Perception-Driven Autonomy and Learning Control for Ground Vehicles

Thesis by
Elena Sorina Lupu

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy in Space Engineering

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2025
Defended January 31st, 2025

Elena Sorina Lupu
ORCID: 0000-0002-3968-2630

ACKNOWLEDGEMENTS

I would like to thank my advisor, Prof. Soon-Jo Chung, for guiding and believing in me and for allowing me to explore very interesting research topics. He was also an example of hard-work, rigor, and perseverance for me that I greatly appreciated. I would also like to thank a number of other professors who have provided guidance and support, including Prof. Yisong Yue, Prof. Joel Burdick, Prof. Fred Hadaegh, Prof. Charles Elachi, Dr. Amir Rahmani, and Prof. Morteza Gharib. Thank you to Prof. John Dabiri and Prof. Richard Murray for serving on my defense committee and qualifying exam committee, and for interesting discussions.

I would also like to thank my many 💙 collaborators (who also became my best friends): Dr. Rebecca Foust, Dr. Matt Anderson, Dr. Xichen Shi, Dr. James Preiss, Alexei Harvard, Nikhil Ranganathan, John Lathrop, Dr. Benjamin Riviere, and Hannah Grauer.

My other amazing collaborators and lab mates (random order): Jedi Alindogan, Dr. Kai Matsuka, Fengze Xie, Joshua Cho, Dr. Vincenzo Capuano, Dr. Thomas Berrueta, Prof. Xingxing Zuo, Prof. Yashwanth Nakka, Joshua Ibrahim, Yujin An, Joudi Hajar, Satvik Kumar, Haeyoon Han, Deemo Chen, Prof. SooJean Han, Prof. Lu Gan, Prof. Kyunam Kim, Prof. Guanya Shi, Prof. Hiroyasu Tsukamoto, Prof. Kyunam Kim, Prof. Wolfgang Hoenig, Dr. Connor Lee, Dr. Michael O'Connell, Dr. Ellande Tang, Dr. Salar Rahili, Dr. Daniel Mitchell, Arion Zimmermann, Mathieu Decker.

My best friends 💙 outside the lab who made the time at Caltech really entertaining and lovely: Dr. Stephanie O'Gara, Dr. Tracy Lu, and Dr. Sahangi Dassanayake.

I would like to thank my DARPA Linc collaborators and JPL friends: Thomas Touma, Anna Sabel, Dr. Jonathan Beckton, Dr. Ersin Das, Dr. Skylar Wei, and many others. I would like to thank our amazing assistants Christine Ramirez, Narin Seraydarