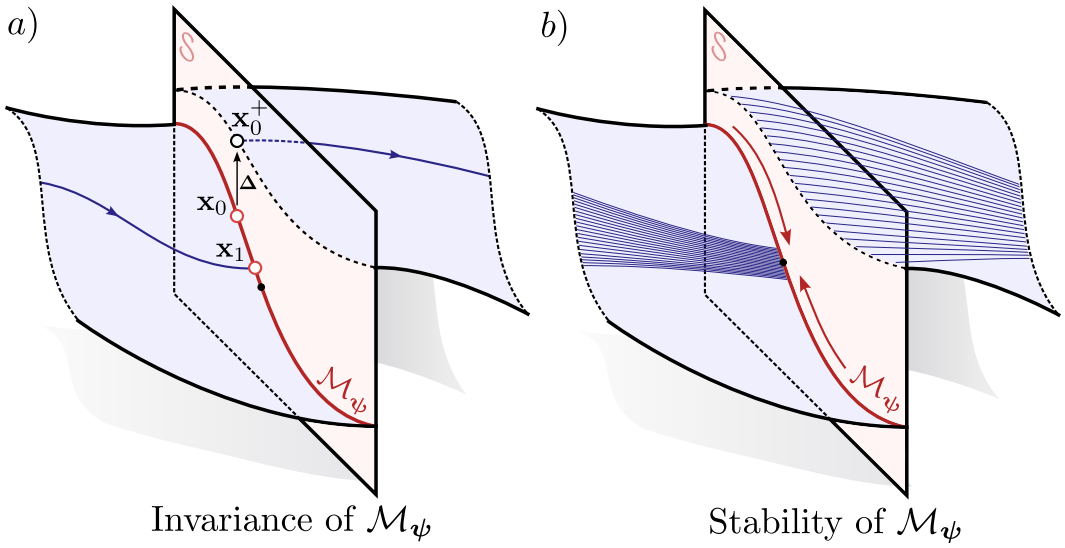


Figure 5.18: A depiction of the two necessary properties of $\mathcal{M}_\psi$: **a)** invariance under the discrete map $\mathbf{F}$, and **b)** stability.

Similarly, the stability of $\mathcal{M}_\psi$ implies the existence of a Lyapunov function $V_\mathbf{z} : \mathcal{Z} \to \mathbb{R}$ satisfying:

$$k_4\|\mathbf{z}_k\|^2 \leq V_\mathbf{z}(\mathbf{z}_k) \leq k_5\|\mathbf{z}_k\|^2$$
$$\Delta V_\mathbf{z}(\mathbf{z}_k) = V_\mathbf{z}(\boldsymbol{\Omega}(\boldsymbol{\psi}(\mathbf{z}_k), \mathbf{z}_k)) - V_\mathbf{z}(\mathbf{z}_k) \leq -k_6\|\mathbf{z}_k\|^2.$$

The Lyapunov function $V_\mathbf{z}$ will additionally satisfy [114]:

$$|V_\mathbf{z}(\mathbf{z}) - V_\mathbf{z}(\mathbf{z}')| \leq k_7\|\mathbf{z} - \mathbf{z}'\| \left(\|\mathbf{z}\| - \|\mathbf{z}'\|\right) \triangleq \Gamma(\mathbf{z}, \mathbf{z}').$$

Consider the composite Lyapunov function candidate $V(\mathbf{e}_k, \mathbf{z}_k) \triangleq \sigma V_\mathbf{e}(\mathbf{e}_k) + V_\mathbf{z}(\mathbf{z}_k)$ with $\sigma > 0$, whereby:

$$\min\{\sigma k_1, k_4\}\|\mathbf{e}, \mathbf{z}\|^2 \leq V(\mathbf{e}, \mathbf{z}) \leq \max\{\sigma k_2, k_5\}\|\mathbf{e}, \mathbf{z}\|^2.$$

Furthermore, since $\mathbf{z}_k$ is exponentially stable on $\mathcal{M}_\psi$, discrete sequences on $\mathcal{M}_\psi$ will be exponentially decreasing:

$$\|\mathbf{z}_{k+1}\| = \|\boldsymbol{\Omega}(\boldsymbol{\psi}(\mathbf{z}_k), \mathbf{z}_k)\| \leq M\lambda\|\mathbf{z}_k\|,$$

for $\lambda \in [0, 1)$ and $M > 0$. Compute the difference of ΔV:

$$\begin{aligned}
\Delta V &= \sigma \Delta V_{\mathbf{e}}(\mathbf{e}_k) + V_{\mathbf{z}}(\boldsymbol{\Omega}(\boldsymbol{\eta}, \mathbf{z}_k)) - V_{\mathbf{z}}(\mathbf{z}_k) \\
&= \sigma \Delta V_{\mathbf{e}}(\mathbf{e}_k) + \Delta V_{\mathbf{z}}(\mathbf{z}_k) \\
&\quad + V_{\mathbf{z}}(\boldsymbol{\Omega}(\boldsymbol{\eta}_k, \mathbf{z}_k)) - V_{\mathbf{z}}(\boldsymbol{\Omega}(\boldsymbol{\psi}(\mathbf{z}_k), \mathbf{z}_k)) \\
&\leq -\sigma k_1 \|\mathbf{e}_k\|^2 - k_6 \|\mathbf{z}_k\|^2 \\
&\quad + \Gamma(\boldsymbol{\Omega}(\boldsymbol{\eta}_k, \mathbf{z}_k), \boldsymbol{\Omega}(\boldsymbol{\psi}(\mathbf{z}_k), \mathbf{z}_k)) \\
&= -\sigma k_1 \|\mathbf{e}_k\|^2 - k_6 \|\mathbf{z}_k\|^2 \\
&\quad + k_7 L_{\boldsymbol{\Omega}}^2 \|\mathbf{e}_k\|^2 + 2M\lambda k_7 L_{\boldsymbol{\Omega}} \|\mathbf{e}_k\| \|\mathbf{z}_k\| \\
&= - \begin{bmatrix} \|\mathbf{e}_k\| \\ \|\mathbf{z}_k\| \end{bmatrix}^\top \begin{bmatrix} \frac{\sigma k_1}{2} - c(\sigma) & -M\lambda k_7 L_{\boldsymbol{\Omega}} \\ -M\lambda k_7 L_{\boldsymbol{\Omega}} & k_6 \end{bmatrix} \begin{bmatrix} \|\mathbf{e}_k\| \\ \|\mathbf{z}_k\| \end{bmatrix}
\end{aligned}$$

where $c(\sigma) = k_7 L_{\boldsymbol{\Omega}}^2 - \frac{\sigma}{2} k_1$, and $\Gamma(\boldsymbol{\Omega}(\boldsymbol{\eta}, \mathbf{z}), \boldsymbol{\Omega}(\boldsymbol{\psi}(\mathbf{z}), \mathbf{z}))$ is bounded using Lipschitz properties of the dynamics. Choosing $\sigma > \max \left\{ \frac{2M^2\lambda^2 k_7^2 L_{\boldsymbol{\Omega}}^2}{k_1 k_6}, \frac{2k_7 L_{\boldsymbol{\Omega}}^2}{k_1} \right\}$ ensures the matrix is positive definite; therefore, V is a Lyapunov function certifying composite stability. □

Remark: *Figure 5.18 depicts each of the assumptions used to prove stability in Theorem 5.7, namely discrete invariance and exponential stability of $\mathcal{M}_\psi$. Subsequent sections will develop constructive techniques leveraging optimal control and learning for finding such manifolds.*

Trajectory Tracking

The manifold, $\mathcal{M}_\psi$, is designed to stabilize the origin. With suitable modification, we can use this same manifold to track a time-varying reference trajectory on the underactuated coordinate, $\bar{\mathbf{z}}_k$, which is a common goal of legged systems whose underactuated coordinates tend to be center of mass related. We begin by defining a related time-varying manifold $\mathcal{M}_\psi^k \triangleq \{(\boldsymbol{\eta}_k, \mathbf{z}_k) \mid \boldsymbol{\eta}_k = \boldsymbol{\psi}(\mathbf{z}_k - \bar{\mathbf{z}}_k)\}$, and demonstrating that this manifold can be rendered control invariant as well:

Corollary 5.8. *Consider a controlled invariant manifold $\mathcal{M}_\psi$ and a discrete reference trajectory $\{\bar{\mathbf{z}}_k\}$ for $k = 1, \ldots, N$ for $N \in \mathbb{Z}_+$. There exists a continuous-time control law $\mathbf{k} \in \mathcal{K}$ which renders the time-varying manifold $\mathcal{M}_\psi^k$ controlled invariant and results in exponential stabilization of $\|\boldsymbol{\eta}_k - \boldsymbol{\psi}(\mathbf{z}_k - \bar{\mathbf{z}}_k)\|$.*

Proof. The proof is similar to Lemma 5.6, just updating the desired signal to $\bar{\mathbf{z}}_{k+1}$ instead of a constant $\bar{\mathbf{z}}$. □

Composite ISS

The zeroing manifold associated with the trajectory $\bar{\mathbf{z}}_k$ can be exponentially stabilized by the constructions of Corollary 5.8. Define the normed state difference of the trajectory:

$$\bar{v}_k \triangleq \|\bar{\mathbf{z}}_{k+1} - \bar{\mathbf{z}}_k\|.$$

We next demonstrate that if the manifold $\mathcal{M}_\psi$ has exponentially stable zero dynamics, then the manifold $\mathcal{M}_\psi^k$ has input to state stable zero dynamics. To accomplish this, we require the following assumption:

Assumption 5.9. The system dynamics, $\mathbf{F}, \mathbf{\Omega}$, are invariant under translation along the desired trajectory $\bar{\mathbf{z}}_k$, i.e.:

$$\begin{aligned}\mathbf{F}(\boldsymbol{\eta}_k, \mathbf{z}_k, \mathbf{v}_k) &= \mathbf{F}(\boldsymbol{\eta}_k, \mathbf{z}_k - \bar{\mathbf{z}}_k, \mathbf{v}_k)\\ \mathbf{\Omega}(\boldsymbol{\eta}_k, \mathbf{z}_k) &= \mathbf{\Omega}(\boldsymbol{\eta}_k, \mathbf{z}_k - \bar{\mathbf{z}}_k).\end{aligned}$$

The above assumption is often satisfied for legged robots, where underactuated states are translational center of mass states, which do not effect the system dynamics.

Lemma 5.10. *If the zero dynamics on $\mathcal{M}_\psi$ are exponentially stable, then there exists a Lyapunov function $V_{\mathbf{z}} : \mathcal{Z} \times \mathbb{Z} \to \mathbb{R}_{\geq 0}$ which satisfies:*

$$\begin{aligned}c_1\|\mathbf{z}_k - \bar{\mathbf{z}}_k\|^2 \leq V(\mathbf{z}_k, k) \leq c_2\|\mathbf{z}_k - \bar{\mathbf{z}}_k\|^2\\ \Delta V_{\mathbf{z}}(\mathbf{z}_k, k) \leq -c_3\|\mathbf{z}_k - \bar{\mathbf{z}}_k\|^2 + c_4\bar{v}_k\|\mathbf{z}_k - \bar{\mathbf{z}}_k\| + c_5\bar{v}_k^2.\end{aligned}$$

for constants $c_1, c_2, c_3, c_4, c_5 > 0$.

Proof. Observe that the next state implied by the mapping $\mathbf{z}_{k+1} = \mathbf{\Omega}(\boldsymbol{\psi}(\mathbf{z}_k - \bar{\mathbf{z}}_k), \mathbf{z}_k)$. Given that the zero dynamics on $\mathcal{M}_\psi$ are exponentially stable, we know that there exists a function $\tilde{V}_{\mathbf{z}}$ satisfying:

$$\Delta\tilde{V}_z(\mathbf{z}_k) = \tilde{V}_{\mathbf{z}}(\mathbf{z}_{k+1}) - \tilde{V}_{\mathbf{z}}(\mathbf{z}_k) \leq -k_3\|\mathbf{z}_k\|^2 \tag{5.38}$$

$$\tilde{V}_{\mathbf{z}}(\mathbf{z}) - \tilde{V}_{\mathbf{z}}(\mathbf{z}') \leq k_4\|\mathbf{z} - \mathbf{z}'\|(\|\mathbf{z}\| + \|\mathbf{z}'\|) \tag{5.39}$$

for $k_3 > 0$. By Assumption 5.9, we know that this inequality holds as well if we shift the $\mathbf{z}$ coordinates at any point on the trajectory, $\mathbf{z}^* = \bar{\mathbf{z}}_k$:

$$\Delta\tilde{V}_z(\mathbf{z}_k - \mathbf{z}^*) \leq -k_3\|\mathbf{z}_k - \mathbf{z}^*\|^2,$$

Furthermore, from this stability property, we know that

$$\|\mathbf{z}_{k+1} - \mathbf{z}^*\| \leq \beta \|\mathbf{z}_k - \mathbf{z}^*\| \tag{5.40}$$

for some $\beta \in [0, 1)$. We propose a new Lyapunov function, $V_{\mathbf{z}}(\mathbf{z}_k, k) = \tilde{V}_{\mathbf{z}}(\mathbf{z}_k - \bar{\mathbf{z}}_k)$. With this, denoting $\Delta V_{\mathbf{z}}(\mathbf{z}_k, k)$ as $\Delta V_{\mathbf{z}}$ for compactness, we have:

$$\begin{aligned}
\Delta V_{\mathbf{z}} &= \tilde{V}_{\mathbf{z}}(\mathbf{z}_{k+1} - \bar{\mathbf{z}}_{k+1}) - \tilde{V}_{\mathbf{z}}(\mathbf{z}_k - \bar{\mathbf{z}}_k) \\
&\leq -k_3 \|\mathbf{z}_k - \bar{\mathbf{z}}_k\|^2 + \tilde{V}_{\mathbf{z}}(\mathbf{z}_{k+1} - \bar{\mathbf{z}}_{k+1}) - \tilde{V}_{\mathbf{z}}(\mathbf{z}_{k+1} - \bar{\mathbf{z}}_k) \\
&\leq -k_3 \|\mathbf{z}_k - \bar{\mathbf{z}}_k\|^2 + k_4 (\|\mathbf{z}_{k+1} - \bar{\mathbf{z}}_{k+1}\| + \|\mathbf{z}_{k+1} - \bar{\mathbf{z}}_k\|) \bar{v}_k \\
&\leq -k_3 \|\mathbf{z}_k - \bar{\mathbf{z}}_k\|^2 + 2k_4 \beta \|\mathbf{z}_k - \bar{\mathbf{z}}_k\| \bar{v}_k + k_4 \bar{v}_k^2,
\end{aligned}$$

where the third line uses (5.39), and the last line uses (5.40) and:

$$\begin{aligned}
\|\mathbf{z}_{k+1} - \bar{\mathbf{z}}_{k+1}\| &= \|\mathbf{z}_{k+1} - \bar{\mathbf{z}}_{k+1} \pm \bar{\mathbf{z}}_k\| \\
&\leq \|\mathbf{z}_{k+1} - \bar{\mathbf{z}}_k\| + \|\bar{\mathbf{z}}_{k+1} - \bar{\mathbf{z}}_k\| \\
&\leq \bar{v}_k + \beta \|\mathbf{z}_k - \bar{\mathbf{z}}_k\|.
\end{aligned}$$

Setting $c_3 = k_3$, $c_4 = 2k_4\beta$, $c_5 = k_4$ yields the desired result. □

The previous result demonstrates that although the mapping $\boldsymbol{\psi}$ implies that the origin is exponentially stable, when we use this policy to track $\bar{\mathbf{z}}_k$ we can only converge to a neighborhood of the desired trajectory on the zeroing manifold. Next, we show that an exponentially convergent output feedback controller to a desired trajectory implies input to state stability of the combined system:

Lemma 5.11. *Consider a controlled invariant manifold $\mathcal{M}_{\boldsymbol{\psi}}$ whose zero dynamics are exponentially stable. Any control law exponentially stabilizing $\|\boldsymbol{\eta}_k - \boldsymbol{\psi}(\mathbf{z}_k - \bar{\mathbf{z}}_k)\| \to 0$ renders the signal $(\mathbf{0}, \bar{\mathbf{z}}_k)$ input to state stable.*

Proof. Consider the error on the output coordinates $\mathbf{e}_k^\eta = \boldsymbol{\eta}_k - \boldsymbol{\psi}(\mathbf{z}_k - \bar{\mathbf{z}}_k)$ and define the error on the zero dynamics coordinates as $\mathbf{e}_k^z = \mathbf{z}_k - \bar{\mathbf{z}}_k$. Next, consider the composite discrete-time Lyapunov function:

$$V(\mathbf{e}_k^\eta, \mathbf{e}_k^z, k) = \sigma V_{\mathbf{e}}(\mathbf{e}_k^\eta, k) + V_{\mathbf{z}}(\mathbf{z}_k, k)$$

for $\sigma > 0$. Then, the difference equation is given by:

$$\begin{aligned}
\Delta V &= \sigma \Delta V_{\mathbf{e}}(\mathbf{e}_k^{\eta}) + \tilde{V}_{\mathbf{z}}(\boldsymbol{\Omega}(\boldsymbol{\eta}_k, \mathbf{z}_k) - \bar{\mathbf{z}}_{k+1}) - \tilde{V}_{\mathbf{z}}(\mathbf{e}_k^{z}) \\
&\leq \sigma \Delta V_{\mathbf{e}}(\mathbf{e}_k^{\eta}) + \Delta V_{\mathbf{z}}(\mathbf{z}_k, k) \\
&\quad + V_{\mathbf{z}}(\boldsymbol{\Omega}(\boldsymbol{\eta}_k, \mathbf{z}_k) - \bar{\mathbf{z}}_{k+1}) \\
&\quad - V_{\mathbf{z}}(\boldsymbol{\Omega}(\boldsymbol{\psi}(\mathbf{e}_k^{z}), \mathbf{z}_k) - \bar{\mathbf{z}}_{k+1}) \\
&\leq -\sigma k_3 \|\mathbf{e}_k^{\eta}\|^2 - c_3 \|\mathbf{e}_k^{z}\|^2 + c_4 \bar{v}_k \|\mathbf{e}_k^{z}\| + c_5 \bar{v}_k^2 \\
&\quad + k_4 \|\mathbf{e}_k^{\eta}\| (L_{\boldsymbol{\Omega}} \|\mathbf{e}_k^{\eta}\| + (L_{\boldsymbol{\Omega}}(1 + L_{\boldsymbol{\psi}}) + \beta) \|\mathbf{e}_k^{z}\| + \bar{v}_k)
\end{aligned}$$

where the last line uses (5.39) and:

$$\begin{aligned}
\|\boldsymbol{\Omega}(\boldsymbol{\eta}_k, \mathbf{z}_k) - \bar{\mathbf{z}}_{k+1}\| &= \|\boldsymbol{\Omega}(\boldsymbol{\eta}_k, \mathbf{z}_k) \pm \boldsymbol{\Omega}(\boldsymbol{\psi}(\mathbf{e}_k^{z}), \mathbf{z}_k) \\
&\qquad \pm \boldsymbol{\Omega}(\boldsymbol{\psi}(\mathbf{0}), \bar{\mathbf{z}}_k) - \bar{\mathbf{z}}_{k+1}\| \\
&\leq L_{\boldsymbol{\Omega}} \|\mathbf{e}_k^{\eta}\| + L_{\boldsymbol{\Omega}}(1 + L_{\boldsymbol{\psi}}) \|\mathbf{e}_k^{z}\| + \bar{v}_k.
\end{aligned}$$

Letting $\mathbf{e}_k \triangleq \begin{bmatrix} \|\mathbf{e}_k^{\eta}\| & \|\mathbf{e}_k^{z}\| \end{bmatrix}^{\top}$ and $c > 0$ be a positive constant, we would like to enforce the condition $\Delta V \leq -c\, \mathbf{e}_k^{\top} \mathbf{e}_k$, which can be equivalently defined as:

$$-\mathbf{e}_k^{\top} \mathbf{M} \mathbf{e}_k + \mathbf{n}^{\top} \mathbf{e}_k \bar{v}_k + c_5 \bar{v}_k^2 \leq 0$$

for matrix $\mathbf{M} \in \mathbb{R}^{2 \times 2}$ and vector $\mathbf{n} \in \mathbb{R}^2$ defined as:

$$\begin{aligned}
\mathbf{M} &= \begin{bmatrix} \sigma k_3 - k_7 L_{\boldsymbol{\Omega}} - c & -\frac{1}{2} k_7 (L_{\boldsymbol{\Omega}}(1 + L_{\boldsymbol{\psi}}) + \beta) \\ -\frac{1}{2} k_7 (L_{\boldsymbol{\Omega}}(1 + L_{\boldsymbol{\psi}}) + \beta) & c_3 - c \end{bmatrix} \\
\mathbf{n} &= \begin{bmatrix} k_7 L_{\boldsymbol{\Omega}} & c_4 \end{bmatrix}^{\top}.
\end{aligned}$$

Ensuring that σ is chosen sufficiently large such that $\mathbf{M}$ has positive eigenvalues, this condition is enforced if:

$$-\lambda_M \|\mathbf{e}_k\|^2 + \|\mathbf{n}\| \|\mathbf{e}_k\|_2 \bar{v}_k + c_5 \bar{v}_k^2 \leq 0$$

for λ_M the largest eigenvalue of M. Completing the square:

$$\|\mathbf{e}_k\|_2 \geq \underbrace{\frac{1}{\sqrt{\lambda_M}} \left(\sqrt{\frac{\|\mathbf{n}\|}{4\lambda_M} + c_5} + \frac{\|\mathbf{n}\|}{2\sqrt{\lambda_M}} \right)}_{\triangleq \alpha} \bar{v}_k$$

yields:

$$\|\mathbf{e}_k\|_2 \geq \alpha \bar{v}_k \implies \Delta V \leq -c\, \mathbf{e}_k^{\top} \mathbf{e}_k.$$

which is the desired ISS conditions. □

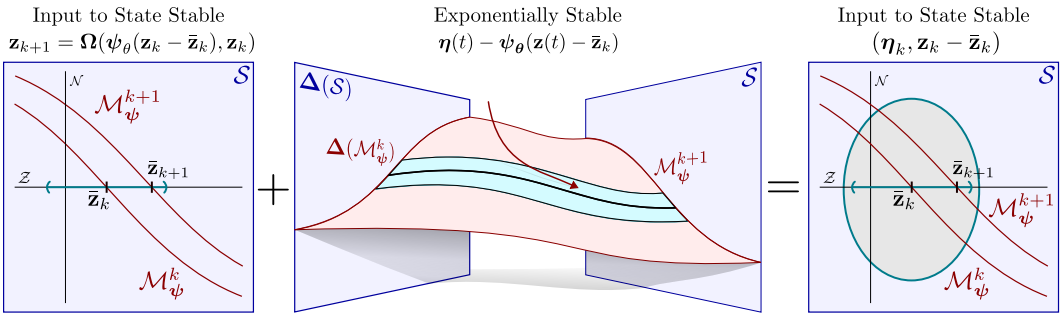


Figure 5.19: A depiction of Lemma 5.11, whereby input to state stability in the zero dynamics coordinates and exponential stability in the output coordinates yields input to state stability of the composite system.

Figure 5.19 illustrates the theoretical components developed in this section. The left pane depicts the input to state stable discrete time zero dynamics coordinates, as established in Lemma 5.10. The middle pane shows the exponentially stable output coordinates generated in Corollary 5.8. Finally, the right pane shows the result of their combination, demonstrating that the composite system is input to state stable, thereby yielding the desired forward invariant error tracking tube.

Stability via Optimal Control

What we have shown thus far is that if we have a mapping $\boldsymbol{\psi}$ which induces an output zeroing manifold with exponentially stable zero dynamics, then we can conclude stability of the overall system to an equilibrium point, or input to state stability of the overall system to a desired trajectory on the underactuated (center of mass) coordinates. In this section, we demonstrate that stable zero dynamics can be generated by leveraging the following discrete-time optimal control problem:

$$\begin{aligned} V(\mathbf{x}_0) \triangleq \min_{\mathbf{x}_k, \mathbf{v}_k} \quad & \sum_{k=0}^{\infty} c(\mathbf{x}_k, \mathbf{v}_k) \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k; \mathbf{v}_k) \\ & \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k) \leq 0 \end{aligned} \tag{5.41}$$

where $V : \mathcal{X}_{\mathrm{L}} \to \mathbb{R}$ is the positive-definite value function, $c : \mathcal{X}_{\mathrm{L}} \times P \to \mathbb{R}$ is a positive-semidefinite cost function and $\mathbf{h} : \mathcal{X}_{\mathrm{L}} \times P \to \mathbb{R}^p$ contains any state and input constraints.

Remark: *This choice of controller could be solved for in real time, as we did in the previous sections. Instead, in this section we will investigate if there is a feedback policy which aligns with* (5.41).

We will leverage optimality to enforce the stability on $\mathcal{M}_\psi$. This choice is motivated by the fact that asymptotic stability is a necessary condition for an optimal controller to be well defined [186]. As Theorem 5.7 rests on assumptions of exponential stability, we define conditions under which optimality implies exponential stability:

Theorem 5.12. *Let $V(\mathbf{x}_k)$ be the value function for the optimal control problem defined in Equation* (5.41)*, where the cost function is quadratic, $c(\mathbf{x}_k, \mathbf{v}_k) = \mathbf{x}_k^\top \mathbf{Q}\mathbf{x}_k + \mathbf{v}_k^\top \mathbf{R}\mathbf{v}_k$, and the domain $\mathcal{X}$ is compact. If there exists an $\varepsilon > 0$ such that the LQR approximation of Equation* (5.41) *taken by linearizing the dynamics around the equilibrium point satisfies:*

$$\mathbf{v}_{LQR}(\mathbf{x}_k) = -\mathbf{K}\mathbf{x}_k \in \mathcal{H}(\mathbf{x}_k) \quad \forall \mathbf{x}_k \in B_\varepsilon(\mathbf{0}), \tag{5.42}$$

with $\mathcal{H}(\mathbf{x}_k) \triangleq \{\mathbf{v}_k \in P \mid \mathbf{h}(\mathbf{x}_k, \mathbf{v}_k) \leq 0\}$, then the nonlinear system is exponentially stable under the optimal controller.

Proof. See [187]. □

Remark: *As exponential stability is preserved through diffeomorphism [26], the resulting zero dynamics are rendered exponentially stable.*

Since analytic forms for a mapping $\boldsymbol{\psi}$ are impossible to construct in general, we rely on either 1) linearizations of (5.41) to construct a local approximation of $\boldsymbol{\psi}$, or 2) machine learning solutions to infer a neural network representation of $\boldsymbol{\psi}$. First, consider the linearization of a constraint-free version of (5.41) in $\boldsymbol{\eta}$ and $\mathbf{z}$ coordinates:

$$\begin{aligned} \min_{\boldsymbol{\eta}_k, \mathbf{z}_k, \mathbf{v}_k} \quad & \sum_{k=0}^{\infty} \begin{bmatrix} \boldsymbol{\eta}_k \\ \mathbf{z}_k \end{bmatrix}^\top \mathbf{Q} \begin{bmatrix} \boldsymbol{\eta}_k \\ \mathbf{z}_k \end{bmatrix} + \mathbf{v}_k^\top \mathbf{R}\mathbf{v}_k \\ \text{s.t.} \quad & \begin{bmatrix} \boldsymbol{\eta}_{k+1} \\ \mathbf{z}_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{\eta\eta} & \mathbf{A}_{\eta\mathbf{z}} \\ \mathbf{A}_{\mathbf{z}\eta} & \mathbf{A}_{\mathbf{z}\mathbf{z}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\eta}_k \\ \mathbf{z}_k \end{bmatrix} + \begin{bmatrix} \mathbf{B}_\eta \\ \mathbf{0} \end{bmatrix} \mathbf{v}_k. \end{aligned} \tag{5.43}$$

This can be solved exactly via discrete time LQR to produce a gain matrix $\mathbf{K}$, from which the optimal closed loop linear dynamics can be written as:

$$\begin{bmatrix} \boldsymbol{\eta}_{k+1} \\ \mathbf{z}_{k+1} \end{bmatrix} = \underbrace{\left(\begin{bmatrix} \mathbf{A}_{\eta\eta} & \mathbf{A}_{\eta\mathbf{z}} \\ \mathbf{A}_{\mathbf{z}\eta} & \mathbf{A}_{\mathbf{z}\mathbf{z}} \end{bmatrix} - \begin{bmatrix} \mathbf{B}_\eta \\ \mathbf{0} \end{bmatrix} \mathbf{K} \right)}_{\triangleq \mathbf{A}_{cl}} \begin{bmatrix} \boldsymbol{\eta}_k \\ \mathbf{z}_k \end{bmatrix}.$$

From this, perform an eigendecomposition of $\mathbf{A}_{\text{cl}}$, and select a $\dim(\mathcal{Z})$ subspace that is not orthogonal to $\mathbf{z}$ spanned by eigenvectors $\mathbf{s}_1, \ldots, \mathbf{s}_{\dim(\mathcal{Z})}$. Defining a matrix $\mathbf{S} = [\mathbf{s}_1 \ \cdots \ \mathbf{s}_{\dim(\mathcal{Z})}]$, we can produce a map $\boldsymbol{\psi}$ as:

$$\boldsymbol{\psi}(\mathbf{z}) = \mathbf{S}\left(\begin{bmatrix}\mathbf{0} & \mathbf{I}\end{bmatrix}\mathbf{S}\right)^{-1} \triangleq \mathbf{S}_{\boldsymbol{\eta}}\mathbf{z}. \tag{5.44}$$

This associates a desired output coordinate $\boldsymbol{\eta}$ with the invariant subspace spanned by the selected eigenvectors $\mathbf{s}_i$. Interested readers should refer to [188] for a complete discussion of the continuous time case.

Leveraging LQR works as a local solution, but if we want more aggressive behaviors, or the ability to include constraints, we need to solve the full optimization problem. The difference between LQR and the full optimization problem is seen in the vector fields Figure 2.9 and Figure 2.10. As an alternative to LQR, we present a learning method that leverages the full optimal control problem (5.41) to ensure both controlled invariance and stability of $\mathcal{M}_{\boldsymbol{\psi}}$ by producing a policy that is invariant under the optimal action. To this end, consider the variable:

$$\boldsymbol{\zeta}_{\boldsymbol{\theta}}(\mathbf{z}) \triangleq \begin{bmatrix}\boldsymbol{\psi}_{\boldsymbol{\theta}}(\mathbf{z}) \\ \mathbf{z}\end{bmatrix} \tag{5.45}$$

where $\boldsymbol{\psi}_{\boldsymbol{\theta}}$ is a neural network parameterization of the output $\boldsymbol{\psi}$. Noting that $\boldsymbol{\zeta}_{\boldsymbol{\theta}}$ encodes a point on the manifold, we can define our loss function:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\mathbf{z}\sim\text{UNIFORM}} \left\|\boldsymbol{\eta}_1^*\left(\boldsymbol{\zeta}_{\boldsymbol{\theta}}(\mathbf{z})\right) - \boldsymbol{\psi}\left(\mathbf{z}_1^*\left(\boldsymbol{\zeta}_{\boldsymbol{\theta}}(\mathbf{z})\right)\right)\right\|_2^2, \tag{5.46}$$

where $\mathbf{z}_1^* = \boldsymbol{\Omega}(\boldsymbol{\psi}(\mathbf{z}), \mathbf{z})$ and $\boldsymbol{\eta}_1^* = \hat{\mathbf{F}}(\boldsymbol{\psi}(\mathbf{z}), \mathbf{z}; \mathbf{v}^*)$, with $\mathbf{v}^*$ the optimal control input. The expectation is taken over a uniform distribution over $\mathcal{Z}$, therefore enabling a dimension reduction as compared to the complete state space. The loss function directly measures how far an initial condition on the manifold deviates from the manifold under one discrete step of the optimal controller as depicted in Figure 5.20.

The learning pipeline outlined in Algorithm 2 starts an epoch by sampling a batch of points from $\mathcal{Z}$. The network is then evaluated to produce a set of points on the current manifold, $\{\boldsymbol{\zeta}_{\boldsymbol{\theta}}(\mathbf{z}_i)\}_{i=1}^N$. We then approximately solve the optimal control problem Equation (5.41) using iLQR, as outlined in Section 2.5. Finally, we simulate the system forwards one step to obtain $(\boldsymbol{\eta}_1^*, \mathbf{z}_1^*)$ which the loss computation in Equation (5.46) requires. If $\boldsymbol{\psi}$ attains zero loss, because of continuity of the network and the loss function we can conclude that the resulting manifold $\mathcal{M}_{\boldsymbol{\psi}}$ is invariant under the optimal control and can render the full order system stable by

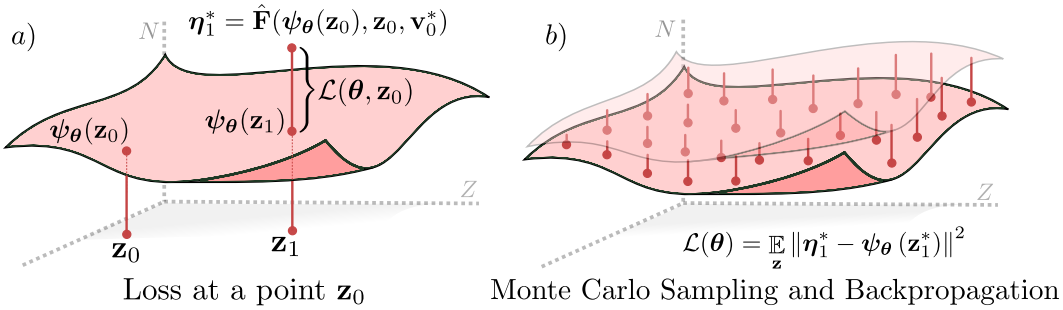


Figure 5.20: Loss function visualization: **a)** The loss function exactly measures the extent to which the manifold is not invariant under optimal action **b)** a Monte Carlo approximation of the spatial loss is used, wherein the optimal policy is backpropogated through to update the surface.

Algorithm 2 Monte Carlo Zero Dynamics Policy Training

hyperparameters: (Ξ, ρ, Υ)
Number of MC samples, Learning Rate and Number of Steps
Initialize $\boldsymbol{\theta}$ ▷ Pretrained with reasonable policy
for $i = 1 : \Upsilon$ **do**
  $\mathbf{z} \sim \text{UNIFORM}(\underline{\mathbf{z}}, \overline{\mathbf{z}})$
  $\boldsymbol{\zeta}_{\boldsymbol{\theta}} \leftarrow \begin{bmatrix} \boldsymbol{\psi}(\mathbf{z}) \\ \mathbf{z} \end{bmatrix}$ the
  $\mathbf{x}_0 \leftarrow \boldsymbol{\Phi}^{-1}(\boldsymbol{\zeta}_{\boldsymbol{\theta}})$
  $\mathbf{x}_{1:T}^*, \mathbf{v}_{1:T}^* \leftarrow \text{iLQR}(\mathbf{x}_0)$
  $\begin{bmatrix} \boldsymbol{\eta}_1^*(\boldsymbol{\zeta}_{\boldsymbol{\theta}}(\mathbf{z})) \\ \mathbf{z}_1^*(\boldsymbol{\zeta}_{\boldsymbol{\theta}}(\mathbf{z})) \end{bmatrix} \leftarrow \boldsymbol{\Phi}(\mathbf{x}_1)$
  $\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i - \rho \nabla_{\boldsymbol{\theta}} \sum_{\mathbf{z}} \|\boldsymbol{\eta}_1^*(\boldsymbol{\zeta}_{\boldsymbol{\theta}}(\mathbf{z})) - \boldsymbol{\psi}(\mathbf{z}_1^*(\boldsymbol{\zeta}_{\boldsymbol{\theta}}(\mathbf{z})))\|_2^2$
end for
return $\boldsymbol{\theta}$

satisfaction of the preconditions for Theorem 5.7. In order to evaluate this loss, we rely on iteratively solving convex approximations of the nonconvex problem via iLQR. This choice is motivated by the fact that backpropagating the loss requires computing $\frac{\partial \mathbf{v}_k^*}{\partial \mathbf{x}}$, which is straightforward for iLQR. To this end, the Lagrangian of the optimization program (5.41) is:

$$L(\mathbf{x}_k, \mathbf{v}_k, \boldsymbol{\lambda}_k) = c(\mathbf{x}_k, \mathbf{v}_k) + \boldsymbol{\lambda}_k^\top \mathbf{h}(\mathbf{v}_k, \mathbf{x}_k)$$

with $\boldsymbol{\lambda}_k \in \mathbb{R}^p$ the Lagrange multipliers. These can be determined from the stationary condition:

$$\mathbf{L}_{\mathbf{v}}(\mathbf{x}_k, \mathbf{v}_k^*, \boldsymbol{\lambda}_k^*) = 0,$$

where $\mathbf{L}_{\mathbf{v}} = \partial L / \partial \mathbf{v}$. Using the result of [189], the gradient of the optimal control input with respect to that state is given by:

$$\begin{bmatrix} \frac{\partial \mathbf{v}_k^*}{\partial \mathbf{x}} \\ \frac{\partial \boldsymbol{\lambda}_k^*}{\partial \mathbf{x}} \end{bmatrix} = - \begin{bmatrix} \mathbf{L}_{\mathbf{vv}} & \mathbf{h}_{\mathbf{v}}^\top \\ \text{diag}(\boldsymbol{\lambda}_k^*)\mathbf{h}_{\mathbf{v}} & \text{diag}(\mathbf{h}) \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{L}_{\mathbf{xv}} \\ \text{diag}(\boldsymbol{\lambda}_k^*)\mathbf{h}_{\mathbf{x}} \end{bmatrix} \tag{5.47}$$

which are evaluated at the optimal point $(\mathbf{x}_k, \mathbf{v}_k^*, \boldsymbol{\lambda}_k^*)$ and the arguments suppressed.

Application of ZDP to ARCHER

We deployed the ZDP method on the 3D hopping robot ARCHER. To discuss the application of ZDPs to ARCHER, consider the pose of the robot $\mathbf{q} = (\mathbf{p}, q) \in \mathcal{Q}$ where $\mathbf{p} \in \mathbb{R}^3$ represents the global position in world frame and $q \in \mathbb{S}^3$ the robot's orientation quaternion. Taking the velocities to be $\mathbf{v} = (\dot{\mathbf{p}}, \boldsymbol{\omega}) \in T_{\mathbf{q}}\mathcal{Q}$ for $\dot{\mathbf{p}} \in \mathbb{R}^3$ the global linear velocity and $\boldsymbol{\omega} \in \mathfrak{s}^3$ the body frame angular rates, we can represent the full state as $\mathbf{x} = (\mathbf{q}, \mathbf{v}) \in \mathcal{X} \triangleq T\mathcal{Q}$.

ARCHER evolves under hybrid dynamics. As such, its flight and ground phase dynamics are governed by (5.32) and it has two impact maps (one for the ground to flight transition, and another for flight to ground). We treat the vertical hopping as an autonomous system, and will focus on how to stabilize the position of the robot via orientation. The flight dynamics can be decomposed into actuated states, i.e., the orientation coordinates, and unactuated states, i.e., position coordinates:

$$\boldsymbol{\eta} = \begin{bmatrix} q \\ \boldsymbol{\omega} \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} \mathbf{p} \\ \dot{\mathbf{p}} \end{bmatrix}.$$

Take $(\boldsymbol{\eta}_k, \mathbf{z}_k)$ to be a pre-impact state. The ground phase does not depend on the control input, and the continuous-time evolution of the $\mathbf{z}$ coordinates has a weak dependence on the discrete-time control input $\mathbf{v}_k$. We can assume $\boldsymbol{\Omega}$ is independent from $\mathbf{v}_k$ because the effect of control inputs on impact time is negligible.

Online Control Implementation

Given a function $\boldsymbol{\psi}$, the controller aims to stabilize its associated zeroing manifold $\mathcal{M}_{\boldsymbol{\psi}}$. Consider a state $(\boldsymbol{\eta}(t), \mathbf{z}(t))$ during the flight phase. We set the desired orientation to $\boldsymbol{\eta}_d(t) = \boldsymbol{\psi}(\mathbf{z}(t))$, and update this continuously throughout the flight phase. The desired set point is converted to a quaternion, q_d, which we stabilize using the following quaternion PD controller in the flight phase:

$$\mathbf{u} = -\mathbf{K}_p \log(q_d^{-1} q) - \mathbf{K}_d \omega,$$

for suitable gains $\mathbf{K}_p, \mathbf{K}_d$. This controller is applied at 1kHz.

One key addition to the controller as compared to the previous section on MPC is the application of flywheel spindown in the ground phase. When the robot is in contact with the floor, the following control action is applied:

$$\mathbf{u} = -\gamma \dot{\boldsymbol{\vartheta}},$$

where $\dot{\boldsymbol{\vartheta}} \in \mathbb{R}^3$ represents the flywheel speed. This allows the system to maintain lower flywheel speeds and mitigates the problem of speed-torque constraints. This ground phase controller preserves the theoretical assumptions since the ground phase control is independent of output of the policy. Without spinning down the flywheels, they quickly reach their maximum speed after a few hops, leaving the system unable to apply stabilizing torques. Crucially, differences between ground and flight dynamics allow for spin-down control without destabilizing the system. During flight, torques are applied about the center of mass, whereas in the ground phase, torque is applied about the contact point. As the effective inertia is larger in the ground phase (as it is computed about the contact point), the spin-down controller can slow the flywheels without disturbing the system's orientation. The spin-down controller effectively maintains the controller bandwidth of the system and avoids the problem of flywheel saturation.

There are a few implementation differences from our theoretical implementation. The controller used in the proof of Lemma 5.6 differs from ours by (1) predicting the pre-impact state $\mathbf{z}_{k+1}$, (2) tracking a trajectory $\boldsymbol{\eta}_d(t)$ defined by a Bézier polynomial, and (3), using a RES-CLF. Empirically, a well tuned PD controller was sufficient to stabilize the continuous time system, and the feedforward input tracking that a trajectory would provide was not necessary.

ZDP Optimization and Learning Details

Notice that for discrete-time systems, Equation (5.41) is a nonlinear program even if the value function is available. To solve this optimal control problem, we employ Iterative LQR (iLQR), subject to box input constraints [190]. The iLQR problem is solved in the $\mathbf{x}$ variable, so the initial condition is obtain via $\mathbf{x} = \boldsymbol{\Phi}^{-1}(\boldsymbol{\eta}, \mathbf{z})$. We implemented Algorithm 2 in the JAX [191] and used a Network of 2 Layers with 256 hidden units each using ReLu activations. In our implementation of iLQR, we assume that the low level controller has perfect tracking and exactly achieves the desired angle with zero angular velocity. This considerably simplifies the flight dynamics and therefore the trajectory optimization, allowing them to be solved

Figure 5.21: A snapshot of the experiments conducted with ARCHER, including set point tracking, disturbance rejection, and hopping over rough terrain.

for in closed form. The input bounds $\mathcal{H}(\mathbf{x}_k)$ were chosen such that the torque applied during flight is bounded by the difference between the post-impact state and the desired pre-impact state. We require gradients of the optimal control, $\frac{d\mathbf{v}}{d\mathbf{x}}$, as presented in [192] — note that if no constraints are active, then this gradient is exactly the feedback matrix $\mathbf{K} = \mathbf{Q}_{\mathbf{vv}}^{-1}\mathbf{Q}_{\mathbf{vx}}$ from the iLQR algorithm.

Hardware Results

A collection of the experiments conducted on ARCHER can be seen in Figure 5.21. iLQR requires a stabilizing initial guess in order to converge; therefore, we use the LQR solution developed in (5.44) (which mirrors the Raibert heuristic) for the first rollout. To eliminate this dependence, other optimal control methods could be used,

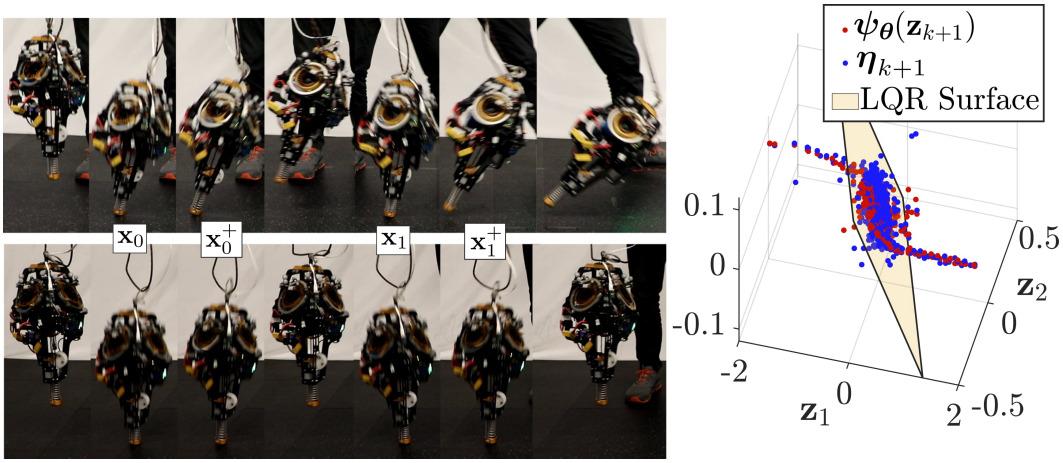


Figure 5.22: Comparing ZDP and LQR: **(Left)** A comparison between LQR (top) and ZDPs (bottom) while tracking a 2 m setpoint. **(Right)** The output of the trained policy and the actual state at impact over 3000 hops, as compared to an LQR controller.

for instance SQP. The authors experienced difficulty with the speed and accuracy of large-scale QP solvers in JAX and leveraged the fact that iLQR solves many small QPs for speed and stability. Additionally, for computational efficiency, we limit the number of iLQR iterations to five (empirically enough to obtain convergence for this system). The full code base for this project can be found at [172].

The policy ψ was exported from JAX to an ONNX file, which is evaluated at 1kHz on an Ubuntu 20.04 machine with AMD Ryzen 5950x @ 3.4 GHz and 64 Gb RAM and torques are passed directly to the robot over ethernet. This controller does not require this amount of compute to run, and could be feasibly implemented on an NVIDIA Jetson or comparable board. A Kalman filter with projectile dynamics is used to filter the position estimates from optitrack in the flight phase. The manif library [193] is used to compute the $\log$ map for the quaternion PD controller.

We logged over 3,000 stable hops when deploying the ZDP method on the ARCHER hardware platform, a selection of which can be seen in Figure 5.21 and in the supplemental video [170]. Figure 5.22 depicts the desired impact angle, i.e., the learned policy evaluation, and the actual impact angle over the complete collection of all hardware tests. In general, as predicted by the theory, this manifold is both invariant under the feedback controller, and stable. Also interesting to note is that around the origin, the learned policy aligns with LQR, as presented in Theorem 5.12. Notably, away from the origin, the learned policy diverges from LQR in order to

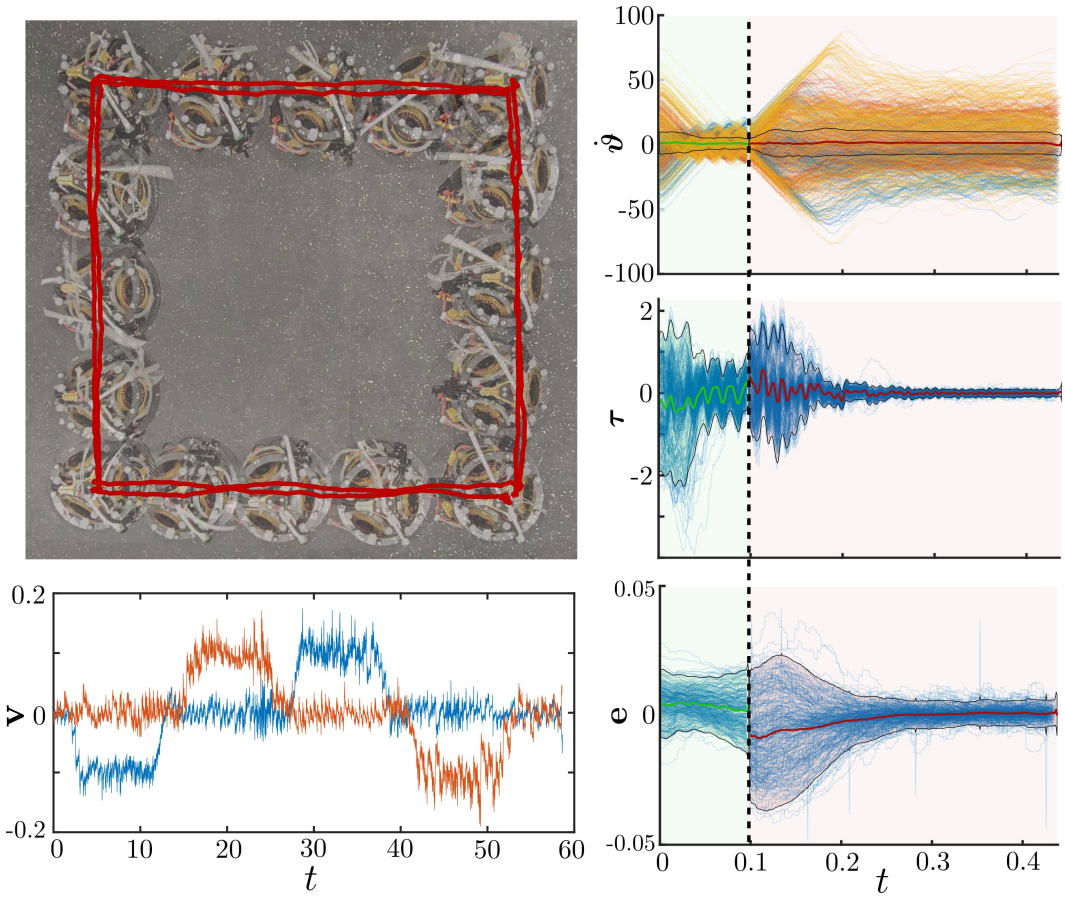


Figure 5.23: Square trajectory tracking: **(Left)** Overhead view with positional hardware data overlayed (top) and velocity tracking (bottom) and **(Right)** wheel velocities (top), torque (mid), and error (bottom) in the ground (green) and flight (red) phase with mean and 2σ deviation.

maintain stability under the enforced input constraints. A comparison between the trained policy and the application of a naive LQR controller when trying to track a setpoint 2 m away is seen in the left part of Figure 5.22, wherein ZDPs maintain stability by implicitly enforcing discrete invariance and optimality over a horizon.

The tight trajectory tracking and system behavior is seen in Figure 5.23, where ARCHER was asked to follow two laps of a 1 m square trajectory. As seen on the right of Figure 5.23, using a PD controller at the feedback level empirically resulted in the error (and therefore the torques) converging exponentially fast to a small neighborhood of zero during the flight phase. During this torque application, the flywheel speed can be seen to grow, while the ground phase controller is able to successfully regulate them close to zero.

For outside testing as shown in Figure 8.1, we took a different approach: linearizing the optimal control problem, and construct a linear mapping as in (5.44). Interestingly, as the underactuated coordinates for this system are position and velocity, this linear mapping is precisely a Raibert Heuristic [194], mapping positions and velocities to desired impact orientations. The Raibert Heuristic, critical in the stabilization of legged systems, was a primary motivation for the development of the ZDP. For the hopping robot, this process ultimately takes the form:

$$\boldsymbol{\eta}_d(t) = \text{EulToQuat}\left(-\mathbf{k}_p(\mathbf{z}(t) - \bar{\mathbf{z}}_k) - \mathbf{k}_d\dot{\mathbf{z}}(t)\right) \tag{5.48}$$

where $\mathbf{k}_p, \mathbf{k}_d \geq 0$ are positive gains of the ZDP. In practice, all signals are also clipped to prevent the robot from taking angles that are too aggressive.

Limitations

As training this policy involves querying the optimal control input and its gradients, each iteration of the training process is computationally expensive (2 seconds per iteration for a batch size of 30). iLQR requires a stabilizing controller to initialize the rollout and therefore only locally improves a stabilizing policy. Furthermore, to avoid sampling initial conditions in the training pipeline which the hopper cannot stabilize, the policy ψ was pretrained with a conservative Raibert heuristic.

5.6 Summary

This chapter discussed methods of producing trajectories which, when tracked, stabilize the underactuated (and therefore the full state) of legged robots. We investigated the use of offline trajectory optimization via hybrid zero dynamics, online trajectory optimization via model predictive control, and learning based methods via zero dynamics policies. All of these methods allow us to abstract the complexity of legged systems away, and focus on planning desired trajectories for the center of mass states that guide legged robots towards goals, which will be the problem of interest in the next chapter.

Chapter 6: High Level Tracking Layer

Contents

Robust feedback controllers at the low level enable efficient planning on states of reduced dimensionality at the high level.

This point in the thesis marks a significant philosophical transition: we shift our focus from real-time feedback control on the full order system to feedforward planning, i.e., the generation of paths that guide the system toward achieving specific goals. Having established controllers capable of stabilizing the underactuated dynamics of robotic systems, we now address the generation of trajectories that make progress towards goals which satisfying environmental constraints. A critical advantage of having robust feedback controllers at the lower levels is that they allow us to simplify the dynamics constraints at planning time — we now only need to plan paths for the center of mass states of the robot. This simplification, in turn, allows us to incorporate more complex task relevant constants such as spatial objectives and environmental constraints.

This section will be focused on answering the following two questions:

> **Question:** *How can we provide continuous-time guarantees for discrete plans, and how can we produce efficient representations of the set of states that a planner/tracker can reach?*

To address these, we turn to Bézier MPC, a direct collocation method for optimal control problems. Bézier MPC will allow us to derive computationally efficient underapproximations of reachable sets, which can be used as a proxy for connectivity between points for long horizon planning.

6.1 Bézier MPC

We begin by assuming that we are given a sequence of states $\{\bar{\mathbf{x}}_k\}$ and associated convex sets $\{\mathcal{X}_k\}$. The task at hand is to formulate an optimal control problem that provides explicit guarantees on continuous-time trajectories which connect these points, which is summarized by the following continuous-time optimization program:

$$\begin{aligned} \inf_{\mathbf{x}_d(\cdot),\mathbf{u}_d(\cdot)} \quad & \int_0^T \mathbf{x}_d(\tau)^\top \mathbf{Q}\mathbf{x}_d(\tau) + \mathbf{N}^\top \mathbf{x}_d(\tau) + \mathbf{u}_d(\tau)^\top \mathbf{R}\mathbf{u}_d(\tau) d\tau \qquad (6.1) \\ \text{s.t.} \quad & \mathbf{x}^{(\gamma)}(t) = \mathbf{f}_d(\mathbf{x}_d(t)) + \mathbf{g}_d(\mathbf{x}_d(t))\mathbf{u}_d(t) \\ & \mathbf{x}_d(t) \in \mathcal{X}_k, \ \ \forall t \in [kT/N, (k+1)T/N)\}, \ \ k = 0, \dots, N \\ & \mathbf{u}_d(t) \in \mathcal{U} \end{aligned}$$

where $\mathbf{x}_d(\cdot)$ is the desired state trajectory, $\mathbf{u}_d(\cdot)$ is the desired input trajectory, γ is the relative degree of the assumed dynamics, $\mathcal{U}$ is an input constraint set, and where the matrices $\mathbf{Q}$, $\mathbf{N}$, and $\mathbf{R}$ define quadratic costs on the state and input capturing deviation from the reference trajectory, path length, and state and input deviations.

Dynamics

While planning models can take many forms, we focus here on systems whose coordinates are given by $\mathbf{q}_d \in \mathbb{R}^m$, with state $\mathbf{x}_d = [\mathbf{q}_d^\top, \dot{\mathbf{q}}_d^\top, \ldots, \mathbf{q}_d^{(\gamma-1)\top}]^\top \in \mathcal{X}_{\mathrm{M}} \triangleq \mathbb{R}^n$ for some $\gamma \in \mathbb{N}$, and control-affine dynamics of the form:

$$\dot{\mathbf{x}}_d = \begin{bmatrix} \mathbf{0} & \mathbf{I}_{n-m} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{x}_d + \begin{bmatrix} \mathbf{0} \\ \mathbf{f}_d(\mathbf{x}_d) \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{g}_d(\mathbf{x}_d) \end{bmatrix} \mathbf{u}_d, \tag{6.2}$$

where $\mathbf{I}_{n-m}$ is an identity matrix of size $n-m$, $\mathbf{u}_d \in \mathcal{U}_{\mathrm{M}} \triangleq \mathbb{R}^m$ is the input, the drift vector $\mathbf{f}_d : \mathcal{X}_{\mathrm{M}} \to \mathbb{R}^m$ is locally Lipschitz continuous on $\mathbb{R}^n$, and the actuation matrix $\mathbf{g}_d : \mathcal{X}_{\mathrm{M}} \to \mathbb{R}^{m \times m}$ is invertible for all $\mathbf{x}_d \in \mathcal{X}_{\mathrm{M}}$.

In order to plan dynamically feasible trajectories for the system (6.2), we choose to parameterize the system evolution via Bézier curves. Recall from Property 2.45 that given a time $\tau > 0$, two points $\mathbf{x}_0, \mathbf{x}_\tau \in \mathbb{R}^n$, and order $p \geq 2\gamma - 1$, there exists a matrix $\mathbf{D} \in \mathbb{R}^{p+1 \times 2n}$ such that any curve $\mathbf{x}_d(\cdot)$ with control points satisfying:

$$\mathbf{pD} = \begin{bmatrix} \mathbf{x}_0^\top & \mathbf{x}_\tau^\top \end{bmatrix} \tag{6.3}$$

also satisfies $\mathbf{x}_d(0) = \mathbf{x}_0$ and $\mathbf{x}_d(\tau) = \mathbf{x}_\tau$. Furthermore, for this curve $\mathbf{x}_(\cdot)$ observe that applying the continuous input signal:

$$\mathbf{u}_d = \mathbf{g}_d(\mathbf{x}_d)^{-1} \Big(\mathbf{q}_d^{(\gamma)} - \mathbf{f}_d(\mathbf{x}_d) \Big), \tag{6.4}$$

results in integrator dynamics:

$$\dot{\mathbf{x}}_d = \begin{bmatrix} \mathbf{0} & \mathbf{I}_{n-m} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{x}_d.$$

As such, the pair $(\mathbf{x}_d, \mathbf{u}_d)$ for a state trajectory $\mathbf{x}_d$ generated via (6.3) and an input trajectory $\mathbf{u}_d$ from (6.4) represents a dynamically feasible trajectory for the planning system (6.2). The next challenge will be to ensure that the continuous time curve satisfies the desired state and input constraints.

State and Input Constraints

We are now interested in satisfying the constraints in the optimal control problem, namely $\mathbf{x}_d(t) \in \mathcal{X}$ and $\mathbf{u}_d(t) \in \mathcal{U}$.

In order to ensure that the trajectories designed at the high level satisfy the assumptions of the low level $\mathcal{A}_{\mathrm{L}}$ (as discussed in Section 3.6), we will need to impose additional constraints. We begin by defining a box input constraint set

$\mathcal{C}_\mathcal{U} \triangleq \{\mathbf{u} \in \mathbb{R}^m \mid \|\mathbf{u}\|_\infty \leq u_{max}\}$ for $u_{max} > 0$. Next, we define a convex state constraint set $\mathcal{C}_\mathcal{X} \triangleq \{\mathbf{x}_d \in \mathcal{X}_\text{M} \mid \mathbf{C}_x\mathbf{x}_d \leq \mathbf{d}_x\}$ with $\mathbf{C}_x \in \mathbb{R}^{p\times \dim(\mathcal{X}_\text{M})}$ and $\mathbf{d}_x \in \mathbb{R}^p$. Note that as $\mathcal{A}_L$ is nonempty and contains the origin in its interior, we can make $\mathcal{C}_\mathcal{U}$ and $\mathcal{C}_\mathcal{X}$ sufficiently small such that $(\mathbf{x}_d(t), \mathbf{u}_d(t)) \in \mathcal{C}_\mathcal{X} \times \mathcal{C}_\mathcal{U}$ for all $t \in [0, T]$ implies $(\mathbf{x}_d(t), \mathbf{u}_d(t)) \in \mathcal{A}_\text{L}$. To ensure tractability of the high level, we assume that the sets passed from the high level $\mathcal{X}_k$ are defined as convex polytopes, i.e., $\mathcal{X}_k = \{\bar{\mathbf{x}} \in \mathcal{X}_\text{H} \mid \mathbf{C}_k\bar{\mathbf{x}} \leq \mathbf{d}_k\}$ for all $k = 1, \ldots, N$, with $\mathbf{C}_k \in \mathbb{R}^{p\times \dim(\mathcal{X}_\text{H})}$ and $\mathbf{d}_k \in \mathbb{R}^p$. We now turn our attention to enforcing these constraints.

First, observe that the input constraints $\mathbf{u}_d \in \mathcal{C}_\mathcal{U}$ can be directly enforced by:

$$\|\mathbf{u}_d\|_\infty \leq u_{max}. \tag{6.5}$$

Next, consider the feasible state constraint set as:

$$\mathcal{Q}_\text{L} \triangleq \{\mathbf{x}_d \mid \mathbf{C}_x(\mathbf{x}_d + \mathbf{v}) \leq \mathbf{d}_x \text{ for all } \mathbf{v} \in \mathcal{E}_\text{L}\}.$$

Enforcing this for each row i separately and letting $\|\mathbf{C}_i\|_*$ represent the dual norm of $\|\cdot\|$, we have:

$$\begin{aligned}\mathcal{Q}_\text{L} &\subseteq \{\mathbf{x}_d \mid \mathbf{C}_{x,i}\mathbf{x}_d \leq \mathbf{d}_{x,i} - \sup_{\mathbf{v}\in\mathcal{E}_\text{L}} \mathbf{C}_{x,i}\mathbf{v}\}\\ &\subseteq \{\mathbf{x}_d \mid \mathbf{C}_{x,i}\mathbf{x}_d \leq \mathbf{d}_i - \|\mathbf{C}_{x,i}\|_* e\}\end{aligned}$$

where $e \triangleq \sup_{\mathbf{v}\in\mathcal{E}_\text{L}} \|\mathbf{v}\|$. Therefore, enforcing:

$$\mathbf{C}_x\mathbf{x}_d \leq \mathbf{d}_x - \mathbf{M}(\mathbf{C}_x)e$$

where $\mathbf{M} : \mathbb{R}^{p\times \dim(\mathcal{X}_\text{H})} \to \mathbb{R}^p$ is defined such that the i^{th} row $\mathbf{M}(\mathbf{C})_i = \|\mathbf{C}_i\|_*$ results in satisfaction of $\mathbf{x}_d \in \mathcal{C}_\mathcal{X}$. Next, consider the constraints imposed by the high level:

$$\mathcal{Q}_\text{H} \triangleq \{\mathbf{x}_d \mid \mathbf{C}_k(\mathbf{\Pi}^\text{M}_\text{H}(\mathbf{x}_d + \mathbf{v})) \leq \mathbf{d}_k \text{ for all } \mathbf{v} \in \mathcal{E}_\text{L}\}.$$

Using the same reformulation as above and recalling that $\mathbf{\Pi}^\text{M}_\text{H}$ is linear, we have:

$$\mathbf{C}_k\mathbf{\Pi}^\text{M}_\text{H}\mathbf{x}_d \leq \mathbf{d}_k - \mathbf{M}(\mathbf{C}_k\mathbf{\Pi}^\text{M}_\text{H})e. \tag{6.6}$$

With the above constructions, we reformulate these constraints onto an affine constraint on the Bézier curve:

Lemma 6.1. *Given a reference point $\bar{\mathbf{x}}_d \in \mathcal{X}_d$, a matrix $\mathbf{A} \in \mathbb{R}^{k\times n+2}$ and vector $\mathbf{b} \in \mathbb{R}^k$, there exists a matrix $\mathbf{L} \in \mathbb{R}^{4knm\times n}$ and a vector $\mathbf{h} \in \mathbb{R}^{4knm}$ such that:*

$$\mathbf{L}\begin{bmatrix}\mathbf{x}_d\\ \mathbf{q}_d^{(\gamma)}\end{bmatrix} \leq \mathbf{h} \implies \mathbf{A}\begin{bmatrix}\mathbf{x}_d\\ \|\mathbf{x}_d - \bar{\mathbf{x}}_d\|\\ \|\mathbf{u}_d\|\end{bmatrix} \leq \mathbf{b}.$$

Proof. We begin by bounding the term $\mathbf{u}_d(\cdot)$:

$$\|\mathbf{u}_d\| \leq \|\mathbf{g}_d(\mathbf{x}_d)^{-1}\|\|\mathbf{q}_d^{(\gamma)} - \mathbf{f}_d(\mathbf{x}_d)\|. \tag{6.7}$$

Taking $\bar{\mathbf{x}}_d \in \mathcal{X}_d$ to be a reference point in the planning state space, we can bound the first term by:

$$\|\mathbf{g}^{-1}(\mathbf{x}_d)\| \leq L_{\mathcal{G}}\|\mathbf{x}_d - \bar{\mathbf{x}}_d\| + \|\mathbf{g}^{-1}(\bar{\mathbf{x}}_d)\|, \tag{6.8}$$

where $L_{\mathcal{G}}$ is a Lipschitz constant of $\mathbf{g}^{-1}$ with respect to the ∞-norm on $\mathcal{C}_{\mathcal{X}}$, which is well defined by the local Lipschitz continuity and nonzero assumptions on $\mathbf{g}$ and the compactness of $\mathcal{C}_{\mathcal{X}}$. Similarly:

$$\|\mathbf{q}_d^{(\gamma)} - \mathbf{f}(\mathbf{x}_d)\| \leq L_{\mathbf{f}}\|\mathbf{x}_d - \bar{\mathbf{x}}_d\| + \|\mathbf{q}_d^{(\gamma)} - \mathbf{f}(\bar{\mathbf{x}}_d)\|. \tag{6.9}$$

Now, let $\mathbf{a} \triangleq \begin{bmatrix}\mathbf{a}_1 & a_2 & a_3\end{bmatrix}$ be a row of the constraint matrix $\mathbf{A}$ with $\mathbf{a}_1 \in \mathbb{R}^n$ and $a_2, a_3 \in \mathbb{R}$ and $b \in \mathbb{R}$ the corresponding entry of the vector $\mathbf{b}$. Substituting (6.8) and (6.9) into (6.7), we can construct a quadratic form:

$$\begin{bmatrix}a_2 & a_3\end{bmatrix}\begin{bmatrix}\|\mathbf{x}_d - \bar{\mathbf{x}}_d\| \\ \|\mathbf{u}_d\|\end{bmatrix} \leq \boldsymbol{\sigma}^\top \mathbf{M}\boldsymbol{\sigma}_d + \mathbf{N}^\top\boldsymbol{\sigma}_d,$$

where $\boldsymbol{\sigma}_d \triangleq \begin{bmatrix}\|\mathbf{x}_d - \bar{\mathbf{x}}_d\| & \|\mathbf{q}_d^{(\gamma)} - \mathbf{f}(\bar{\mathbf{x}}_d)\|\end{bmatrix}^\top$ and:

$$\mathbf{M} = \frac{a_3}{2}\begin{bmatrix}2L_{\mathcal{G}}L_{\mathbf{f}} & L_{\mathcal{G}} \\ L_{\mathcal{G}} & 0\end{bmatrix}, \quad \mathbf{N} = \begin{bmatrix}a_3 L_{\mathbf{f}}\|\mathbf{g}^{-1}(\bar{\mathbf{x}}_d)\| + a_2 \\ a_2\|\mathbf{g}^{-1}(\bar{\mathbf{x}}_d)\|\end{bmatrix}.$$

Next, consider $\widehat{\mathbf{M}}$ as the projection of $\mathbf{M}$ onto the positive semidefinite cone. With this, we can define the function $h : \mathcal{X}_d \times \mathbb{R}^m \to \mathbb{R}$ as:

$$h(\mathbf{x}_d, \mathbf{q}_d^{(\gamma)}) = \boldsymbol{\sigma}_d^\top\widehat{\mathbf{M}}\boldsymbol{\sigma}_d + \mathbf{N}^\top\boldsymbol{\sigma}_d + \mathbf{a}_1^\top\mathbf{x}_d.$$

Because $\mathbf{M}$ is symmetric, we have that $\widehat{\mathbf{M}} \preceq \mathbf{M}$. As such, points in the set $\boldsymbol{\Omega} \triangleq \{(\mathbf{x}_d, \mathbf{q}_d^{(\gamma)}) \mid h(\mathbf{x}_d, \mathbf{q}_d^{(\gamma)}) \leq b\}$ satisfy the desired inequality. Next, consider a function $\ell : \mathcal{X}_d \times \mathbb{R}^m \to \mathbb{R}$ of the form:

$$\ell(\mathbf{x}_d, \mathbf{q}_d^{(\gamma)}) = \mathbf{c}^\top\boldsymbol{\sigma}_d + \mathbf{a}_1^\top\mathbf{x}_d,$$

for some vector $\mathbf{c} \in \mathbb{R}^2$, along with the following optimization program:

$$\begin{aligned}\delta^* = \sup_{\delta\in\mathbb{R}} \quad & \delta \\ \text{s.t.} \quad & \ell(\mathbf{x}_d, \mathbf{q}_d^{(\gamma)}) \leq \delta \implies h(\mathbf{x}_d, \mathbf{q}_d^{(\gamma)}) \leq b.\end{aligned}$$

In general, this set containment problem may be challenging to solve; however, given the specific problem structure this can be solved for in closed form (the details of which can be found in [172]). Then, we have that the set $\mathbf{\Lambda} \triangleq \{(\mathbf{x}_d, \mathbf{q}_d^{(\gamma)}) \,|\, \ell(\mathbf{x}_d, \mathbf{q}_d^{(\gamma)}) \leq \delta^*\} \subset \mathbf{\Omega}$; therefore points in $\mathbf{\Lambda}$ satisfy the desired constraints.

Finally, we will show that there exists a matrix $\mathbf{L}_i \in \mathbb{R}^{4nm \times n+m}$ and a vector $\mathbf{h}_i \in \mathbb{R}^{4nm}$ such that:

$$\mathbf{L}_i \begin{bmatrix} \mathbf{x}_d \\ \mathbf{q}_d^{(\gamma)} \end{bmatrix} \leq \mathbf{h}_i \Rightarrow \ell(\mathbf{x}_d, \mathbf{q}_d^{(\gamma)}) \leq \delta^*.$$

Based on the definition of $\boldsymbol{\sigma}_d$, the set $\mathbf{\Lambda}$ is given by:

$$\mathbf{c}^\top \begin{bmatrix} \max_i |\mathbf{x}_d - \bar{\mathbf{x}}|_i \\ \max_i \left|\mathbf{q}_d^{(\gamma)} - \mathbf{f}(\bar{\mathbf{x}})\right|_i \end{bmatrix} + \mathbf{a}_1^\top \mathbf{x}_d \leq \delta^*,$$

which, taking $\mathbf{c}^\top = [c_1, c_2]$, is equivalent to:

$$\underbrace{\begin{bmatrix} c_1 & c_1 & -c_1 & -c_1 \\ c_2 & -c_2 & c_2 & -c_2 \end{bmatrix}^\top}_{\triangleq \mathbf{F}^\top} \begin{bmatrix} (\mathbf{x}_d - \bar{\mathbf{x}})_i \\ \left(\mathbf{q}_d^{(\gamma)} - \mathbf{f}(\bar{\mathbf{x}})\right)_j \end{bmatrix} + \mathbf{a}_1^\top \mathbf{x}_d \leq \boldsymbol{\delta}^*,$$

for all row pairs $i \leq n$ and $j \leq m$ and where $\boldsymbol{\delta}^* \triangleq \delta^* \otimes \mathbf{1}$ with $\otimes$ denoting the Kronecker product. Letting $\mathbf{L}_i \in \{0,1\}^{4nm \times n+m}$ be matrices capturing the i, j permutations of the scaling matrix $\mathbf{F}^\top$ above, we can reformulate this as:

$$\begin{bmatrix} \mathbf{L}_1 & \mathbf{L}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_d - \bar{\mathbf{x}} \\ \mathbf{q}_d^{(\gamma)} - \mathbf{f}(\bar{\mathbf{x}}) \end{bmatrix} + (\mathbf{a}_1^\top \otimes \mathbf{1})\mathbf{x}_d \leq \boldsymbol{\delta}^*,$$

which can be further rearranged as:

$$\underbrace{\begin{bmatrix} \mathbf{L}_1 + \mathbf{a}_1^\top \otimes \mathbf{1} & \mathbf{L}_2 \end{bmatrix}}_{\triangleq \mathbf{L}_i} \begin{bmatrix} \mathbf{x}_d \\ \mathbf{q}_d^{(\gamma)} \end{bmatrix} \leq \underbrace{\boldsymbol{\delta}^* + \begin{bmatrix} \mathbf{L}_1 & \mathbf{L}_2 \end{bmatrix} \begin{bmatrix} \bar{\mathbf{x}}_d \\ \mathbf{f}(\bar{\mathbf{x}}_d) \end{bmatrix}}_{\triangleq \mathbf{h}_i}.$$

Repeating this process for each of the k rows of the constraint matrix $\mathbf{A}$ yields the desired result. □

The previous Lemma demonstrates that the inequalities on the desired trajectory $\mathbf{x}_d(\cdot)$ imposed by state and input constraints can be framed as affine constraints on the space of possible trajectories. As curves are infinite dimensional objects, traditional trajectory optimizers that attempt to solve (6.1) directly would generally

only approximately enforce these constraints. This is precisely where we see the usefulness of Bézier curves — we can exactly enforce these constraints on the continuous-time curve by reasoning about a discrete, low-dimensional collection of Bézier control points:

Theorem 6.2. *There exist matrices* $\mathbf{F}$ *and* $\mathbf{G}$ *such that any Bézier curve* $\mathbf{B} : I \to \mathcal{X}_d$ *with control points* $\mathbf{p}$ *satisfying:*

$$\mathbf{F}\vec{\mathbf{p}} \leq \mathbf{G},$$

also satisfies $\mathbf{x}_d(t) \in \mathcal{C}_\mathcal{X}$ *and* $\mathbf{u}_d \in \mathcal{C}_\mathcal{U}$ *for all* $t \in I$.

Proof. Lemma 6.1 demonstrates that there exists a matrix $\mathbf{L}$ and vector $\mathbf{h}$ such that the state and input constraints (6.6) and (6.5) can be enforced via a linear inequality on the desired trajectory $\mathbf{x}_d$. Based on Property 2.44, we know that if we enforce this constraint on the control points, it will be enforced for the continuous time curve. Therefore, instead we must enforce:

$$\mathbf{L}\begin{bmatrix}(\mathbf{P})_j \\ (\mathbf{p}\mathbf{H}^\gamma)_j\end{bmatrix} \leq \mathbf{h}$$

for $j = 0, \ldots, p$. As this imposes linear constraints on the columns of $\mathbf{p}$, this can be vectorized and written as:

$$\mathbf{F}\vec{\mathbf{p}} \leq \mathbf{G},$$

where $\mathbf{F}$ and $\mathbf{G}$ are appropriate reformulations of $\mathbf{L}$ and $\mathbf{h}$ to account for the vectorization. Enforcing this constraint results in state and input constraint satisfaction as desired. □

Model Predictive Control Formulation

In this section we establish how to compute the collection of points $\{\mathbf{x}_k\}$ used to define $\mathbf{x}_d$ while meeting the desired constraints on the Bézier control points. With Theorem 6.2, we have a constraint on the desired trajectory, which if satisfied, implies state and input constraint satisfaction for the output of the high level. In addition to these constraints, we will be interested in enforcing a trust region constraint with respect to the reference signal $\{\bar{\mathbf{x}}_k\}$:

$$\|\mathbf{x}_k - \bar{\mathbf{x}}_k\|_1 \leq \epsilon \tag{6.11}$$

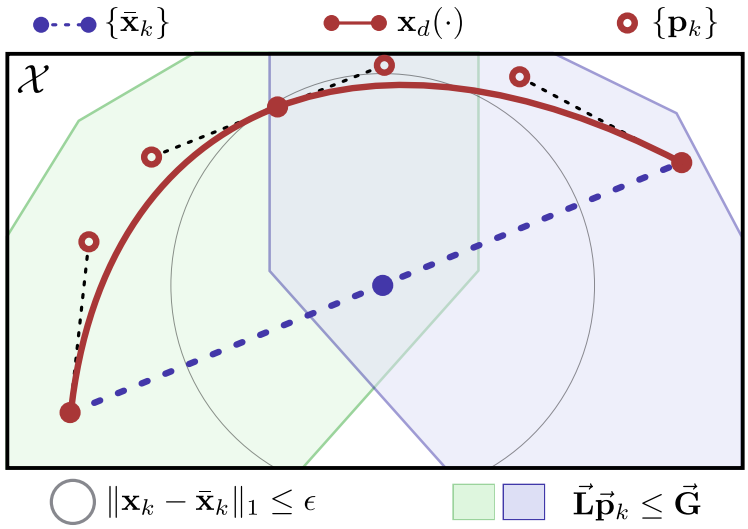


Figure 6.1: The constraints enforced in the Béz-MPC program, which optimizes over the Bézier control points subject to satisfying the continuous time state and input constraints.

for some $\epsilon > 0$. Note that this can be converted into a linear constraint during implementation. In order to improve the quality of the path $\bar{\mathbf{x}}_k$, we formulate an MPC program:

$$\min_{\mathbf{x}_k, \mathbf{p}_k} \quad \sum_{k=0}^{N-1} \mathbf{x}_k^\top \mathbf{Q} \mathbf{x}_k + \mathbf{F}\bar{\mathbf{x}}_k + \mathbf{x}_N^\top \mathbf{V} \mathbf{x}_N \tag{Béz-MPC}$$

$$\text{s.t.} \quad \mathbf{x}_0 = \bar{\mathbf{x}}_0, \quad \mathbf{x}_N = \bar{\mathbf{x}}_N \tag{6.12a}$$

$$\|\mathbf{x}_k - \bar{\mathbf{x}}_k\|_1 \leq \epsilon \tag{6.12b}$$

$$\vec{\mathbf{D}}\vec{\mathbf{p}}_k = \text{vec}([\mathbf{x}_k^\top \ \ \mathbf{x}_{k+1}^\top]) \tag{6.12c}$$

$$\vec{\mathbf{L}}\vec{\mathbf{p}}_k \leq \vec{\mathbf{G}} \tag{6.12d}$$

where $\mathbf{Q}, \mathbf{F} \in \mathbb{R}^{n \times n}$ are symmetric positive definite matrices weighting distance to reference as well as path length, $\mathbf{R} \in \mathbb{R}^{m \times m}$ is a positive definite input scaling matrix, and $\mathbf{V} \in \mathbb{R}^{n \times n}$ is a terminal cost. Solutions to this program can be converted to dynamically feasible curves $\mathbf{x}_d(\cdot)$ for the reduced order model (6.2) via (2.22).

Simulation

We consider the following nonlinear system in simulation:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \sin(x_1) + x_2^3 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \begin{bmatrix} w_1(t) \\ w_2(t) \end{bmatrix}.$$

The goal is to drive the system to the origin while satisfying state and input constraints for all time. Fig. 6.2 demonstrates that at different time scales, both with and without

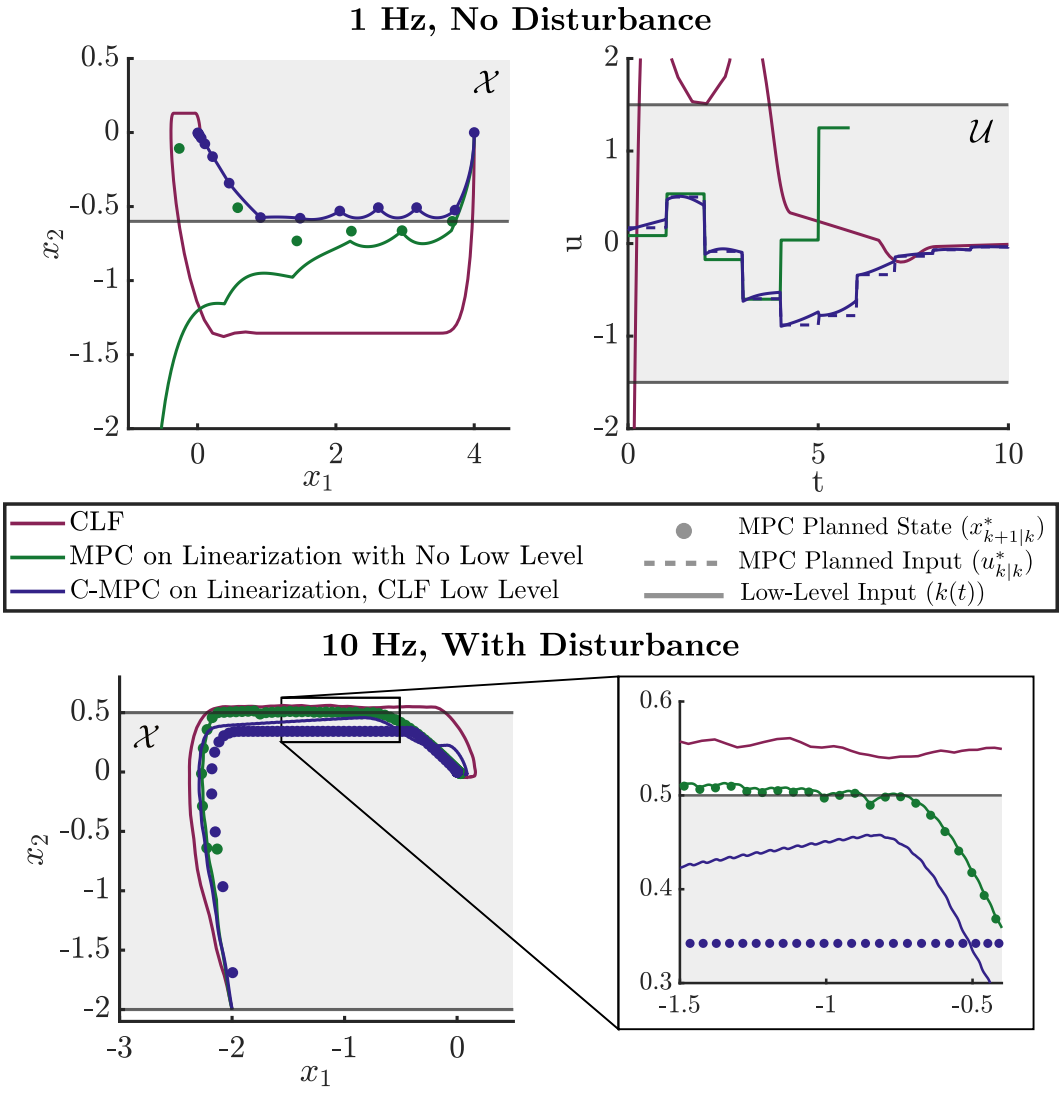


Figure 6.2: Comparison of three control methods: only using a low level controller (CLF), applying MPC with no low level controller, and applying the proposed Béz-MPC with a CLF at the low level. In both scenarios, just using the planning or tracking layers separately yields both state and input violation.

added disturbances, using only either a low level or mid level controller results in state and/or input violation, whereas the proposed combined approach is able to satisfy both for all time. The MPC program (Béz-MPC) can be seen in Figure 6.3. In the top left, we see how the reference path satisfying that assumption set of the mid level, $\mathcal{A}_{\mathrm{M}}$, is smoothed by the MPC program. In the top right, with a valid reference signal, the MPC escapes from a local minimum that many myopic planners would suffer from. In the bottom left, we observe that as MPC is done in the continuous

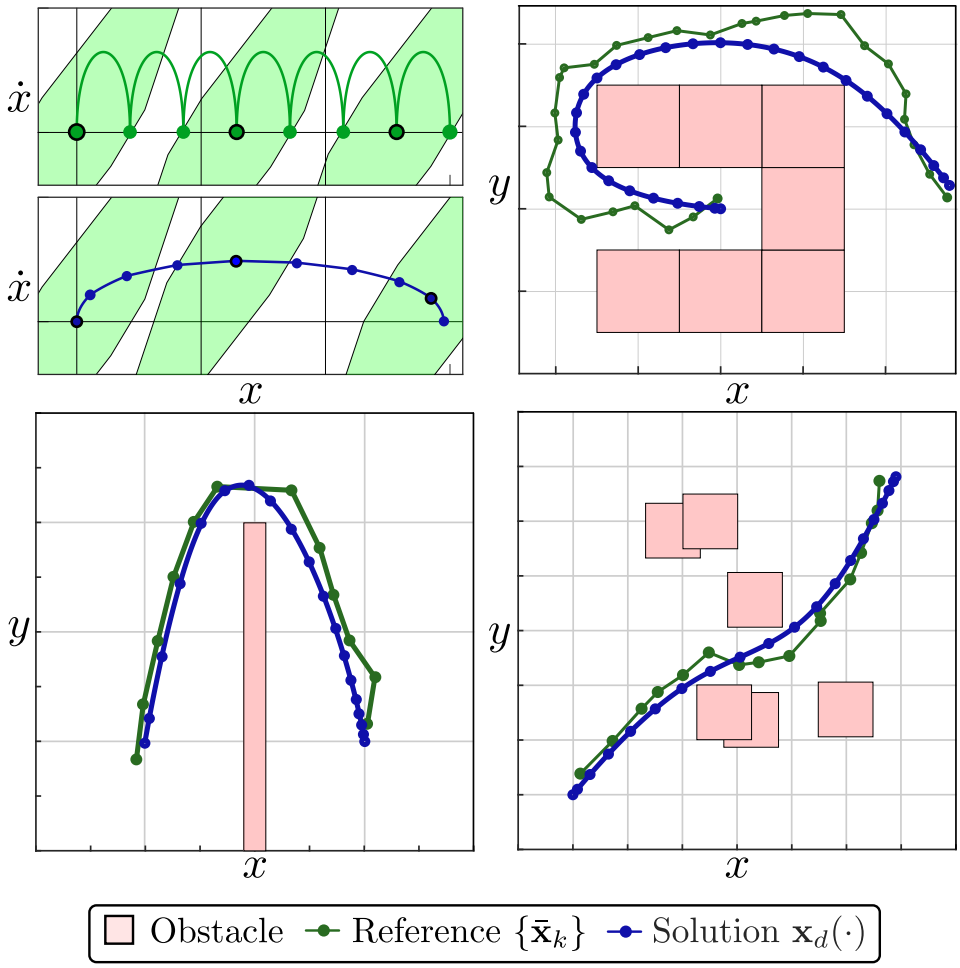


Figure 6.3: A depiction of the MPC program, which takes a sequence of reference states $\bar{\mathbf{x}}_k$ and refines them into a dynamically admissible path. **(Top Left)** Reference points with associated Bézier curves, with subsequent points satisfying the assumption set of the low level, $\mathcal{A}_{\mathrm{M}}$, as shown by the green polytope. **(Rest)** Various planning problems show MPC refining the reference.

time Bézier curve space, we are able to successfully traverse thin wall features that some numerical optimizers struggle with. Finally the bottom right shows a path through a randomly generated environment.

6.2 Bézier Reachable Polytopes

We now turn to the second question of this section:

> **Question:** *How can we efficiently approximate the set of states that a tracker can achieve?*

Given the constructions in Section 6.2, there exists an affine inequality that guaran-

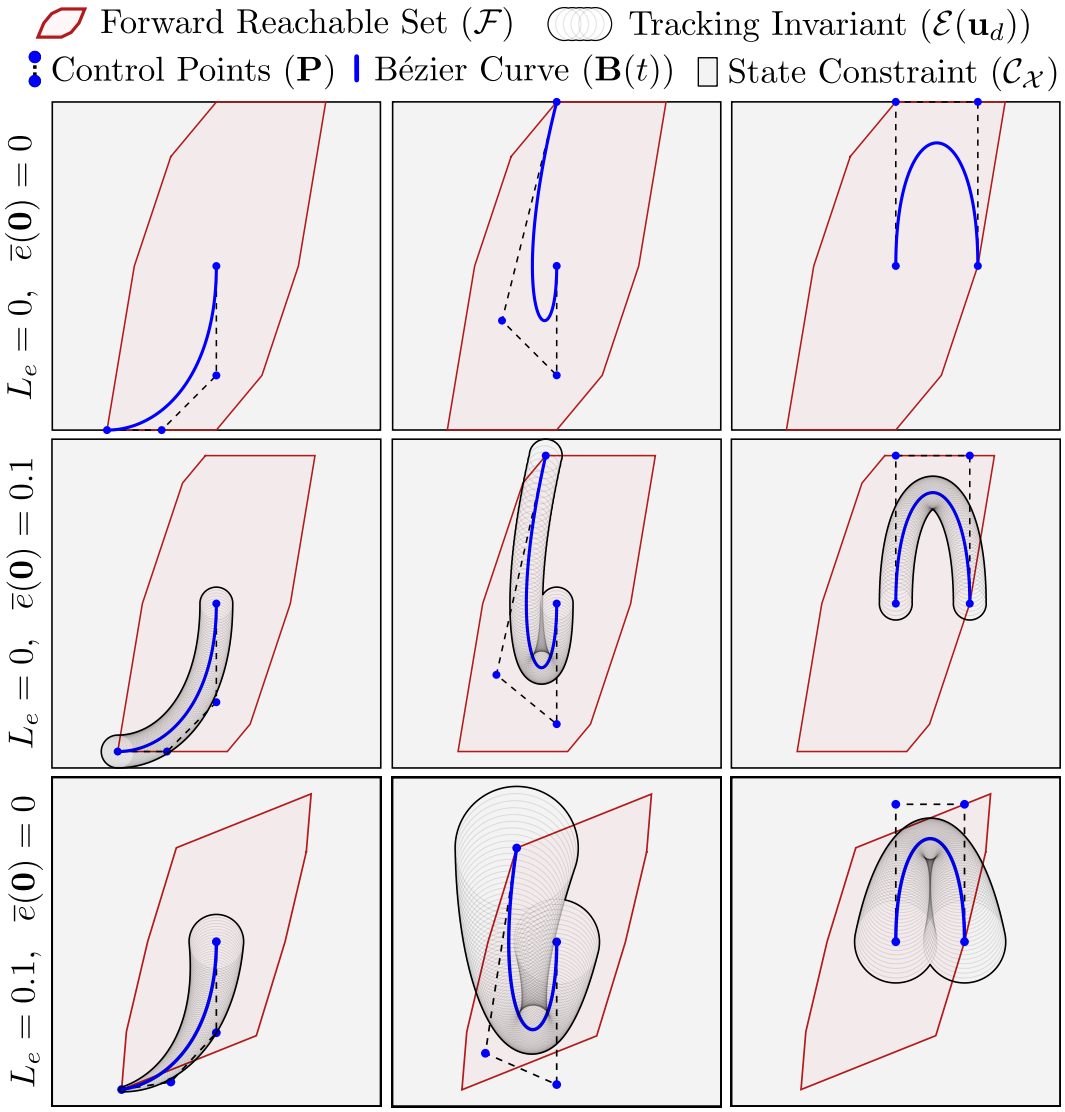


Figure 6.4: A selection of Bézier curves and forward reachable sets. The top row depicts curves with exact tracking, the middle row with a fixed size tracking certificate, and the bottom row with a tracking certificate whose upper bound scales linearly with the planning input $\mathbf{u}_d$.

tees the existence of a Bézier polynomial which results in the closed-loop planner-tracker system satisfying state and input constraints. The matrix $\mathbf{F}$ and vector $\mathbf{G}$ represent an efficient oracle to check whether Bézier curves connecting initial and terminal points satisfy these constraints. Combining this affine constraint with Property 2.45 allows us to place constraints on the desired boundary conditions of the

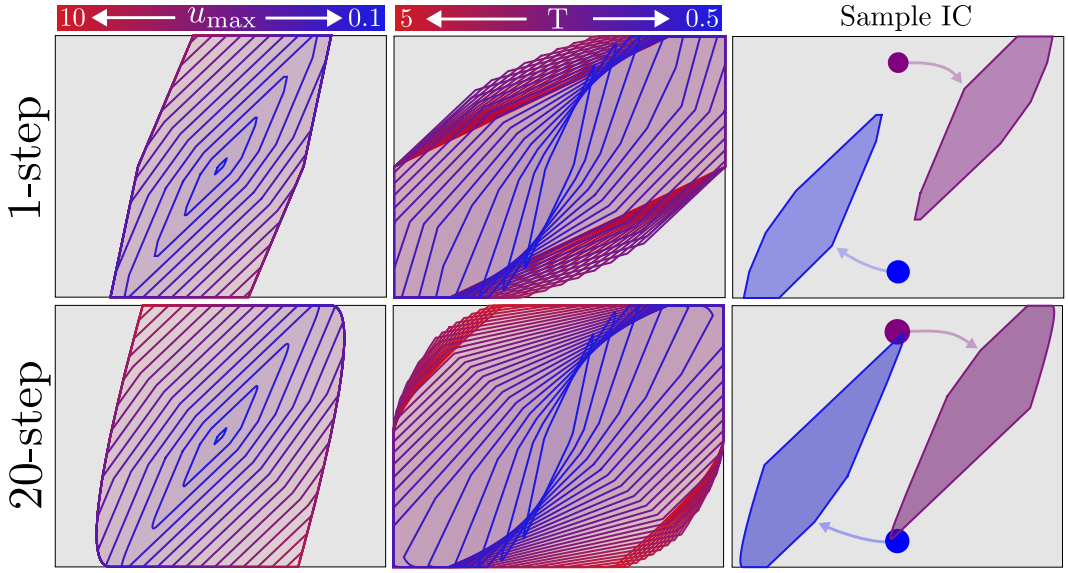


Figure 6.5: A depiction of the forward reachable sets as a function of system parameters. **(Top Row)** 1-step reachable sets, as in Theorem 6.2. **(Bottom Row)** 20-step reachable sets, as in Corollary 6.3. **(Left Column)** Varying the input constraint $\mathbf{u}_{\max}$ **(Middle Column)** Varying the time horizon T **(Right Column)** Varying the initial condition.

Bézier polynomial — that is, given an initial condition $\mathbf{x}_0$, the set characterized by:

$$\mathcal{F}(\mathbf{x}_0) = \{\mathbf{x}_d \in \mathcal{X}_d \mid \mathbf{F}\vec{\mathbf{D}}^\dagger \begin{bmatrix}\mathbf{x}_0^\top & \mathbf{x}_T^\top\end{bmatrix}^\top \leq \mathbf{G}\},$$

represents all terminal conditions for which there exists a feasible Bézier polynomial. As such, the set $\mathcal{F}(\mathbf{x}_0)$ can be thought of as the forward reachable set of the point $\mathbf{x}_0$. Similarly, given a terminal condition $\mathbf{x}_T$, the backward reachable set is characterized by:

$$\mathcal{B}(\mathbf{x}_T) = \{\mathbf{x}_0 \in \mathcal{X}_d \mid \mathbf{F}\vec{\mathbf{D}}^\dagger \begin{bmatrix}\mathbf{x}_0^\top & \mathbf{x}_T^\top\end{bmatrix}^\top \leq \mathbf{G}\}.$$

A depiction of the forward reachable set for a pendulum system and a variety of system parameters can be seen in Figure 6.4. As the error tracking tube $\mathcal{E}$ varies in its dependence on $\mathbf{u}_d$, the reachable sets change shape to ensure that closed loop system still satisfies the desired constraints.

Reducing Conservatism

In the previous discussion, we used a reference point $\bar{\mathbf{x}}_d$ and bounded the deviation of a trajectory from this point. While this enables tractability, it creates conservatism

in the bound as the same reference point was used over the entire trajectory $\mathbf{x}_d(\cdot)$. To resolve this conservatism, we would like to instead bound the trajectory with a collection of reference points $\{\bar{\mathbf{x}}_k\}$ spread out over the time interval $[0, T]$. Towards this goal, we leverage the notion of a k-refinement of the interval $[0, T]$ from Definition 2.46 as well as reference points $\{\bar{\mathbf{x}}_i\}$ for $i = 1, \dots k$ With these, we can construct a piecewise constant reference trajectory $\bar{\mathbf{x}}(t) = \bar{\mathbf{x}}_i$ for $t \in [T_{i-1}, T_i)$ with $i = 1, \dots, k$. With this reference trajectory, we have the following:

Corollary 6.3. *Let system Σ_d be a planning model for a system Σ with tracking certificate $\mathcal{E}$, and consider a piecewise-constant trajectory $\bar{\mathbf{x}}(t)$ defined with respect to a $k-$refinement of the interval $[0, T]$. There exist matrices $\widehat{\mathbf{F}}$ and $\widehat{\mathbf{G}}$ such that any Bézier curve $\mathbf{B} : I \to \mathcal{X}_d$ with control points $\mathbf{p}$ satisfying:*

$$\widehat{\mathbf{F}}\vec{\mathbf{p}} \leq \widehat{\mathbf{G}}, \tag{6.13}$$

when tracked results in the closed loop system satisfying $\mathbf{\Pi}(\mathbf{x}_{\text{cl}}(t)) \in \mathcal{C}_{\mathcal{X}}$ and $\mathbf{k}(\mathbf{x}_{\text{cl}}(t), \mathbf{x}_d(t)) \in \mathcal{C}_{\mathcal{U}}$ for all $t \in I$.

Proof. As refinement is linear in the control points, we can leverage the matrices from Theorem 6.2 and right multiply $\mathbf{F}$ by $\vec{\mathbf{Q}}_i$, the vectorized version of the refinement matrix $\mathbf{Q}_i$ for $i = 1, \dots, k$ to produce $\widehat{\mathbf{F}}$. Taking $\widehat{\mathbf{G}} = \widehat{\mathbf{G}}$ yields the desired result. □

By enforcing the constraint in (6.13), we are able to ensure that the desired trajectory stays close to the piecewise constant reference trajectory, as opposed to a single reference point. This will reduce the conservatism of the bound, but requires increasing the number of constraints needed (and therefore faces of the polytope), demonstrating an obvious trade-off. A depiction in the difference in resulting reachable sets can be seen in Figure 6.5. When a single points is used, the reachable set indicates the neighborhood around which that reference point can be feedback linearized, potentially requiring significant input over long time horizons. Instead, if we have a sequence of points, we can forward simulate the drift dynamics to produce reference trajectories, whereby the reachable set represents the neighborhood around the trajectory which we can converge to, thereby reducing conservatism. This notion is especially useful when using such reachable sets to represent an MPC layer, which often uses a sequence of reference points to linearize around.

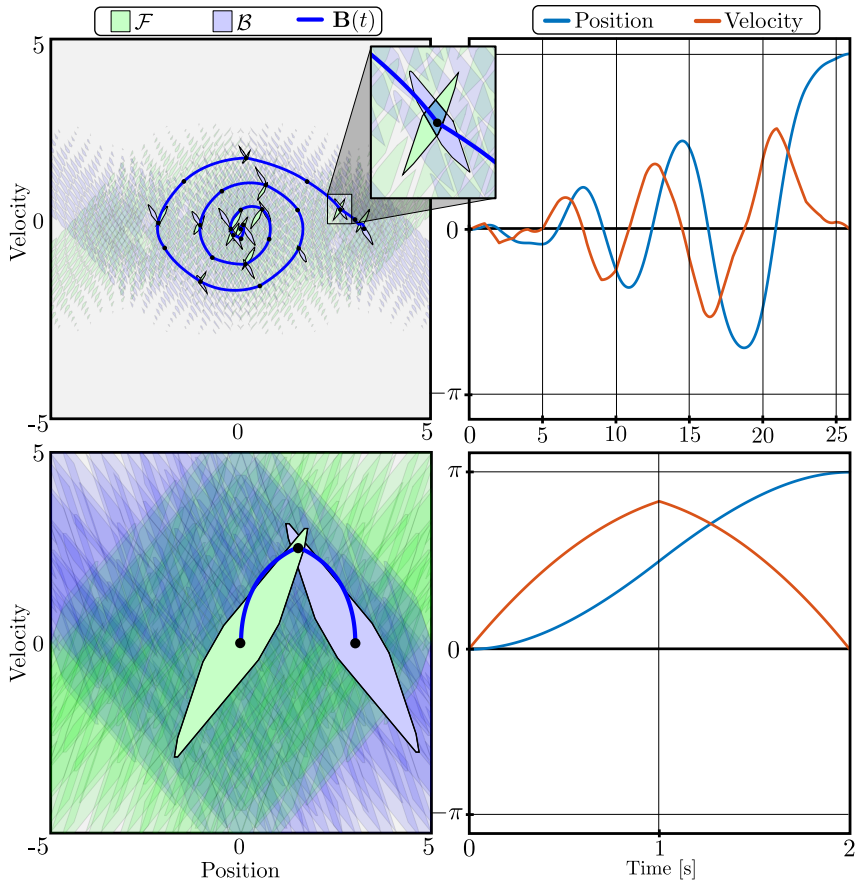


Figure 6.6: The proposed method applied to the pendulum swingup problem. As the input bounds are tightened from 5 Nm (bottom) to 0.5 Nm (top), the resulting graph search trajectory increases in complexity and length.

Simulation Results

We deploy the use of Bézier Reachable Polytopes towards the task of swinging up the pendulum. The duration of planning horizon needed to accomplish this task depends highly on how tight the input constraint for the system is. In this setup, the tracker was taken to be the feedback linearizing controller, and the planner produced trajectories on the pendulum dynamics. This planner-tracker was interfaced with a graph-search problem, which samples states uniformly from the state space and connects two vertices $\mathbf{v}_i, \mathbf{v}_j \in \mathcal{X}_d$ with an edge if the intersection of their forward and reachable sets were nonempty, i.e., $\mathcal{F}(\mathbf{v}_i) \cap \mathcal{B}(\mathbf{v}_j) \neq \emptyset$. This represents a graph of dynamically feasible Bézier curves, whereby a suitable Bézier curve between two boundary conditions can be found by solving a discrete graph search problem. As seen in Figure 6.6, when the low level input constraints are tight, the graph search has to produce a long sequence of points to achieve pendulum swingup. Instead, if the

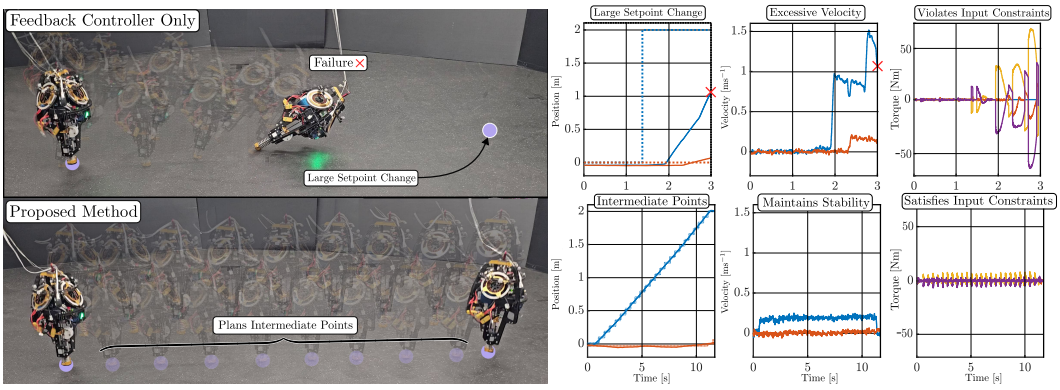


Figure 6.7: Hardware results on the 3D hopping robot, ARCHER. When commanded to cross the room, naive decision making provides a setpoint to the planner which is outside what is achievable by the low level, leading to system failure. Instead, if Bézier Reachable Polytopes are used, the decision level provides a sequence of waypoints to the planner that results in completion of the objective while satisfying state and input constraints.

input constraints are loose, then a nearly direct swingup behavior can be achieved. In this way, we observe that the computational complexity of the decision making level is imposed by the limitations of the underlying full order system. The code for this project is available at [172].

Hardware Results

We also deploy the Bézier Reachable Polytopes framework towards the control of a 3D hopping robot, ARCHER [107], as seen in Figure 6.7. Let $(\mathbf{p}, q) \in \mathbb{R}^3 \times \mathbb{S}^3$ denote the global position and quaternion of the robot, and $(\mathbf{v}, \boldsymbol{\omega}) \in \mathbb{R}^3 \times \mathfrak{s}^3$ the global linear velocity and body frame angular rates. The full state of the robot $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^{20}$ contains these values, as well as foot and flywheel positions and velocities. Planning long-horizon tasks for this robot is extremely challenging due to the large number of passive degrees of freedom, tight input constraints, and hybrid dynamics. Separating the path planning problem into a layered architecture consisting of a tracking controller, a planner, and a decision layer enables this task to be split up, whereby behavior can be generated efficiently.

In this setup, we take the planning model to be a double integrator with state $\mathbf{x}_d \in \mathcal{X}_d \triangleq \mathbb{R}^4$ and input $\mathbf{u}_d \in \mathcal{U}_d \triangleq \mathbb{R}^2$. This planning model Σ_d can be corresponded with the hopping robot Σ by a projection map $\mathbf{\Pi} : \mathcal{X} \to \mathbb{R}^4$ taken to be the restriction of the full order state to the center of mass x and y positions

and velocities. The tracking model is the linearized version of the ZDP developed in the previous section. As seen in Figure 6.7, if only the feedback layer is used, the system fails because the desired setpoint is outside the region of what can be accomplished by the tracking system. Instead, if the proposed method is used, the decision layer can autonomously produce a sequence of points which maintain stability and constraint satisfaction over the task.

6.3 Summary

In this chapter, we saw that by parameterizing the trajectory optimization problem with Bézier curves, we could make continuous time guarantees for the state and input constraints satisfaction of the continuous-time closed loop system. Furthermore, we saw how this transcription method enabled efficient representations of the reachable sets of the planner, which will be leveraged in the next section to provide and efficient oracle for determining the kinodynamic connectivity of points in the reduced order state space.

Chapter 7: High Level Planning Layer

Contents

Graphs of behaviors provide an abstraction for complex high level reasoning.

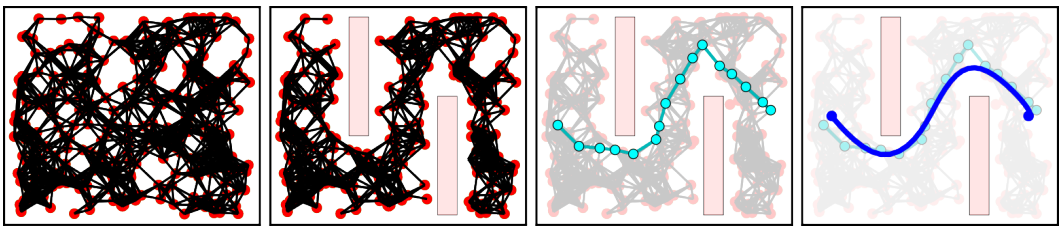


Figure 7.1: The path planning framework. From left to right: **a)** A Bézier graph is constructed, **b)** it is cut based on the present obstacles, **c)** a path is solved, and d) the path is refined with MPC.

In the previous section, we saw how to design an efficient oracle for the planning system, which, when queried gives a notion of connectivity between points in the state space. In this section, we leverage this oracle to produce graphs of kinodynamically feasible curves, and pair this with hardware accelerated compute to efficiently prune the graph and solve for paths in cluttered environments in real time.

7.1 Kinodynamic Bézier Graphs

The goal of this section will be to produce sequences of points $\{\bar{\mathbf{x}}_k\}$ and convex cell decompositions $\{\mathcal{X}_k\}$ when obstacles are present, i.e., the free space is non-convex. The approach will be to produce a graph of dynamically feasible Bézier curves leveraging the *Bézier reachable polytopes* developed in the previous section, cut the edges that intersect obstacles, and perform a graph search that will serve as a feasible warm start for the previously developed Bézier trajectory optimization program. The paradigm is outlined in Algorithm (3) and shown in Figure 7.1, and starts with building a graph of Bézier curves.

Build Graph

The compact state space of interest $\mathcal{X}_d$ represents the free space of the problem setup, and will serve as a seed which we can prune in the presence of obstacles. A graph $\mathcal{G}$ is described by a tuple (V, E) with vertices V connected by edges E. For

Algorithm 3 Bézier Graphs

1: **hyperparameters:** (N, T) ▷ (Node count and time interval)
2: $\mathcal{G} \leftarrow$ buildGraph($\mathcal{X}_d, \mathcal{U}$)
3: $\{\mathcal{X}_k\} \leftarrow$ decomposeFreeSpace($\mathcal{O}$)
4: $\mathcal{C} \leftarrow$ cutGraph($\mathcal{G}$, $\mathcal{O}$)
5: $\mathbf{v}_k^* \leftarrow$findPath($\mathcal{C}$)
6: $\mathbf{x}_d(\cdot) \leftarrow$refineWithMpc($\mathbf{v}_k^*, \mathcal{O}$)
7: **return** $\mathbf{x}_d(\cdot)$

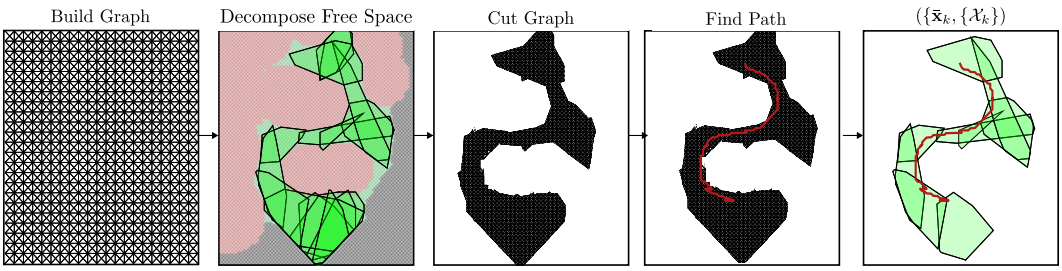


Figure 7.2: A depiction of the process of the high layer, which 1) builds a graph of Bézier polynomials, 2) decomposes the free space into convex free spaces and obstacles, 3) cuts edges based on the present obstacles, and 4) solves for a path.

the purpose of our analysis, the vertices will represent points in the reduced-order state space $\mathbb{R}^n$. A directed edge between two vertices $e = (v_1, v_2)$ for $v_1, v_2 \in V$ will represent the existence of a dynamically feasible trajectory connecting v_1 to v_2. To begin, N points are uniformly sampled $\mathbf{v}_i \sim \mathrm{U}(\mathcal{X}_d)$ and the matrices $\mathbf{F}$ and $\mathbf{G}$ from Theorem 6.2 are generated. Two points $\mathbf{v}_i$ and $\mathbf{v}_j$ are connected with an edge $\mathbf{e}_{i,j}$ if they satisfy $\mathbf{F}\begin{bmatrix}\mathbf{v}_i^\top & \mathbf{v}_j^\top\end{bmatrix}^\top \leq \mathbf{G}$. As such, every edge in the graph implies the existence of a dynamically feasible Bézier curve connecting them. This initial graph will be fixed throughout the remainder of the Algorithm.

Build Environment Representation

Given the graph $\mathcal{G}$ constructed previously, we would like the remove graph edges which lead to collisions with obstacles in the environment. This operation can be efficiently performed if we have either 1) a representation of obstacles as convex polytopes, or 2) a representation of the free space as a union of convex polytopes. The experiments later in this section leverage both of these perspectives depending on the available sensing capability.

Algorithm 4 DecomposeFreeSpace$(\mathcal{P}_f, \mathcal{P}_o)$

Input: $\mathcal{P}_f$ — set of free points; $\mathcal{P}_o$ — set of occupied points
Output: $\mathcal{X}_f$ — collection of free-space regions

$\{\mathcal{P}_f^{(i)}\}_{i=1}^k \leftarrow$ KMeansCluster$(\mathcal{P}_f, k)$
$\mathcal{X}_f \leftarrow \emptyset$
for all (i, j) such that $1 \leq i \leq j \leq k$ **do**
    $\mathbf{A}, \mathbf{b} \leftarrow$ FitEllipse$(\mathcal{P}_f^{(i)} \cup \mathcal{P}_f^{(j)})$
    $\mathcal{P}_o^{(i,j)} \leftarrow \{\mathbf{y} = \mathbf{A}(\mathbf{p} - \mathbf{b}) \mid \mathbf{p} \in \mathcal{P}_o \text{ and } \mathbf{y} \leq 1\}$
    $\mathcal{X}^{(i,j)} \leftarrow$ FitFree$(\mathcal{P}_o^{(i,j)})$
    $\mathcal{X}_f \leftarrow \mathcal{X}_f \cup \mathbf{A}^{-1}\mathcal{X}^{(i,j)} + \mathbf{b}$
end for
return $\mathcal{X}_f$

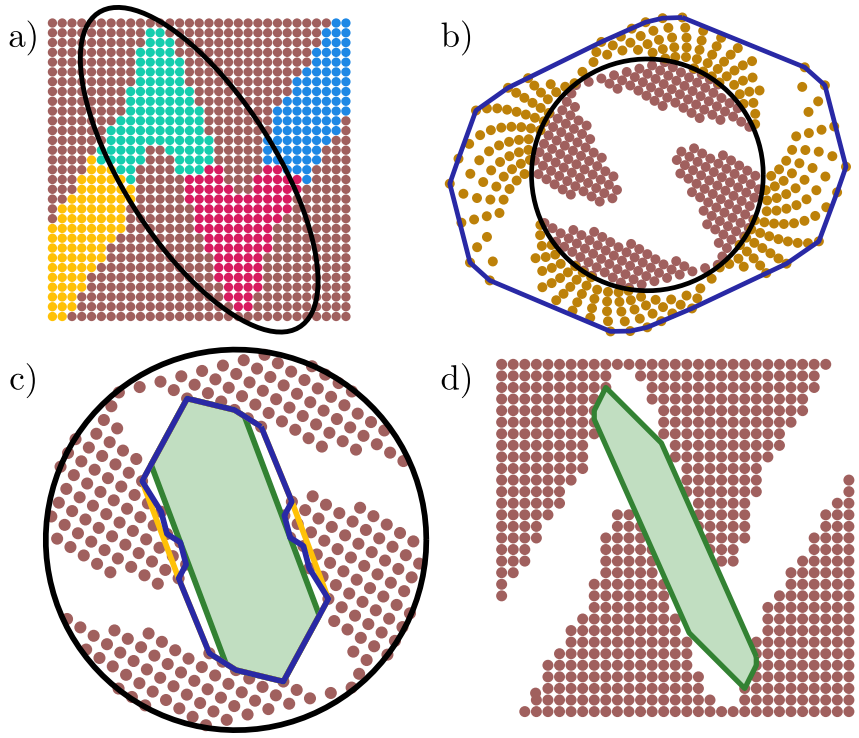


Figure 7.3: A visual depiction of the DECOMPOSEFREESPACE algorithm, which fits convex polytopes to free spaces by **a)** fitting an ellipse to clustered free points, **b)** mapping occupied points outside a circle and computing a convex hull **c)** mapping the convex hull back inside the circle, and convexifying, and **d)** mapping back to the environment.

Union of Convex Free Spaces Let $\mathcal{O} : \mathbb{Z}^{\dim(\mathcal{X}_{\mathrm{H}})} \to \{0, 1\}$ denote a 2D occupancy grid. To produce a convex cell decomposition, we implement a modified version of FITFREE, the algorithm from [195], outlined in Algorithm 4. Our modifications aim to automatically choose sets of points to run FITFREE on, avoiding tuning of the radius of points considered, retaining fast computation speeds by considering subsets of the occupied points, and encouraging connectivity of the resulting convex sets.

First, the free space of $\mathcal{O}$ is divided into k clusters using the K-means clustering algorithm [196], [197]. Then, for each cluster (when $i = j$), and each pair of clusters (when $i < j$), FITELLIPSE outer-approximates the free points with an ellipse using three standard deviations of the variance matrix, and picks the center point to be the free space point closest to the mean. Critically, evaluating pairs of clusters encourages connectivity of the resulting convex polytope cover of the free space. Next, all of the occupied points inside the ellipse are transformed into the unit circle,

and FITFREE is run on the transformed points, exactly as in [195]. Finally, the convex free polytope is transformed back into the original space. With this, Algorithm 4 produces a convex cell decomposition of the free space, i.e., $\cup \mathcal{X}_k \subset \mathcal{X}$.

Union of Convex Obstacles Alternatively, a collection of convex obstacle representations are directly fit to the occupied space of $\mathcal{O}$. In this regime, a set of convex obstacles $\mathcal{O}_k$ is returned instead for an *a priori* specified hyperparameter of number of obstacles k. This is done by fitting a rectangular (but not axis aligned) bounding box to an image mask of the obstacle using the OpenCV library in Python. Either representation is conducive to efficient graph cutting operations, as discussed in the next section.

Cut Graph

Given a graph of Bézier curves $\mathcal{G}$, we remove the edges that could result in collisions with any obstacles that are present.

Union of Convex Free Spaces In this case, the cutting of the graph is straightforward: any Bézier curves whose control points do not lie in any convex cell are cut. As with the following discussion, this operation can easily be parallelized on GPU, as it simply requires simultaneous linear inequality checks.

Union of Convex Obstacles Given a set of obstacles $\mathcal{O}$, Property 2.43 states that a Bézier curve is guaranteed to be collision-free if the convex hull of its control points does not intersect with any obstacles, i.e., $\text{conv}\{\mathbf{P}\} \cap \mathcal{O} = \emptyset$. For each edge $\mathbf{e}$ with control points $\mathbf{P}$ and convex obstacle $\mathcal{O}_i$ characterized by n_c hyperplane constraints $\mathbf{A}^{\mathcal{O}_i} \in \mathbb{R}^{n \times n_c}$ and $\mathbf{b}^{\mathcal{O}_i} \in \mathbb{R}^{n_c}$, the problem of determining if the curve is collision-free can be formulated as a linear program feasibility check via Property (2.43), which we solve using the following quadratic program:

$$\begin{aligned} \min_{\boldsymbol{\lambda}, \boldsymbol{\delta}} \quad & \boldsymbol{\delta}^\top \boldsymbol{\delta} \\ \text{s.t.} \quad & \mathbf{A}^{\mathcal{O}_i} \mathbf{P} \boldsymbol{\lambda} \leq \mathbf{b}^{\mathcal{O}_i} + \boldsymbol{\delta} \\ & \boldsymbol{\lambda}_j \geq 0, \quad \sum_j \boldsymbol{\lambda}_j = 1 \end{aligned} \tag{Cut-QP}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{p+1}$ is the convex interpolation variable and $\boldsymbol{\delta} \in \mathbb{R}^{n_c}$ is a slack variable. Given the solution $\boldsymbol{\delta}^*$, if $\|\boldsymbol{\delta}^*\|_2 > 0$ then the curve is obstacle free. This program can easily be vectorized to solve all edges simultaneously.

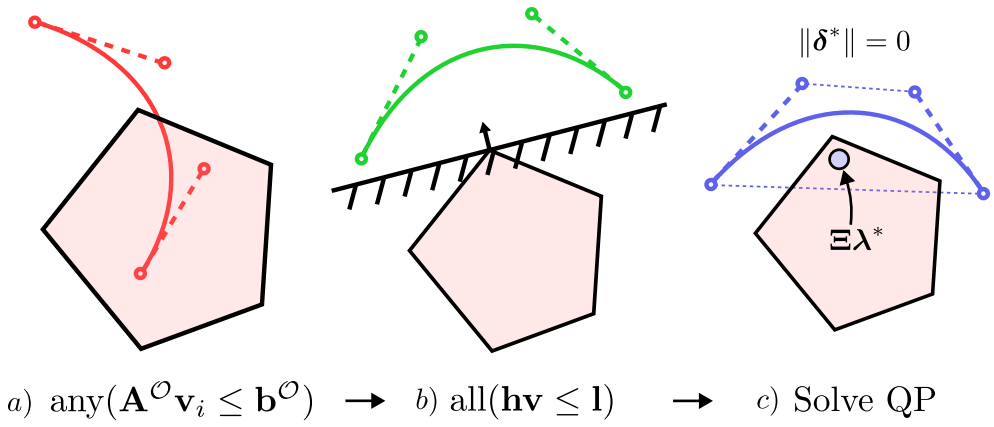


Figure 7.4: The heuristic employed to check if the Bézier curve is collision free. First, obstacle membership is checked. If not satisfied, a single proposed separating hyperplane is checked. Finally, the Quadratic Program (Cut-QP) is solved.

Although this is a quadratic program and therefore can be solved efficiently, solving it for every edge and obstacle becomes too computationally expensive for real-time operation (even when parallelized), especially when the edge and and number of obstacle count is large. To address this, we introduce the heuristic outlined in Algorithm 5, which consists of three checks. First, we determine if any of the control points of $\mathbf{P}$ lie in the obstacle — if so, the curve may intersect the obstacle. Next, we attempt to find a separating hyperplane between the obstacle and the control points — if one exists, then the curve will be collision free. If the points do not lie on one side of the hyperplane, then the heuristic is indeterminate, and the original quadratic program must be solved. In practice, this heuristic typically removes > 99% of the edges that need to be checked; furthermore, this heuristic is easily implemented on GPU as it only requires linear algebra operations, which significantly improves execution speed (as reported in Section 7.2).

Algorithm 5 cutHeuristic

1: **if** any$(\mathbf{A}\mathbf{v}_i \leq \mathbf{b})$ **then** ▷ Figure 7.4a)
2: return 0
3: **end if**
4: $\mathbf{v}^o \leftarrow$closestPoint$(\mathbf{v}_i)$
5: $(\mathbf{h}, \mathbf{l}) \leftarrow$adjacentHyperplane$(\mathbf{v}^o)$
6: **if** all$(\mathbf{h}\mathbf{v}_i \leq \mathbf{l})$ **then** ▷ Figure 7.4b)
7: return 1
8: **end if**
9: Solve QP ▷ Figure 7.4c)
10: return $\|\boldsymbol{\delta}^*\| > 0$

Find Path

Due to the random sampling of the points in the graph, the starting and goal positions are likely not vertices. Therefore, the start and goal nodes are chosen as the closest normed-distance points to the desired start and goal positions. The edge cost is taken to be $\sum_i \|\mathbf{P}_{i+1} - \mathbf{P}_i\|$, the upper bound on the path length of the Bézier curve from Property 2.49. We then use Dijkstra's algorithm to solve Problem 7.1:

Problem 7.1. Consider a collision-free graph of Bézier curves $\mathcal{C} = (V, E)$ produced by Algorithm 3. Find a path $\{\mathbf{v}_k^*\}_{k=0}^K$ in the graph connecting $\mathbf{v}_0^*$ to $\mathbf{v}_K^* = \mathbf{\Pi}(x_G)$ with $\mathbf{v}_0^*$ satisfying $\Pi(\mathbf{x}(0)) \in \mathbf{v}_0^* \oplus \bar{\mathcal{E}}$.

Refine the Path

If the graph solve $\mathbf{v}_k^*$ exists, it represents a feasible path for the system to safely traverse the cluttered environment; however, it may be significantly suboptimal. To improve its optimality, we can solve a finite time optimal control program, which balances tracking the graph solution with short-horizon optimality. Note that by Property 2.45, a solution Problem 7.1 can be converted into a dynamically feasible curve $\mathbf{x}_d(\cdot)$ for the reduced order model (6.1). Then, we sample this curve at the optimal control time discretization to produce a sequence of reference points $\mathbf{r}_k$, which can be optimized:

$$\min_{\mathbf{x},\mathbf{u}} \quad \sum_{k=0}^{N-1} \mathbf{x}_k^\top \mathbf{Q}\mathbf{x}_k + \mathbf{F}\mathbf{r}_k + \mathbf{u}_k^\top \mathbf{R}\mathbf{u}_k + \mathbf{x}_N^\top \mathbf{V}\mathbf{x}_N \tag{MPC}$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{7.2a}$$

$$\mathbf{x}_0 \in \mathbf{\Pi}(\mathbf{x}(0)) \oplus \bar{\mathcal{E}} \tag{7.2b}$$

$$\mathbf{D}\begin{bmatrix}\mathbf{x}_k & \mathbf{x}_{k+1}\end{bmatrix} \in \mathcal{X}_d \backslash (\mathcal{O} \oplus \mathcal{E}) \tag{7.2c}$$

$$\mathbf{F}\begin{bmatrix}\mathbf{x}_k & \mathbf{x}_{k+1}\end{bmatrix} \leq \mathbf{G} \tag{7.2d}$$

$$\mathbf{x}_N = \mathbf{r}_N \tag{7.2e}$$

where $\mathbf{Q}, \mathbf{F} \in \mathbb{R}^{n\times n}$ are symmetric positive definite matrices weighting distance to reference as well as path length, $\mathbf{R} \in \mathbb{R}^{m\times m}$ is a positive definite input scaling matrix, and $\mathbf{V} \in \mathbb{R}^{n\times n}$ is a terminal cost. The matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times m}$ are the exact discretization of the integrator dynamics in (6.1). The constraint $\mathcal{X}_d \backslash (\mathcal{O} \oplus \mathcal{E})$ is enforced by fining a separating hyperplane between the node $\mathbf{x}_k$ and the closest obstacle (where the same adjacentHyperplane call form the heuristic can be made). In order to improve the quality of solutions, (MPC) can be iteratively solved in an SQP fashion.

Importantly, the graph solve represents a feasible warm start for the MPC program, and can be thought of as providing a corridor and a dynamically feasible path which MPC can refine. The main motivation for the structures used in this paper is the notion of guaranteed feasibility:

Theorem 7.2. *If the graph solve is feasible, then the high level problem is solved by applying* (MPC) *in closed loop.*

Proof. First, we must show that at time $t = 0$, the solution $\mathbf{v}_k^*$ from Problem 7.1 provides a feasible solution for (MPC). Let $\mathbf{r}_k$ denote the refined sequence of $\mathbf{v}_k^*$, which is constructed to produce the reference for (MPC) using Property 2.47. As this is associated with a Bézier curve for the reduced-order model, it satisfies the discrete dynamics (7.2a). Next, by the definition of Problem 7.1, we know $\Pi(\mathbf{x}_0) \in \mathbf{v}_0^* \oplus \bar{\mathcal{E}}$, which implies that (7.2b) is satisfied by Mikoswki addition properties. As $\mathbf{v}_k^*$ is in the collision-free graph $\mathcal{C}$, it satisfies (7.2d) and (7.2c). By construction, (7.2e) is satisfied. Therefore, $\mathbf{r}_k^*$ represents a feasible solution for (MPC).

Next, take $\mathbf{x}_k^*$ to denote the MPC solution and consider the time interval $I = \begin{bmatrix} 0 & h \end{bmatrix}$ for MPC discretization $h > 0$. From Lemma (2.45), we know that there exists a unique curve $\mathbf{x}_d(\cdot)$ defined over I connecting $\mathbf{x}_0^*$ to $\mathbf{x}_1^*$. By Theorem 6.2, the closed loop system tracking this curve will satisfy $\mathcal{C}_\mathcal{X}$ and $\mathcal{C}_\mathcal{U}$ for all $t \in I$. Furthermore, by virtue of the low level controller, we have that $\mathbf{\Pi}(\mathbf{x}(h)) \in \mathbf{x}_1^* \oplus \bar{\mathcal{E}}$. Therefore, we can appeal to standard Robust tube MPC theory [75] to claim recursive feasibility and robust stability of the closed loop system tracking the MPC solution, meaning the goal state is achieved and $\mathcal{C}_\mathcal{X}$ and $\mathcal{C}_\mathcal{U}$ are satisfied for all time. □

We have shown that if there is a solution to the graph problem in Algorithm 3, then the closed loop system tracking MPC will satisfy state and input constraints $\mathcal{X}_d \backslash \mathcal{O}$ and $\mathcal{U}$ for all time, and will approach a neighborhood of the goal $\mathbf{x}_G$.

7.2 Nonconvex Path Planning in Real Time

We deploy Algorithm 3 on the 3D hopping robot, ARCHER [107] — a video can be found at [198] and the code for this project is available at [172]. The most challenging problem preventing the nonconvex problem from being directly solved in real time is appropriately choosing a collection of corridors to traverse. By combining problems at various time scales, the graph solve is able to provide a coarse estimate of the keep in constraints while foregoing path optimality, and the short horizon MPC is able to refine it into a more optimal path. Therefore, we

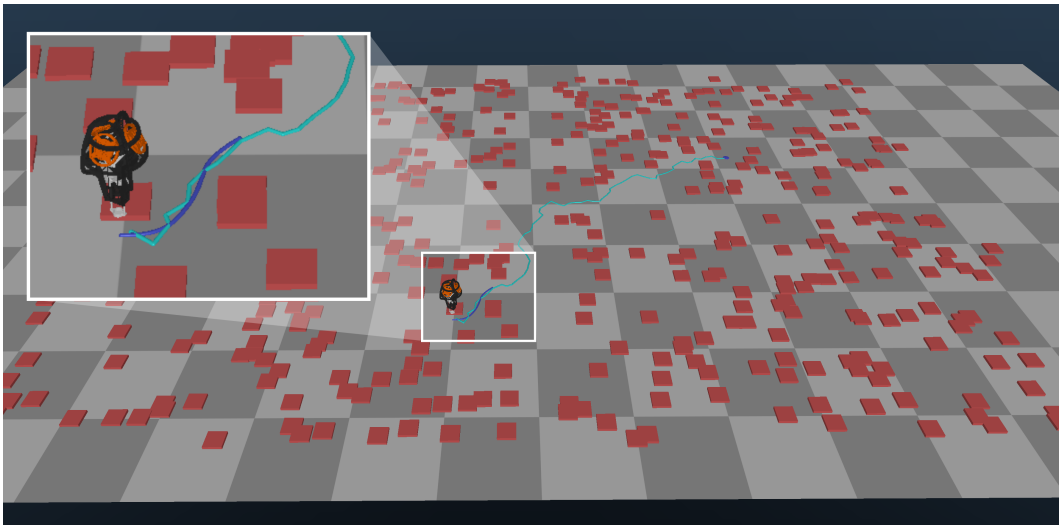


Figure 7.5: 500 randomly generated obstacles with graph replanning at 10 Hz and (MPC) at 200 Hz. Cyan indicates the Bézier graph solve $\mathbf{v}_k^*$ and blue the MPC Bézier solution $\mathbf{x}_d(\cdot)$.

choose to plan desired center of mass trajectories $\mathbf{x}_d \in \mathbb{R}^4$ with virtual force inputs $\mathbf{u}_d \in \mathbb{R}^2$, which are assumed to have double-integrator dynamics.

In order to solve (Cut-QP) and the MPC program, the OSQP library [199] was used with maximum iterations limited to 50. The heuristic for cutting the graph and getting the separating hyperplanes was running on a custom written CUDA kernel. Using the GPU to perform these operations was critical to real-time performance — as summarized in Table 7.1, the CUDA kernel provided a significant speedup as compared to multithreaded CPU implementation. A nominal graph with 0.5 second Bézier curves and a 50 node MPC horizon with a timestep of 0.1 s were used. Each iteration of the MPC program was solved at 100 Hz, and in practice 1 SQP iteration was taken. Finally, path length cost was scaled significantly higher than tracking cost in order to incentivize shorter paths than the graph solve could provide.

Table 7.1: Comparison of functions on CPU, parallelized CPU, and GPU. Any CPU implementation would violate real time constraints.

	cutHeuristic	adjacentHyperplane
CPU	5920 ± 36 ms	6.24 ± 0.08 ms
Parallel CPU	3640 ± 31 ms	2.52 ± 0.33 ms
GPU	72.0 ± 1.5 ms	0.61 ± 0.14 ms

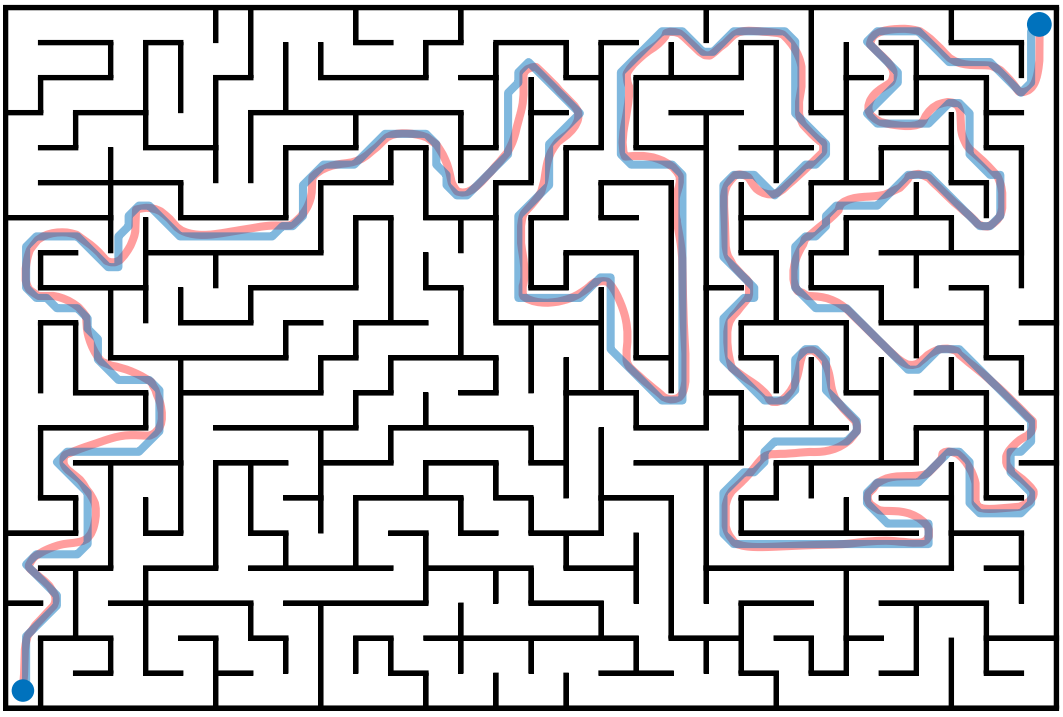


Figure 7.6: A long-horizon maze with 600 cells, 300 obstacles and 50,000 edges. The graph cut is solved at 10 Hz, and graph solve and MPC both run at 200 Hz. Blue represents the graph solve, and red the closed loop system behavior.

Simulation and Hardware

As can be seen in Figure 7.5 and Figure 7.6, the proposed framework is able to solve extremely long horizon tasks with numerous obstacles in real time. Figure 7.5 shows how the MPC solution $\mathbf{x}_d(\cdot)$ can refine the graph solve $\mathbf{v}_k^*$ to improve the optimality of the path while maintaining collision avoidance. In Figure 7.6, a grid was used instead of random sampling due to the structured nature of the problem. In both of these settings, about 5000 nodes were sampled for the graph, leading to 50,000 edges to have to be processed every graph solve time step. In this setup, the (Cut-QP) takes 2.7 seconds to run per obstacle, even when limited to 50 solver iterations. When the heuristic runs, it is able to remove > 99% of the edges and leads to solve times of 10-20 ms.

For the hardware setup seen in Figure 1.3, in order to estimate obstacle locations, we had an overhead ZED 2 camera and used the SAM2 segmenting repository [200]. Once the experiment started, the segmenter was initialized with a single click per obstacle, and was able to parse obstacle locations for the remainder of the experiment, which were streamed over ROS to the hierarchical controller. SAM2

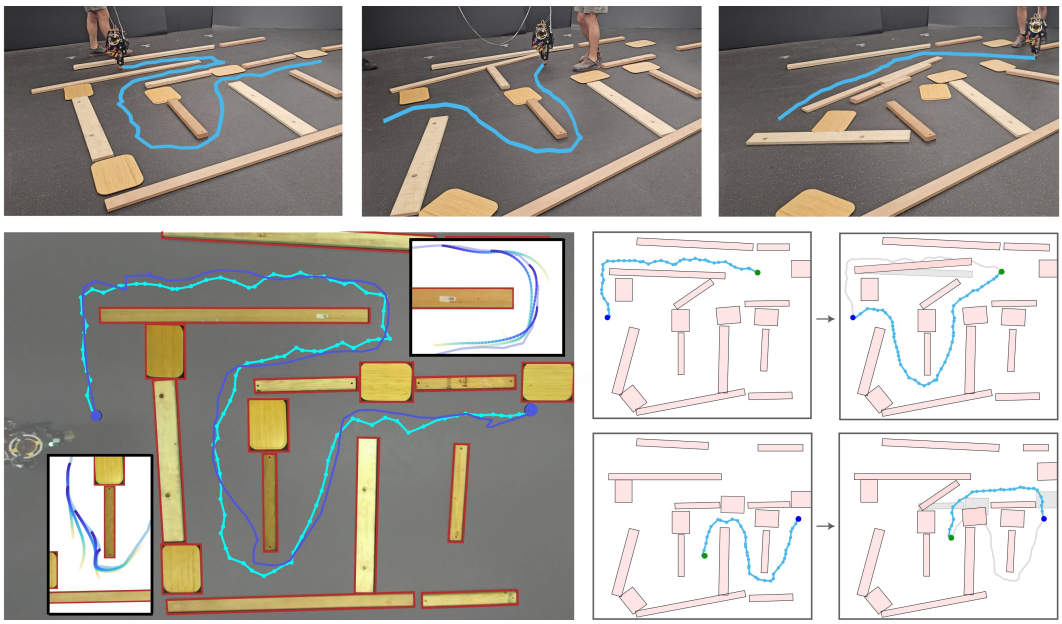


Figure 7.7: Experiments run on the ARCHER hardware Platform. **(Top)** 3 snapshots of the graph solve for various obstacle configurations. **(Left)** An overhead image of the graph solve and the closed loop system trajectory. In the upper right and bottom left corner, the MPC refinement around the corner can be seen. **(Right)** Freeze frames of the graph updating online when the obstacle locations are moved.

segmented 20 obstacles at 4 Hz, and the graph cut and MPC ran at 50 Hz and 200 Hz, respectively. We provided various cluttered environments to ARCHER, which was able to solve for feasible paths and traverse them in real time. In the bottom right of Figure 7.7, the real-time replanning of the graph can be seen as the obstacles were kicked around in the environment.

7.3 Summary

In this chapter, we saw how abstractions of the mid and low level controllers in the form of reachable sets could be used to efficiently produce graphs of kinodynamically feasible curves. Then, by leveraging the structure of compute to efficiently prune and solve for paths in this graph, the long-horizon planning problem could efficiently be deployed in real time, enabling significant speedup when compared to a CPU implementation. These solutions provide initial guesses for gradient based optimizers, and successfully enabled real-time nonconvex path planning in cluttered environments.

Chapter 8: The Complete Architecture

Contents

General autonomy will, in some way, shape, or form, leverage layered control architectures.

8.1 Theory

We begin this section by recalling the problem we set out to solve:

Problem. Given an initial state $\mathbf{x}_0 \in \mathcal{X}$, a goal state $\mathbf{x}_G \in \mathcal{X}$, constraint sets $\mathcal{X} \subset \mathbb{R}^n$ and $\mathcal{U} \subset \mathbb{R}^m$, a horizon $T > 0$, and a tolerance $\varepsilon > 0$, design a controller $\mathbf{k} : \mathbb{R}^n \to \mathbb{R}^m$ such that the closed-loop trajectory $\mathbf{x}_{cl}$ of system (3.1) satisfies:

- $\|\mathbf{x}_{cl}(T) - \mathbf{x}_G\| \leq \varepsilon$
- $\mathbf{x}_{cl}(t) \in \mathcal{X} \subset \mathbb{R}^n$ for all $t \in [0, T]$
- $\mathbf{k}(\mathbf{x}_{cl}(t)) \in \mathcal{U} \subset \mathbb{R}^m$ for all $t \in [0, T]$.

At the outset, we developed the specifications of each component of the hierarchy in Section 3.6. From Theorem 3.8, we know that if we can produce components that meets these specifications, then we can solve the main problem of interest:

Low Level Tracking Layer $\mathcal{L}_{\text{Low}}^{\text{Tracking}}$

- $\texttt{In}: \iota_{\text{L}} = (\mathbf{x}_d, \mathbf{u}_d) \in C^1(\mathbb{R}_+, \mathcal{X}_{\text{L}}) \times C^1(\mathbb{R}_+, \mathcal{U}_{\text{L}})$
- $\texttt{Out}$: $y_{\text{L}} = (\mathbf{x}, \mathbf{u}) \in C^1(\mathbb{R}_+, \mathcal{X}_{\text{L}}) \times C^1(\mathbb{R}_+, \mathcal{U}_{\text{L}})$
- $\mathcal{G}_{\text{L},1} : \iota_{\text{L}} \mapsto \{y_{\text{L}} \mid \mathbf{x} \in \mathbf{x}_d \oplus \mathcal{E}_{\text{L}}\}$
- $\mathcal{G}_{\text{L},2} : \iota_{\text{L}} \mapsto \{y_{\text{L}} \mid \mathbf{u} \in \mathcal{U}_{\text{L}}\}$,

for $\mathcal{E}_{\text{L}} \subset \mathcal{X}_L$ satisfying $\mathcal{E}_{\text{L}} \subset B_\varepsilon(\mathbf{0})$ for the desired tolerance $\varepsilon > 0$ defined in Problem 3.1.

Low Level Planning Layer $\mathcal{L}_{\text{Low}}^{\text{Planning}}$

- $\texttt{In}: \iota_{\text{L}} = (\bar{\mathbf{x}}_d, \bar{\mathbf{u}}_d) \in C^1(\mathbb{R}_+, \mathcal{X}_{\text{M}}) \times C^1(\mathbb{R}_+, \mathcal{U}_{\text{M}})$
- $\texttt{Out}$: $y_{\text{L}} = (\mathbf{x}_d, \mathbf{u}_d) \in C^1(\mathbb{R}_+, \mathcal{X}_{\text{L}}) \times C^1(\mathbb{R}_+, \mathcal{U}_{\text{L}})$
- $\mathcal{G}_{\text{L},1} : \iota_{\text{L}} \mapsto \{y_{\text{L}} \mid \mathbf{\Pi}_{\text{M, L}}(\mathbf{x}_d) \in \bar{\mathbf{x}}_d \oplus \mathcal{E}_{\text{M}}\}$
- $\mathcal{G}_{\text{L},2} : \iota_{\text{L}} \mapsto \{y_{\text{L}} \mid \mathbf{u}_d \in \mathcal{U}_{\text{L}}\}$,

for $\mathcal{E}_{\text{M}} \subset \mathcal{X}_M$ satisfying $\mathbf{\Pi}_{\text{H, M}}(\mathcal{E}_{\text{M}}) \subset B_\varepsilon(\mathbf{0})$.

High Level Tracking Layer $\mathcal{L}^{\text{Tracking}}_{\text{High}}$

- $\texttt{In}: \iota_{\text{M}} = (\{\bar{\mathbf{x}}_k\}, \{\mathcal{X}_k\}) \in \mathcal{X}_{\text{H}} \times \mathcal{P}(\mathcal{X}_{\text{H}})$
- $\texttt{Out}$: $y_{\text{M}} = (\bar{\mathbf{x}}_d, \bar{\mathbf{u}}_d) \in C^1(\mathbb{R}_+, \mathcal{X}_{\text{M}}) \times C^1(\mathbb{R}_+, \mathcal{U}_{\text{M}})$
- $\mathcal{G}_{\text{M},1} : \iota_{\text{M}} \mapsto \{y_{\text{M}} \,|\, \mathbf{\Pi}_{\text{H, M}}(\bar{\mathbf{x}}_d(t)) \in \mathcal{X}_k \ominus \mathbf{\Pi}_{\text{H, M}}(\mathcal{E}_L) \forall t \in [kT/N, (k{+}1)T/N)\}$
- $\mathcal{G}_{\text{M},2} : \iota_{\text{M}} \mapsto \{y_{\text{M}} \mid \mathbf{\Pi}_{\text{H, M}}\left(\bar{\mathbf{x}}_d\left(\frac{kT}{N}\right)\right) \in \bar{\mathbf{x}}_k \oplus \mathcal{E}_{\text{M}}\}$.

High Level Planning Layer $\mathcal{L}^{\text{Planning}}_{\text{High}}$

- $\texttt{In}$: $\iota_{\text{H}} = (\mathbf{x}_0, \mathbf{x}_G, \mathcal{X}) \in \mathcal{X}_{\text{H}} \times \mathcal{X}_{\text{H}} \times \mathcal{P}(\mathcal{X}_{\text{H}})$
- $\texttt{Out}$: $y_{\text{H}} = (\{\bar{\mathbf{x}}_k\}, \{\mathcal{X}_k\}) \in \mathcal{X}_{\text{H}} \times \mathcal{P}(\mathcal{X}_{\text{H}})$
- $\mathcal{G}_{\text{H},1} : \iota_{\text{H}} \mapsto \{y_{\text{H}} \mid \bar{\mathbf{x}}_0 = \mathbf{x}_0 \text{ and } \bar{\mathbf{x}}_N = \mathbf{x}_G \}$
- $\mathcal{G}_{\text{H},2} : \iota_{\text{H}} \mapsto \{y_{\text{H}} \mid \mathcal{X}_k \subseteq \mathcal{X} \text{ for } k = 0, \dots, N\}$
- $\mathcal{G}_{\text{H},3} : \iota_{\text{H}} \mapsto \{y_{\text{H}} \mid \bar{\mathbf{x}}_k \in \mathcal{X}_k \text{ for } k = 0, \dots, N\}$,

where $\mathcal{X} \subseteq \mathcal{X}_{\text{H}}$ is the free space.

Given the methods constructed in this thesis, we have the necessary tools to prove that the instantiated components satisfy these requirements:

Low Level

Theorem 8.1. *There exists $\mathcal{A}_{\text{L}} \subset \mathcal{X}_{\text{M}} \times \mathcal{U}_{\text{M}}$ and $\mathcal{E} \subseteq \mathcal{X}_{\text{M}}$ such that if $\mathbf{\Pi}_{\text{M},L}(\mathbf{x}_0) \in \mathbf{x}_d(0) \oplus \mathcal{E}$, then for any reference trajectory $\mathbf{x}_d : \mathbb{R}_{\geq 0} \to \mathcal{X}_{\text{M}}$ and corresponding input trajectory $\mathbf{u}_d : \mathbb{R}_{\geq 0} \to \mathcal{U}_{\text{M}}$ satisfying $(\mathbf{x}_d(t), \mathbf{u}_d(t)) \in \mathcal{A}_{\text{L}}$ for all $t \geq 0$, the tracking error satisfies $\mathbf{\Pi}_{\text{M},L}(\mathbf{x}) \in \mathbf{x}_d \oplus \mathcal{E}$ and the input satisfies $\mathbf{u}(t) \in \mathcal{U}_{\text{L}}$ for all $t \geq 0$.*

Proof. We leverage the Zero Dynamics Policies construction from Section 5.5 to show this. From the training of the policy $\boldsymbol{\psi}$ with respect to the loss function defined in (5.46), we know that the resulting zeroing manifold $\mathcal{M}_{\boldsymbol{\psi}}$ is controlled invariant, and has exponentially stable zero dynamics. This fact combined with Corollary 5.8 implies that the zeroing manifold $\mathcal{M}^k_{\boldsymbol{\psi}}$ can be rendered controlled invariant, and the output $\boldsymbol{\eta}_k - \boldsymbol{\psi}(\mathbf{z}_k - \bar{\mathbf{z}}_k)$ can be rendered exponentially stable. Combining

Lemma 5.10 and Lemma 5.11 results in the composite discrete-time system being input to state stable. Denote the set rendered invariant around the desired trajectory to be $\mathcal{E}_d(\mathbf{x}_d, \mathbf{u}_d)$. To demonstrate the satisfaction of the input bounds, note that since the origin is an unactuated equilibrium, there exists open sets around the origin which use arbitrarily small input. With these, we can define the assumption set which satisfies state and input constraints as $\mathcal{A}_{\mathrm{L}} = \{(\mathbf{x}_d, \mathbf{u}_d) \mid \mathbf{\Pi}_{\mathrm{M},L}(\mathcal{E}_d(\mathbf{x}_d, \mathbf{u}_d)) \subseteq \mathcal{E} \text{ and } \mathbf{k}((\mathbf{0}, \bar{\mathbf{z}}_k) \oplus \mathcal{E}_d(\mathbf{x}_d, \mathbf{u}_d)) \subseteq \mathcal{U}_{\mathrm{L}}\}$. □

A depiction of the empirical existence of this tracking invariant is shown in Figure 5.23.

High Level

Theorem 8.2. *Given $T > 0$ and a tracking invariant $\mathcal{E} \subseteq \mathcal{X}_{\mathrm{M}}$, there exists a set $\mathcal{A}_{\mathrm{M}} \subset \mathcal{X}_{\mathrm{H}}^N \times \mathcal{P}(\mathcal{X}_{\mathrm{H}})^N$ such that all sequences of reference states $\bar{\mathbf{x}}_k \in \mathcal{X}_H$ and state constraint sets $\mathcal{X}_k \subset \mathcal{X}_{\mathrm{H}}$ satisfying $(\bar{\mathbf{x}}_k, \mathcal{X}_k) \in \mathcal{A}_{\mathrm{M}}$ for all $k = 1, \ldots, N$ results in the existence of a curve $\mathbf{x}_d : [0, T] \to \mathcal{X}_{\mathrm{M}}$ satisfying:*

$$\mathbf{\Pi}_{\mathrm{H,\,M}}(\mathbf{x}_d(t)) \in \mathcal{X}_k \ominus \mathbf{\Pi}_{\mathrm{H,\,M}}(\mathcal{E}_{\mathrm{L}}) \text{ for } t \in [kT/N, (k+1)T/N)$$
$$\mathbf{\Pi}_{\mathrm{H,\,M}}\left(\mathbf{x}_d\left(kT/N\right)\right) \in \bar{\mathbf{x}}_k \oplus \mathcal{E}_{\mathrm{M}}$$

for some $\mathcal{E}_M \subseteq \mathcal{X}_{\mathrm{H}}$.

Proof. We leverage the Bézier reachable polytopes construction from Section 6.2 to prove this. From the construction of $\mathcal{C}_{\mathcal{X}}$ and $\mathcal{C}_{\mathcal{U}}$, the solution curves satisfy $(\mathbf{x}_d, \mathbf{u}_d) \in \mathcal{A}_{\mathrm{L}}$. From (6.6) — enforced in (7.2d) — we have $\mathbf{\Pi}_{\mathrm{H,\,M}}(\mathbf{x}_d(t)) \in \mathcal{X}_k \ominus \mathbf{\Pi}_{\mathrm{H,\,M}}(\mathcal{E}_{\mathrm{L}})$ for $t \in [kT/N, (k+1)T/N)$. Letting $\mathcal{E}_{\mathrm{M}} = B_\epsilon(0)$, the constraint (6.11) enforces $\mathbf{\Pi}_{\mathrm{H,\,M}}\left(\mathbf{x}_d\left(kT/N\right)\right) \in \bar{\mathbf{x}}_k \oplus \mathcal{E}_{\mathrm{M}}$. Finally, we will produce $\mathcal{A}_{\mathrm{M}}$ as:

$$\begin{aligned}\mathcal{A}_{\mathrm{M}} = \{(\{\bar{\mathbf{x}}_k\}, \{\mathcal{X}_k\}) \in \mathcal{X}_{\mathrm{H}}^N \times \mathcal{P}(\mathcal{X}_{\mathrm{H}})^N \mid \\ \vec{\mathbf{L}}(\mathcal{X}_k)\vec{\mathbf{D}}^\dagger \begin{bmatrix}\mathbf{x}_k^\top & \mathbf{x}_{k+1}^\top\end{bmatrix}^\top \leq \vec{\mathbf{h}}(\mathcal{X}_k) \text{ for } k = 1, ..., N\},\end{aligned} \tag{8.1}$$

which reformulates the constraint (7.2d) into a linear constraint characterizing the sets of admissible reference points $\bar{\mathbf{x}}_k$. □

These plans for the planning model of a 2D double integrator (as used on the hopping robot), as well as a representation of the set $\mathcal{E}_{\mathrm{M}}$ is shown in Figure 6.4.

Figure 8.1: A snapshot of the experiments conducted with ARCHER, including path planning, disturbance rejection, and hopping over rough terrain.

Theorem 8.3. *Given a goal state* $\mathbf{x}_G \in \mathcal{X}_\text{H}$*, a constraint set* $\mathcal{X} \subset \mathcal{X}_\text{H}$ *and an initial condition* $\mathbf{x}_0 \in \mathcal{X}_\text{H}$*, if the high level controller is feasible at* $t = 0$ *then there exists a sequence of points* $\{\bar{\mathbf{x}}_k\}$ *with* $\bar{\mathbf{x}}_k \in \mathcal{X}_\text{H}$ *and convex sets* $\{\mathcal{X}_k\}$ *with* $\mathcal{X}_k \subseteq \mathcal{X}_\text{H}$ *which satisfy* $\bar{\mathbf{x}}_0 = \mathbf{x}_0$*,* $\bar{\mathbf{x}}_N = \mathbf{x}_G$*,* $\mathcal{X}_k \subseteq \mathcal{X}$*,* $\bar{\mathbf{x}}_k \in \mathcal{X}_k$*, and* $(\{\bar{\mathbf{x}}_k\}, \{\mathcal{X}_k\}) \in \mathcal{A}_\text{M}$*.*

Proof. We leverage the graph solve method developed in Section 7.1 to show this. First, observe that if the graph solve in the previous section is feasible, it will return a sequence of points $\{\bar{\mathbf{x}}_k\}_{k=1}^N$ with $\bar{\mathbf{x}}_0 = \mathbf{x}_0$ to $\bar{\mathbf{x}}_N = \mathbf{x}_G$ which satisfy $\mathbf{x}_k \in \mathcal{X}_k$ for $k = 1, \ldots, N$. From the construction of $\mathcal{X}_k$, we know that $\mathcal{X}_k \subseteq \mathcal{X}$. Finally, from the construction and cut of the graph, we have that $(\bar{\mathbf{x}}_k, \bar{\mathbf{x}}_{k+1}, \mathcal{X}_k) \in \mathcal{A}_\text{M}$ for all k. □

An example of the graph solve in practice is seen in Figure 7.7. What good, though, is a proof of being able to complete a task without actually doing so? Next, we

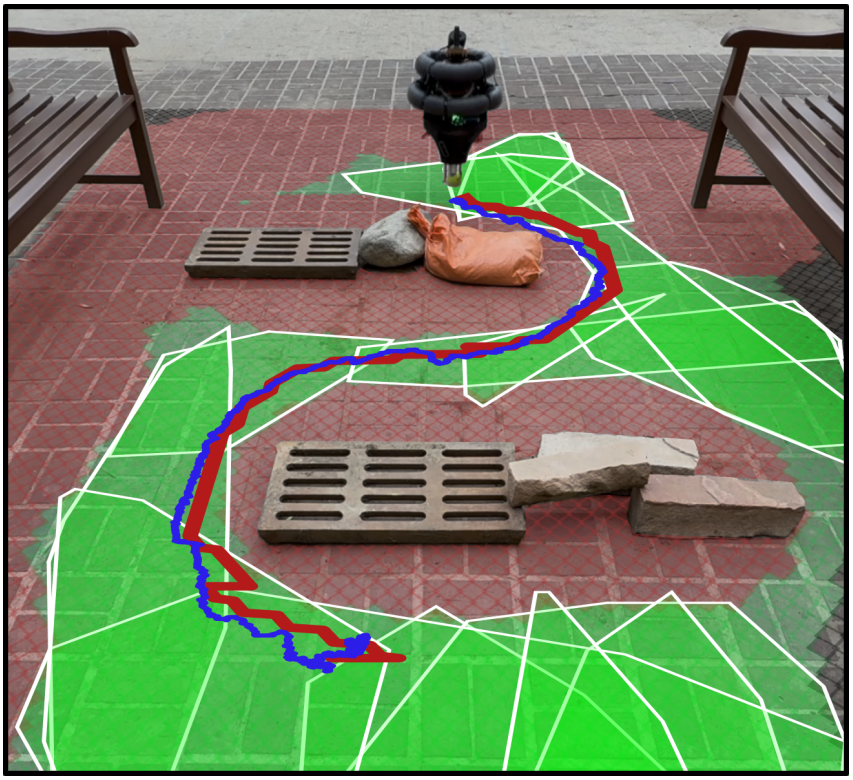


Figure 8.2: Vision-based nonconvex path planning showing the segmented occupied space (red), segmented free space (green), convex polytope decomposition (green polytopes) graph plan (red) and ARCHER trajectory (blue).

demonstrate the complete hierarchy in a number of outdoor experiments on the 3D hopping robot ARCHER.

8.2 Experiment

We utilize the complete hierarchy to achieving long horizon nonconvex path planning. We elect to model the system as a graph over 2D position states. We create the graph by sampling points in a grid and attempting to connect them with Bézier curves of duration 0.5 s. Points are considered connected with an edge if they satisfy the assumptions of the mid layer controller, i.e., $((\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_j), \mathcal{X}_{\mathrm{H}}) \in \mathcal{A}_M$ for all pairs $\mathbf{x}_i$ and $\mathbf{x}_j$. The graph cutting is implemented as a CUDA kernel to parallelize computation on GPU. The edge cost is taken to be $\sum_i \|\mathbf{P}_{i+1} - \mathbf{P}_i\|$, which serves as an upper bound on the path length of the Bézier curve from Property 2.49. In the implementation, about 5000 nodes were sampled for the graph. The high layer is capable of handling long term plans and highly nonconvex spaces, as it relies on a discrete system representation (the graph) and search methods, which scale with graph complexity rather than time, space, or nonconvexity.

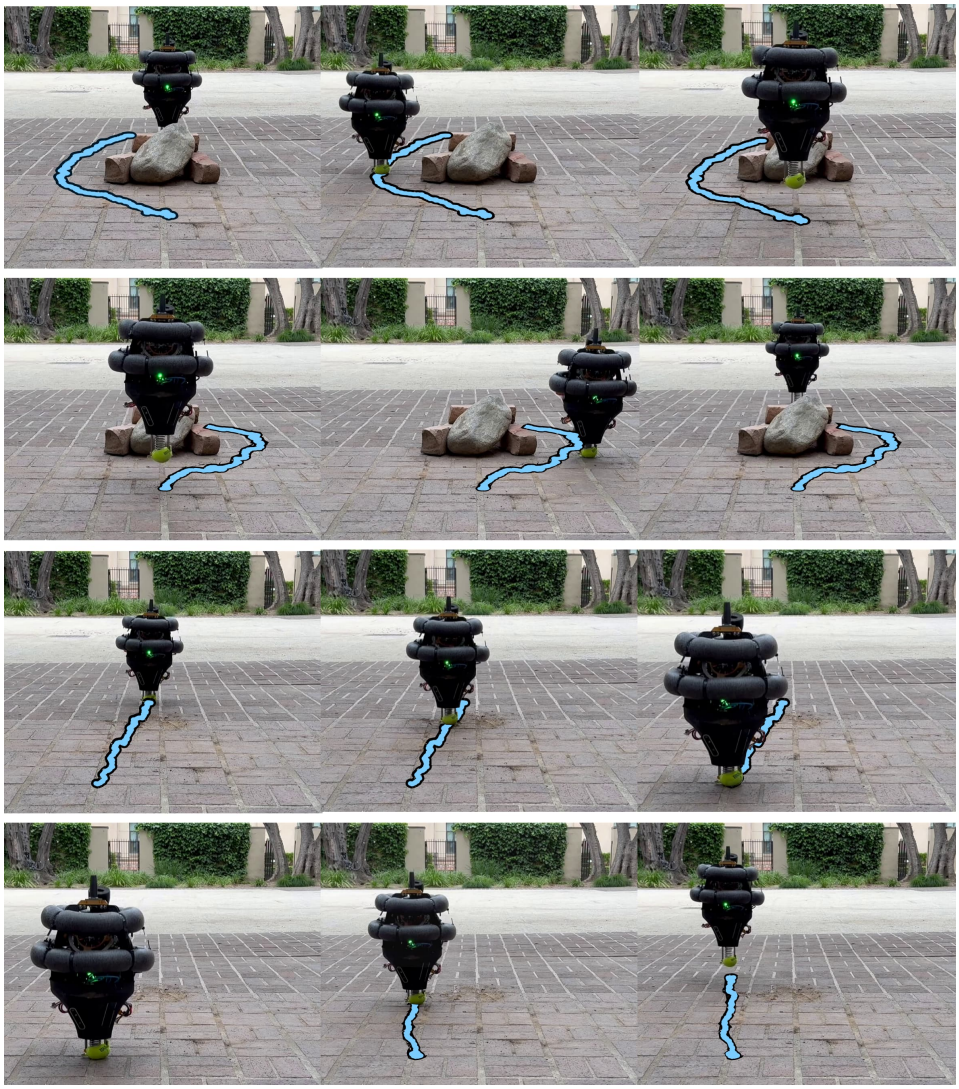


Figure 8.3: As the scene is updated, the robot is able to replan path through the available free space. Closed loop trajectories for the system are in light blue.

When running the system outdoors, or without the presence of the overhead obstacle segmenting camera, the approach for environmental construction differs. The Intel Realsense D435 camera, located on the front of the robot, as show in Figure 1.3, captures RGBD images at the apex of each hop. Traversible regions of the RGB image, such as sidewalk or brick walkway, are segmented using the Efficient Track-Anything-Model [201]. The pose of the hopper is used to transform the RGBD image into a point cloud, which is divided into free and occupied points by applying the segmentation mask. The point cloud is binned vertically, mapping points onto a 2D occupancy grid, which is sent to Algorithm 4 to construct the environment representation. All computations are carried out on the on board Nvidia ORIN.

Figure 8.2 depicts the robot's view of the environment, including the segmented terrain and convex polytope decomposition. Figure 8.3 shows that as the environment changes the robot is able to adapt its plan to navigate efficiently.

8.3 Summary

By properly abstracting the role of each layer of the hierarchy, we were able to provide system-level guarantees with independently designed components. Furthermore, we showed that these components satisfied the assumptions and guarantees that we set out to, thereby enabling a layered architecture with guarantees. Finally, we saw the complete architecture deployed on ARCHER, a highly underactuated 3D hopping robot, in unstructured outdoor environments.

Chapter 9: Conclusion

Contents

Be true to your passions, and the rest will follow.

9.1 Summary of Thesis

This thesis has developed a theory of layered control architectures, motivated by the need for modular, scalable control strategies for dynamic robotic systems. By formalizing layered architectures and their interfaces, we showed how independently designed controllers can be composed to enable system-level guarantees. We then specialized the discussion to address a common problem in robotics — long horizon, highly constrained planning for dynamical systems, with application to legged robots — and addressed it with high-level decision making, and low-level feedback. Along the way, we have provided constructive controller synthesis techniques at each layer and have avoided assuming the existence of functions or sets unless we have shown a constructive method for generating them. Furthermore, we have used experiments as our guiding light, helping us decide what tools are working and which should be abandoned.

The efficacy of this perspective of control has been validated through extensive experiments on a quadruped, biped, and hopping robot. These results demonstrate that layered architectures are not only an engineering convenience, but provide a principled approach to achieving *efficient, feasible, and generalizable* controller design. As such, we conclude by reiterating that layered architectures are a powerful tool for structured controller design.

9.2 Future Work

There are many, many research areas in robotics that are interesting and should be pursued. I will focus on listing a few that, given the context of this thesis, would be interesting to investigate:

- As we enter a paradigm of increasing parallel compute, there are opportunities where control could benefit from leveraging the GPU besides just training large models. Thinking about constructive areas to leverage this compute (as we did in the graph cutting for the Bézier graphs) could offer orders of magnitude speedup over parallel CPU implementations. There is a trade-off — GPUs can run (the same) simple operation on large batches of data, and therefore have a rigidity in what will benefit from their use. This constraint must be considered when choosing application areas for hardware acceleration.
- Refining the areas where zeroth order (sampling based) and first order (gradient based) optimization methods shine, both with and without learning and in various application contexts.

- Exploring the connections between trajectory optimization methods and generative methods of reaching optima, like flow matching for diffusion [202]. See [203], [204] for inspiration.

- Incorporating perception into the controller synthesis problem as done in [205], instead of leaving it as a least piece. Perception will be noisy, late, and wrong, and only considering those challenges at the end of the design process will fundamentally limit what can be achieved.

- As we move past pure locomotion as a field, the next frontier for humanoid robotics is understanding how locomotion and manipulation should be paired. The current solution of first walking, and then grabbing is a local minimum — instead we should think forward to methods that co-design these tasks.

- We need to understand what it means to relax the complementarity constraint in contact rich dynamics problems, especially as it pertains to controller synthesis. See [71] for inspiration.

9.3 Advice to Younger Students

The process of the PhD is long and grueling, so you may as well enjoy it. You are offered a lot of flexibility, both with your time and your research directions — use this to your advantage. Along the way, don't forget that your primary goal is to learn. In terms of research, seek opportunities that you think will help you develop the skills that you want to learn — there will always be questions to ask in the directions you want to learn. Also, try (as best you can) to pursue research that you truly find interesting. This is a good exercise in life in general, and will help keep you motivated. Finally, don't be afraid to break robots. As scary as it is, the only way to find the boundary of performance is by pushing past it every once in a while.

Bibliography

[1] *Humanoid robots run a Chinese half-marathon alongside flesh-and-blood competitors*, URL: https://apnews.com/article/china-robot-half-marathon-153c6823bd628625106ed26267874d21, 2025.

[2] K. Black, N. Brown, D. Driess, *et al.*, π_0*: A vision-language-action flow model for general robot control*, 2024. arXiv: 2410.24164 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2410.24164.

[3] Y. Nakahira, Q. Liu, T. J. Sejnowski, and J. C. Doyle, "Diversity-enabled sweet spots in layered architectures and speed–accuracy trade-offs in sensorimotor control," *Proceedings of the National Academy of Sciences*, vol. 118, no. 22, e1916367118, 2021.

[4] D. Kahneman, *Thinking, fast and slow*. New York: Farrar, Straus and Giroux, 2011. [Online]. Available: https://www.amazon.de/Thinking-Fast-Slow-Daniel-Kahneman/dp/0374275637/ref=wl_it_dp_o_pdT1_nS_nC?ie=UTF8&colid=151193SNGKJT9&coliid=I3OCESLZCVDFL7.

[5] H. Tsien, T. Adamson, and E. Knuth, "Automatic navigation of a long range rocket vehicle," *Journal of the American Rocket Society*, vol. 22, no. 4, pp. 192–199, 1952.

[6] *Apollo Guidance and Navigation : R-500 Space Navigation Guidance and Control ... 2 volumes by C. S. Draper, Dr. W. Wrigley, D. G. on Kuenzig Books*, en-US. (visited on 04/10/2024).

[7] B. Kuipers, E. A. Feigenbaum, P. E. Hart, and N. J. Nilsson, "Shakey: From conception to history," *AI Magazine*, vol. 38, no. 1, pp. 88–103, 2017. DOI: 10.1609/aimag.v38i1.2716. [Online]. Available: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2716.

[8] M. D. Mesarovic, D. Macko, and Y. Takahara, *Theory of hierarchical, multilevel systems*. Elsevier, 1970, vol. 68.

[9] G. J. Balas, "Flight control law design: An industry perspective," *European Journal of Control*, vol. 9, no. 2, pp. 207–226, 2003, ISSN: 0947-3580. DOI: https://doi.org/10.3166/ejc.9.207-226. [Online]. Avail-

able: https://www.sciencedirect.com/science/article/pii/S0947358003702763.

[10] S. Kuindersma, R. Deits, M. Fallon, *et al.*, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous Robots*, vol. 40, pp. 429–455, 2016.

[11] A. Dixit, D. D. Fan, K. Otsu, S. Dey, A.-A. Agha-Mohammadi, and J. W. Burdick, "Step: Stochastic traversability evaluation and planning for risk-aware navigation; results from the darpa subterranean challenge," *IEEE Transactions on Field Robotics*, vol. 2, pp. 81–99, 2025. DOI: 10.1109/TFR.2024.3512433.

[12] N. Matni, A. D. Ames, and J. C. Doyle, *Towards a Theory of Control Architecture: A quantitative framework for layered multi-rate control*, arXiv:2401.15185 [cs, eess, math], Jan. 2024. [Online]. Available: http://arxiv.org/abs/2401.15185 (visited on 04/02/2024).

[13] U. Rosolia, A. Singletary, and A. D. Ames, "Unified multirate control: From low-level actuation to high-level planning," *IEEE Transactions on Automatic Control*, vol. 67, no. 12, pp. 6627–6640, 2022, Publisher: IEEE. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9802791/ (visited on 04/01/2024).

[14] M. M. Jr., W. Compton, M. H. Cohen, and A. D. Ames, *A contract theory for layered control architectures*, 2024. arXiv: 2409.14902 [eess.SY]. [Online]. Available: https://arxiv.org/abs/2409.14902.

[15] M. H. Cohen, T. G. Molnar, and A. D. Ames, "Safety-critical control for autonomous systems: Control barrier functions via reduced-order models," *Annual Reviews in Control*, vol. 57, p. 100 947, 2024.

[16] G. Zardini, "Co-design of complex systems: From autonomy to future mobility systems," Ph.D. dissertation, ETH Zurich, 2023.

[17] D. Fridovich-Keil, S. L. Herbert, J. F. Fisac, S. Deglurkar, and C. J. Tomlin, *Planning, Fast and Slow: A Framework for Adaptive Real-Time Safe Trajectory Planning*, arXiv:1710.04731 [cs], Mar. 2018. DOI: 10.48550/arXiv.1710.04731. [Online]. Available: http://arxiv.org/abs/1710.04731 (visited on 04/02/2024).

[18] N. J. Nilsson, "A mobile automaton: An application of artificial intelligence techniques," en, Defense Technical Information Center, Fort Belvoir, VA, Tech. Rep., Jan. 1969. DOI: 10.21236/ADA459660. [Online]. Available: http://www.dtic.mil/docs/citations/ADA459660 (visited on 03/23/2023).

[19] J. Wang, T. Zhang, N. Ma, *et al.*, "A survey of learning-based robot motion planning," en, *IET Cyber-Systems and Robotics*, vol. 3, no. 4, pp. 302–314, 2021, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/csy2.12020,

ISSN: 2631-6315. DOI: 10.1049/csy2.12020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1049/csy2.12020 (visited on 05/19/2023).

[20] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, *Motion Planning Networks: Bridging the Gap Between Learning-based and Classical Motion Planners*, en, arXiv:1907.06013 [cs], Jun. 2020. [Online]. Available: http://arxiv.org/abs/1907.06013 (visited on 05/19/2023).

[21] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," en, in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, arXiv:1609.07910 [cs], May 2017, pp. 1527–1533. DOI: 10.1109/ICRA.2017.7989182. [Online]. Available: http://arxiv.org/abs/1609.07910 (visited on 05/19/2023).

[22] M. J. Bency, A. H. Qureshi, and M. C. Yip, *Neural Path Planning: Fixed Time, Near-Optimal Path Generation via Oracle Imitation*, en, arXiv:1904.11102 [cs], Apr. 2019. [Online]. Available: http://arxiv.org/abs/1904.11102 (visited on 05/19/2023).

[23] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter, "Perceptive locomotion through nonlinear model-predictive control," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3402–3421, 2023.

[24] A. Isidori, "Elementary theory of nonlinear feedback for single-input single-output systems," en, in *Nonlinear Control Systems*, ser. Communications and Control Engineering, A. Isidori, Ed., London: Springer, 1995, pp. 137–217, ISBN: 978-1-84628-615-5. DOI: 10.1007/978-1-84628-615-5_4. (visited on 01/30/2024).

[25] P. Kokotović and M. Arcak, "Constructive nonlinear control: A historical perspective," *Automatica*, vol. 37, no. 5, pp. 637–662, 2001.

[26] H. K. Khalil, *Nonlinear systems; 3rd ed.* Upper Saddle River, NJ: Prentice-Hall, 2002, The book can be consulted by contacting: PH-AID: Wallet, Lionel. [Online]. Available: https://cds.cern.ch/record/1173048.

[27] R. Sepulchre, M. Jankovic, and P. V. Kokotovic, *Constructive nonlinear control*. Springer Science & Business Media, 2012.

[28] Z. Artstein, "Stabilization with relaxed controls," *Nonlinear Analysis: Theory, Methods & Applications*, vol. 7, no. 11, pp. 1163–1173, 1983.

[29] E. D. Sontag, "A 'universal' construction of artstein's theorem on nonlinear stabilization," *Systems & Control Letters*, vol. 13, no. 2, pp. 117–123, 1989.

[30] E. D. Sontag, "Smooth stabilization implies coprime factorization," *Transactions on Automatic Control*, vol. 34, no. 4, pp. 435–443, 1989.

[31] R. Freeman and P. V. Kokotovic, *Robust nonlinear control design: state-space and Lyapunov techniques*. Birkhauser Basel, 1996.

[32] A. D. Ames and M. Powell, "Towards the unification of locomotion and manipulation through control Lyapunov functions and quadratic programs," in *Control of Cyber-Physical Systems*, Springer, 2013, pp. 219–240.

[33] S. Kolathaya, J. Reher, A. Hereid, and A. D. Ames, "Input to state stabilizing control Lyapunov functions for robust bipedal robotic locomotion," in *American Control Conference (ACC)*, IEEE, 2018, pp. 2224–2230.

[34] S. Sastry, "Linearization by state feedback," en, in *Nonlinear Systems: Analysis, Stability, and Control*, ser. Interdisciplinary Applied Mathematics, S. Sastry, Ed., New York, NY: Springer, 1999, pp. 384–448, ISBN: 978-1-4757-3108-8. DOI: 10.1007/978-1-4757-3108-8_9. (visited on 10/15/2023).

[35] M. Vukobratović and B. Borovac, "Zero-moment point—Thirty five years of its life," *International Journal of Humanoid Robotics*, vol. 1, no. 01, pp. 157–173, 2004.

[36] M. H. Raibert, H. B. Brown, and M. Chepponis, "Experiments in balance with a 3D one-legged hopping machine," en, *The International Journal of Robotics Research*, vol. 3, no. 2, pp. 75–92, Jun. 1984, Publisher: SAGE Publications Ltd STM, ISSN: 0278-3649. DOI: 10.1177/027836498400300207. (visited on 09/16/2022).

[37] S. Kajita, M. Morisawa, K. Miura, *et al.*, "Biped walking stabilization based on linear inverted pendulum tracking," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2010, pp. 4489–4496.

[38] H. Geyer, A. Seyfarth, and R. Blickhan, "Compliant leg behavior explains basic dynamics of walking and running," *Proceedings in Biological Sciences / The Royal Society*, vol. 273, pp. 2861–7, Dec. 2006. DOI: 10.1098/rspb.2006.3637.

[39] R. J. Full and D. E. Koditschek, "Templates and anchors: Neuromechanical hypotheses of legged locomotion on land," *Journal of Experimental Biology*, vol. 202, no. 23, pp. 3325–3332, 1999, Publisher: The Company of Biologists Ltd. [Online]. Available: https://journals.biologists.com/jeb/article-abstract/202/23/3325/8334 (visited on 04/01/2024).

[40] E. Westervelt, J. Grizzle, and D. Koditschek, "Hybrid zero dynamics of planar biped walkers," *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 42–56, Jan. 2003, Conference Name: IEEE Transactions on Automatic Control, ISSN: 1558-2523. DOI: 10.1109/TAC.2002.806653. (visited on 03/15/2024).

[41] A. Hereid, E. A. Cousineau, C. M. Hubicki, and A. D. Ames, "3D dynamic walking with underactuated humanoid robots: A direct collocation framework for optimizing hybrid zero dynamics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 1447–1454. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7487279/ (visited on 04/02/2024).

[42] Y. Gong, R. Hartley, X. Da, *et al.*, "Feedback control of a Cassie bipedal robot: Walking, standing, and riding a segway," in *American Control Conference (ACC)*, IEEE, 2019, pp. 4559–4566.

[43] Q. Nguyen, X. Da, J. Grizzle, and K. Sreenath, "Dynamic walking on stepping stones with gait library and control barrier functions," in *Algorithmic Foundations of Robotics XII*, Springer, 2020, pp. 384–399.

[44] J. Reher and A. D. Ames, "Inverse dynamics control of compliant hybrid zero dynamic walking," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 2040–2047.

[45] X. Da and J. Grizzle, "Combining trajectory optimization, supervised machine learning, and model structure for mitigating the curse of dimensionality in the control of bipedal robots," en, *The International Journal of Robotics Research*, vol. 38, no. 9, pp. 1063–1097, Aug. 2019, ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364919859425. (visited on 04/02/2024).

[46] F. Allgower, R. Findeisen, Z. K. Nagy, *et al.*, "Nonlinear model predictive control: From theory to application," *J.-Chinese Institute Of Chemical Engineers*, vol. 35, no. 3, pp. 299–316, 2004.

[47] F. Allgöwer and A. Zheng, *Nonlinear model predictive control*. Birkhäuser, 2012, vol. 26.

[48] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[49] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter, "Perceptive locomotion through nonlinear model-predictive control," *IEEE Transactions on Robotics*, 2023, Publisher: IEEE. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10138309/ (visited on 04/01/2024).

[50] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the MIT Cheetah 3 through convex model-predictive control," in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE/RSJ, 2018, pp. 1–9.

[51] J.-P. Sleiman, F. Farshidian, M. V. Minniti, and M. Hutter, "A unified MPC framework for whole-body dynamic locomotion and manipulation," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4688–4695, 2021.

[52] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *Transactions on Control Systems Technology*, vol. 15, no. 3, pp. 566–580, 2007.

[53] D. Hrovat, S. Di Cairano, H. E. Tseng, and I. V. Kolmanovsky, "The development of model predictive control in automotive industry: A survey," in *International Conference on Control Applications*, IEEE, 2012, pp. 295–302.

[54] P. F. Lima, G. C. Pereira, J. Mårtensson, and B. Wahlberg, "Experimental validation of model predictive control stability for autonomous driving," *Control Engineering Practice*, vol. 81, pp. 244–255, 2018.

[55] S. Bengea, A. Kelman, F. Borrelli, R. Taylor, and S. Narayanan, "Model predictive control for mid-size commercial building hvac: Implementation, results and energy savings," in *International Conference on Building Energy and Environment*, 2012, pp. 979–986.

[56] G. Serale, M. Fiorentini, A. Capozzoli, D. Bernardini, and A. Bemporad, "Model predictive control (MPC) for enhancing building and hvac system energy efficiency: Problem formulation, applications and opportunities," *Energies*, vol. 11, no. 3, p. 631, 2018.

[57] E. T. Maddalena, Y. Lian, and C. N. Jones, "Data-driven methods for building control—a review and promising future directions," *Control Engineering Practice*, vol. 95, p. 104 211, 2020.

[58] U. Rosolia and F. Borrelli, "Learning how to autonomously race a car: A predictive control approach," *Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2713–2719, 2019.

[59] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[60] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using gaussian process regression," *Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2019.

[61] A. Carvalho, Y. Gao, A. Gray, H. E. Tseng, and F. Borrelli, "Predictive control of an autonomous ground vehicle using an iterative linearization approach," in *International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2013, pp. 2335–2340.

[62] P. M. Wensing, M. Posa, Y. Hu, A. Escande, N. Mansard, and A. Del Prete, "Optimization-based control for dynamic legged robots," *IEEE Transactions on Robotics*, vol. 40, pp. 43–63, 2023.

[63] M. Y. Galliker, N. Csomay-Shanklin, R. Grandia, *et al.*, *Bipedal locomotion with nonlinear model predictive control: Online gait generation using whole-body dynamics*, Publication Title: arXiv preprint arXiv:2203.07429, 2022. [Online]. Available: `https://ieeexplore.ieee.org/abstract/document/10000132/` (visited on 04/01/2024).

[64] N. Csomay-Shanklin, V. D. Dorobantu, and A. D. Ames, "Nonlinear model predictive control of a 3D hopping robot: Leveraging lie group integrators for dynamically stable behaviors," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 12 106–12 112. (visited on 04/01/2024).

[65] S. Kuindersma. "Recent progress on atlas, the world's most dynamic humanoid robot." Accessed: 2025-04-22. (2023), [Online]. Available: https://www.youtube.com/watch?v=EGABAx52GKI.

[66] C. Khazoom, S. Hong, M. Chignoli, E. Stanger-Jones, and S. Kim, "Tailoring solution accuracy for fast whole-body model predictive control of legged robots," *preprint arXiv:2407.10789*, 2024. arXiv: 2403.03995 [cs.RO].

[67] H. Li and P. M. Wensing, "Cafe-mpc: A cascaded-fidelity model predictive control framework with tuning-free whole-body control," *preprint arXiv:2403.03995*, 2024. arXiv: 2403.03995 [cs.RO].

[68] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations*, 2016.

[69] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, eabk2822, 2022.

[70] Z. Li, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, "Reinforcement learning for versatile, dynamic, and robust bipedal locomotion control," *preprint arXiv:2401.16889*, 2024. arXiv: 2401.16889 [cs.RO].

[71] H. J. Suh, M. Simchowitz, K. Zhang, and R. Tedrake, "Do differentiable simulators give better policy gradients?" In *International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2022, pp. 20 668–20 696.

[72] G. Pappas, G. Lafferriere, and S. Sastry, "Hierarchically consistent control systems," *IEEE Transactions on Automatic Control*, vol. 45, no. 6, pp. 1144–1160, 2000. DOI: 10.1109/9.863598.

[73] A. Girard and G. J. Pappas, "Hierarchical control using approximate simulation relations," in *Proceedings of the 45th IEEE Conference on Decision and Control*, 2006, pp. 264–269. DOI: 10.1109/CDC.2006.377051.

[74] A. van der Schaft, "Equivalence of dynamical systems by bisimulation," *IEEE Transactions on Automatic Control*, vol. 49, no. 12, pp. 2160–2172, 2004. DOI: 10.1109/TAC.2004.838497.

[75] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017, ISBN: 9780975937730. [Online]. Available: https://books.google.com/books?id=MrJctAEACAAJ.

[76] D. Q. Mayne, M. M. Seron, and S. V. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219–224, 2005.

[77] M. Chen, S. L. Herbert, H. Hu, *et al.*, "Fastrack: A modular framework for real-time motion planning and guaranteed safe tracking," *IEEE Transactions on Automatic Control*, vol. 66, no. 12, pp. 5861–5876, 2021. DOI: 10.1109/TAC.2021.3059838.

[78] E. D. Sontag, "Input to state stability: Basic concepts and results," in *Nonlinear and Optimal Control Theory*, Springer, 2008, pp. 163–220.

[79] N. Csomay-Shanklin, A. J. Taylor, U. Rosolia, and A. D. Ames, "Multi-rate planning and control of uncertain nonlinear systems: Model predictive control and control Lyapunov functions," *arXiv:2204.00152*, 2022.

[80] A. Wu, S. Sadraddini, and R. Tedrake, "R3t: Rapidly-exploring random reachable set tree for optimal kinodynamic planning of nonlinear hybrid systems," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4245–4251. DOI: 10.1109/ICRA40945.2020.9196802.

[81] Y. Gao, A. Gray, H. E. Tseng, and F. Borrelli, "A tube-based robust nonlinear predictive control approach to semiautonomous ground vehicles," *Vehicle System Dynamics*, vol. 52, no. 6, pp. 802–823, 2014.

[82] M. Kögel and R. Findeisen, "Discrete-time robust model predictive control for continuous-time nonlinear systems," in *American Control Conference (ACC)*, IEEE, 2015, pp. 924–930.

[83] S. Yu, C. Maier, H. Chen, and F. Allgöwer, "Tube mpc scheme based on robust control invariant set with application to lipschitz nonlinear systems," *Systems & Control Letters*, vol. 62, no. 2, pp. 194–200, 2013.

[84] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone, "Robust online motion planning via contraction theory and convex optimization," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 5883–5890.

[85] J. Köhler, R. Soloperto, M. A. Müller, and F. Allgöwer, "A computationally efficient robust model predictive control framework for uncertain nonlinear systems," *Transactions on Automatic Control*, vol. 66, no. 2, pp. 794–801, 2020.

[86] S. Singh, M. Chen, S. L. Herbert, C. J. Tomlin, and M. Pavone, "Robust tracking with model mismatch for fast and safe planning: An SOS optimization approach," in *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Springer, 2018, pp. 545–564.

[87] H. Yin, M. Bujarbaruah, M. Arcak, and A. Packard, "Optimization based planner–tracker design for safety guarantees," in *American Control Conference (ACC)*, IEEE, 2020, pp. 5194–5200.

[88] U. Rosolia, A. Singletary, and A. D. Ames, "Unified multi-rate control: From low-level actuation to high-level planning," *Transactions on Automatic Control*, 2022.

[89] U. Rosolia and A. D. Ames, "Multi-rate control design leveraging control barrier functions and model predictive control policies," *Control Systems Letters*, vol. 5, no. 3, pp. 1007–1012, 2021. DOI: 10.1109/LCSYS.2020.3008326.

[90] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the Association for Computing Machinery*, vol. 40, no. 5, pp. 1048–1066, Nov. 1993, ISSN: 0004-5411.

[91] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001, ISSN: 0278-3649, 1741-3176.

[92] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," en, in *2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany: IEEE, May 2013, pp. 5054–5061. DOI: 10.1109/ICRA.2013.6631299. [Online]. Available: http://ieeexplore.ieee.org/document/6631299/ (visited on 01/30/2023).

[93] E. Schmerling and M. Pavone, "Kinodynamic Planning," in *Encyclopedia of Robotics*, M. H. Ang, O. Khatib, and B. Siciliano, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2021, pp. 1–9, ISBN: 978-3-642-41610-1.

[94] S. M. LaValle, "Rapidly-exploring random trees : A new tool for path planning," *The Annual Research Report*, 1998.

[95] A. Shkolnik, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 2859–2865. DOI: 10.1109/ROBOT.2009.5152874.

[96] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards.," en, *Pacific Journal of Mathematics*, Oct. 1990. DOI: 10.2140/pjm.1990.145.367. [Online]. Available: https://www.scinapse.io/papers/1971998222 (visited on 05/19/2023).

[97] A. Shkolnik, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," in *2009 IEEE International Conference on Robotics and Automation*, ser. 1, ISSN: 1050-4729, May 2009, pp. 2859–2865. DOI: 10.1109/ROBOT.2009.5152874.

[98] A. Wu, S. Sadraddini, and R. Tedrake, "R3T: Rapidly-exploring random reachable set tree for optimal kinodynamic planning of nonlinear hybrid systems," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, ser. 1, ISSN: 2577-087X, May 2020, pp. 4245–4251. DOI: 10.1109/ICRA40945.2020.9196802.

[99] D. Ioan, I. Prodan, S. Olaru, F. Stoican, and S.-I. Niculescu, "Mixed-integer programming in motion planning," *Annual Reviews in Control*, vol. 51, pp. 65–87, 2021.

[100] R. Deits and R. Tedrake, "Efficient mixed-integer planning for UAVs in cluttered environments," en, in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, ser. 1, Seattle, WA, USA: IEEE, May 2015, pp. 42–49, ISBN: 978-1-4799-6923-4. DOI: 10.1109/ICRA.2015.7138978. [Online]. Available: http://ieeexplore.ieee.org/document/7138978/ (visited on 02/22/2023).

[101] T. Marcucci, J. Umenberger, P. A. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," *arXiv:2101.11565*, 2021.

[102] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, *Motion Planning around Obstacles with Convex Optimization*, en, arXiv:2205.04422 [cs], May 2022. [Online]. Available: http://arxiv.org/abs/2205.04422 (visited on 01/30/2023).

[103] J. Lee, E. Bakolas, and L. Sentis, "An efficient and direct method for trajectory optimization of robots constrained by contact kinematics and forces," *Autonomous Robots*, vol. 45, pp. 135–153, 2021.

[104] S. Stoneman and R. Lampariello, "Embedding nonlinear optimization in rrt for optimal kinodynamic planning," in *53rd IEEE Conference on Decision and Control*, IEEE, 2014, pp. 3737–3744.

[105] P. Fernbach, S. Tonneau, A. Del Prete, and M. Taix, "A kinodynamic steering-method for legged multi-contact locomotion," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 3701–3707.

[106] E. Ambrose, W. Ma, C. Hubicki, and A. D. Ames, "Toward benchmarking locomotion economy across design configurations on the modular robot: Amber-3m," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*, 2017, pp. 1270–1276. DOI: 10.1109/CCTA.2017.8062633.

[107] E. R. Ambrose, "Creating ARCHER: A 3D Hopping Robot with Flywheels for Attitude Control," en, Ph.D. dissertation, California Institute of Technology, 2022. DOI: 10.7907/gbts-va63. (visited on 09/16/2022).

[108] K. Astrom and R. Murray, *Feedback Systems: An Introduction for Scientists and Engineers, Second Edition*. Princeton University Press, 2021, ISBN: 9780691193984. [Online]. Available: https://books.google.com/books?id=l50DEAAAQBAJ.

[109] S. S. Sastry, *Nonlinear Systems: Analysis, Stability and Control*. NY: Springer, 1999.

[110] M. Nagumo, "Ueber die lage der integralkurven gewoehnlicher differentialgleichungen," *Proceedings of the Physico-Mathematical Society of Japan. 3rd Series*, vol. 24, pp. 551–559, 1942.

[111] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2017.

[112] A. Ames, J. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *Conference on Decision & Control (CDC)*, IEEE, 2014, pp. 6271–6278.

[113] M. Spivak, *Calculus on manifolds: a modern approach to classical theorems of advanced calculus* (Mathematics monograph series), en. CRC Press, Taylor & Francis Group, 2018, ISBN: 978-0-8053-9021-6.

[114] A. D. Ames and I. Poulakakis, "Hybrid zero dynamics control of legged robots," *Bioinspired Legged Locomotion: Models, Concepts, Control and Applications*, pp. 292–331, 2017.

[115] Q. Nguyen and K. Sreenath, "Exponential control barrier functions for enforcing high relative-degree safety-critical constraints," in *American Control Conference (ACC)*, IEEE, 2016, pp. 322–328.

[116] A. J. Taylor, A. Singletary, Y. Yue, and A. D. Ames, "A control barrier perspective on episodic learning via projection-to-state safety," *IEEE Control Systems Letters*, vol. 5, no. 3, pp. 1019–1024, 2021.

[117] W. F. Ames and B. Pachpatte, *Inequalities for differential and integral equations*. Elsevier, 1997, vol. 197.

[118] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer.

[119] M. Kamermans, "A primer on Bézier curves," *(online book)*, 2020. [Online]. Available: `https://pomax.github.io/bezierinfo/`.

[120] N. Csomay-Shanklin and A. D. Ames, "Bezier reachable polytopes: Efficient certificates for robust motion planning with layered architectures," *arXiv preprint arXiv:2411.13506*, 2024.

[121] T. Marcucci, P. Nobel, R. Tedrake, and S. Boyd, *Fast Path Planning Through Large Collections of Safe Boxes*, en, arXiv:2305.01072 [cs, eess], May 2023. [Online]. Available: `http://arxiv.org/abs/2305.01072` (visited on 05/19/2023).

[122] Autonomy Talks, *Autonomy Talks - Animesh Garg: Building Blocks of Generalizable Autonomy: Duality of Discovery&Bias*, Nov. 2022. [Online]. Available: `https://www.youtube.com/watch?v=oe9DUvI_lUU` (visited on 04/03/2024).

[123] R. Bellman, "Dynamic programming," *Princeton University Press*, 1957.

[124] I. Radosavovic, T. Xiao, B. Zhang, T. Darrell, J. Malik, and K. Sreenath, "Real-world humanoid locomotion with reinforcement learning," *Science Robotics*, vol. 9, no. 89, eadi9579, 2024.

[125] T. He, J. Gao, W. Xiao, *et al.*, "Asap: Aligning simulation and real-world physics for learning agile humanoid whole-body skills," *arXiv preprint arXiv:2502.01143*, 2025.

[126] T. He, C. Zhang, W. Xiao, G. He, C. Liu, and G. Shi, "Agile but safe: Learning collision-free high-speed legged locomotion," *arXiv preprint arXiv:2401.17583*, 2024.

[127] Y. Chen, M. Ahmadi, and A. D. Ames, "Optimal safe controller synthesis: A density function approach," in *2020 American Control Conference (ACC)*, IEEE, 2020, pp. 5407–5412.

[128] L. Chisci, J. Rossiter, and G. Zappa, "Systems with persistent disturbances: Predictive control with restricted constraints," *Automatica*, vol. 37, no. 7, pp. 1019–1028, 2001, ISSN: 0005-1098. DOI: https://doi.org/10.1016/S0005-1098(01)00051-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109801000516.

[129] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-Jacobi reachability: A brief overview and recent advances," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, IEEE, 2017, pp. 2242–2253.

[130] P. O. Scokaert and D. Q. Mayne, "Min-max feedback model predictive control for constrained linear systems," *IEEE Transactions on Automatic control*, vol. 43, no. 8, pp. 1136–1142, 1998.

[131] D. Mayne, M. Seron, and S. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219–224, 2005, ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2004.08.019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109804002870.

[132] J. Primbs, V. Nevistic, and J. Doyle, "A receding horizon generalization of pointwise min-norm controllers," *IEEE Transactions on Automatic Control*, vol. 45, no. 5, pp. 898–909, May 2000, Conference Name: IEEE Transactions on Automatic Control, ISSN: 1558-2523. DOI: 10.1109/9.855550.

[133] P. Tabuada, *Verification and control of hybrid systems: A symbolic approach*. Springer Science & Business Media, 2009.

[134] *Quadrupedal robotic walking on sloped terrains.* https://youtu.be/uJedboyzDjc.

[135] M. Tucker, E. Novoseller, C. Kann, *et al.*, "Preference-based learning for exoskeleton gait optimization," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 2351–2357.

[136] A. D. Ames, K. Galloway, K. Sreenath, and J. W. Grizzle, "Rapidly exponentially stabilizing control Lyapunov functions and hybrid zero dynamics," *IEEE Transactions on Automatic Control*, vol. 59, no. 4, pp. 876–891, 2014. DOI: 10.1109/TAC.2014.2299335.

[137] K. Galloway, K. Sreenath, A. D. Ames, and J. W. Grizzle, "Torque saturation in bipedal robotic walking through control Lyapunov function-based quadratic programs," *IEEE Access*, vol. 3, pp. 323–332, 2015.

[138] J. Reher, C. Kann, and A. D. Ames, *An inverse dynamics approach to control Lyapunov functions*, 2020. arXiv: 1910.10824 [cs.RO].

[139] M. Tucker, M. Cheng, E. Novoseller, *et al.*, "Human preference-based learning for high-dimensional optimization of exoskeleton walking gaits," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020.

[140] *Learning controller gains on bipedal walking robots via user preferences.* https://youtu.be/jMX5a_6Xcuw?si=w9KSizYj6S16A66F.

[141] A. J. Taylor, V. D. Dorobantu, H. M. Le, Y. Yue, and A. D. Ames, "Episodic learning with control lyapunov functions for uncertain robotic systems," in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 6878–6884.

[142] A. J. Taylor, A. Singletary, Y. Yue, and A. D. Ames, "Learning for safety critical control with control barrier functions," *arXiv preprint arXiv:1912.10099*, 2019.

[143] R. Grandia, A. J. Taylor, A. D. Ames, and M. Hutter, "Multi-layered safety for legged robots via control barrier functions and model predictive control," *arXiv preprint arXiv:2011.00032*, 2020.

[144] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 895–902, 2018.

[145] *Episodic learning for safe bipedal locomotion with control barrier functions and projection-to-state safety*, https://vimeo.com/481809664.

[146] Learning Code, https://github.com/noelc-s/amber_learning.

[147] E. R. Westervelt, J. W. Grizzle, C. Chevallereau, J. H. Choi, and B. Morris, *Feedback Control of Dynamic Bipedal Robot Locomotion*. Taylor & Francis/CRC Press, 2007.

[148] K. Sreenath, H.-W. Park, I. Poulakakis, and J. W. Grizzle, "A compliant hybrid zero dynamics controller for stable, efficient and fast bipedal walking on mabel," *The International Journal of Robotics Research*, vol. 30, no. 9, pp. 1170–1193, 2011.

[149] J. Reher, W.-L. Ma, and A. D. Ames, "Dynamic walking with compliance on a cassie bipedal robot," in *European Control Conference (ECC)*, IEEE, 2019, pp. 2589–2595.

[150] A. Hereid, S. Kolathaya, M. S. Jones, J. Van Why, J. W. Hurst, and A. D. Ames, "Dynamic multi-domain bipedal walking with atrias through slip based human-inspired control," in *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, 2014, pp. 263–272.

[151] *Preference-based learning for user-guided hzd gait generation on bipedal robots*, `https://youtu.be/rLJ-m65F6C4`.

[152] *Supplementary website featuring full-length experimental videos.* `https://maegant.github.io/ICRA2021-LearningHZD/`.

[153] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017, vol. 2.

[154] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," in *Fast Motions in Biomechanics and Robotics*, Springer, 2006, pp. 65–93.

[155] C. Feller and C. Ebenbauer, "Relaxed logarithmic barrier function based model predictive control of linear systems," *Transactions on Automatic Control*, vol. 62, no. 3, pp. 1223–1238, 2016.

[156] *OCS2: An open source library for optimal control of switched systems*, [Online]. Available: `https://github.com/leggedrobotics/ocs2`.

[157] J. Carpentier, G. Saurel, G. Buondonno, *et al.*, "The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *International Symposium on System Integration (SII)*, IEEE/SICE, 2019, pp. 614–619.

[158] B. M. Bell, "CppAD: A package for C++ algorithmic differentiation," *Computational Infrastructure for Operations Research*, vol. 57, no. 10, 2012.

[159] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 93–100.

[160] C. Mastalli, R. Budhiraja, W. Merkt, *et al.*, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 2536–2542.

[161] C. D. Bellicoso, F. Jenelten, P. Fankhauser, C. Gehring, J. Hwangbo, and M. Hutter, "Dynamic locomotion and whole-body control for quadrupedal robots," in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE/RSJ, 2017, pp. 3359–3365.

[162] H. Ferrolho, V. Ivan, W. Merkt, I. Havoutis, and S. Vijayakumar, "Inverse dynamics vs. forward dynamics in direct transcription formulations for trajectory optimization," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 12 752–12 758.

[163] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[164] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[165] A. Hereid and A. D. Ames, "Frost∗: Fast robot optimization and simulation toolkit," in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE/RSJ, 2017, pp. 719–726.

[166] A. D. Ames, "Human-inspired control of bipedal walking robots," *Transactions on Automatic Control*, vol. 59, no. 5, pp. 1115–1130, 2014.

[167] *Bipedal locomotion with nonlinear model predictive control*, https://youtu.be/3g8ZNsCWdOA.

[168] J. Stillwell, "The matrix logarithm," en, in *Naive Lie Theory*, ser. Undergraduate Texts in Mathematics, J. Stillwell, Ed., New York, NY: Springer, 2008, pp. 139–159, ISBN: 978-0-387-78215-7. DOI: 10.1007/978-0-387-78214-0_7. [Online]. Available: https://doi.org/10.1007/978-0-387-78214-0_7 (visited on 09/14/2022).

[169] J. Solà, J. Deray, and D. Atchuthan, *A micro Lie theory for state estimation in robotics*, en, arXiv:1812.01537 [cs], Dec. 2021. (visited on 07/27/2022).

[170] *Robust agility via learned zero dynamics policies: 3D hopping*. https://vimeo.com/923800815.

[171] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, ISSN: 2153-0866, Oct. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[172] *Code*, 2024. [Online]. Available: https://github.com/ivandariojr/LearnedZeroDynamicsPolicies.

[173] A. D. Ames, P. Tabuada, A. Jones, *et al.*, "First steps toward formal controller synthesis for bipedal robots with experimental implementation," en, *Nonlinear Analysis: Hybrid Systems*, vol. 25, pp. 155–173, Aug. 2017, ISSN: 1751570X. DOI: 10.1016/j.nahs.2017.01.002. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1751570X1730002X (visited on 12/05/2021).

[174] I. D. J. Rodriguez, A. D. Ames, and Y. Yue, *Lyanet: A Lyapunov framework for training neural odes*, 2022. DOI: 10.48550/ARXIV.2202.02526. [Online]. Available: https://arxiv.org/abs/2202.02526.

[175] S.-C. Hsu, X. Xu, and A. D. Ames, "Control barrier function based quadratic programs with application to bipedal robotic walking," in *2015 American Control Conference (ACC)*, IEEE, 2015, pp. 4542–4548.

[176] J. E. Pratt and S. V. Drakunov, "Derivation and application of a conserved orbital energy for the inverted pendulum bipedal walking model," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, IEEE, 2007, pp. 4653–4660.

[177] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[178] *Learning to walk by enforcing forward invariance*, `https://www.youtube.com/watch?v=8TeXd0AYtpA`.

[179] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 895–902, 2018. DOI: 10.1109/LRA.2018.2792536.

[180] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[181] Y. Liu, J. Bernstein, M. Meister, and Y. Yue, "Learning by turning: Neural architecture aware optimisation," in *International Conference on Machine Learning*, PMLR, 2021, pp. 6748–6758.

[182] *Learning Code*. [Online]. Available: `https://github.com/ivandariojr/NeuralGaits` (visited on 12/07/2021).

[183] X. Da and J. Grizzle, "Combining trajectory optimization, supervised machine learning, and model structure for mitigating the curse of dimensionality in the control of bipedal robots," *The International Journal of Robotics Research*, vol. 38, no. 9, pp. 1063–1097, 2019. DOI: 10.1177/0278364919859425.

[184] B. Han, H. Yi, Z. Xu, X. Yang, and X. Luo, "3d-slip model based dynamic stability strategy for legged robots with impact disturbance rejection," *Scientific Reports*, vol. 12, no. 1, p. 5892, 2022.

[185] J. Reher and A. D. Ames, "Control Lyapunov functions for compliant hybrid zero dynamic walking," *preprint arXiv:2107.04241*, 2021.

[186] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton University Press, 2012, ISBN: 978-0-691-15187-8. DOI: 10.2307/j.ctvcm4g0s. (visited on 01/31/2024).

[187] N. Csomay-Shanklin, W. D. Compton, I. D. J. Rodriguez, E. R. Ambrose, Y. Yue, and A. D. Ames, "Robust agility via learned zero dynamics policies," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2024, pp. 9116–9123.

[188] W. D. Compton, I. D. J. Rodriguez, N. Csomay-Shanklin, Y. Yue, and A. D. Ames, "Constructive nonlinear control of underactuated systems via zero dynamics policies," in *2024 IEEE 63rd Conference on Decision and Control (CDC)*, 2024, pp. 8350–8357. DOI: 10.1109/CDC56724.2024.10886411.

[189] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*, PMLR, 2017, pp. 136–145.

[190] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 1168–1175.

[191] J. Bradbury, R. Frostig, P. Hawkins, *et al.*, "Jax: Composable transformations of python+numpy programs," *GitHub. Note: https://github.com/google/jax*, 2018.

[192] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[193] J. Deray and J. Solà, "Manif: A micro Lie theory library for state estimation in robotics applications," *Journal of Open Source Software*, vol. 5, no. 46, p. 1371, 2020. DOI: 10.21105/joss.01371.

[194] M. H. Raibert, H. B. Brown Jr, and M. Chepponis, "Experiments in balance with a 3d one-legged hopping machine," *The International Journal of Robotics Research*, vol. 3, no. 2, pp. 75–92, 1984.

[195] X. Zhong, Y. Wu, D. Wang, Q. Wang, C. Xu, and F. Gao, *Generating large convex polytopes directly on point clouds*, 2020. arXiv: 2010.08744 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2010.08744.

[196] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[197] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[198] *Obstacle avoidance on a 3D hopping robot*, https://vimeo.com/1009702220.

[199] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. DOI: 10.1007/s12532-020-00179-2.

[200] A. Kirillov, E. Mintun, N. Ravi, *et al.*, "Segment anything," *arXiv:2304.02643*, 2023.

[201] Y. Xiong, C. Zhou, X. Xiang, *et al.*, "Efficient track anything," *preprint arXiv:2411.18933*, 2024.

[202] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, *Flow matching for generative modeling*, 2023. arXiv: 2210.02747 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2210.02747.

[203] V. Kurtz and J. W. Burdick, "Generative predictive control: Flow matching policies for dynamic and difficult-to-demonstrate tasks," *arXiv preprint arXiv:2502.13406*, 2025.

[204] V. Kurtz and J. W. Burdick, "Equality constrained diffusion for direct trajectory optimization," *arXiv preprint arXiv:2410.01939*, 2024.

[205] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," en, *Science Robotics*, vol. 7, no. 62, eabk2822, Jan. 2022, ISSN: 2470-9476. DOI: 10.1126/scirobotics.abk2822. [Online]. Available: https://www.science.org/doi/10.1126/scirobotics.abk2822 (visited on 04/01/2024).

Index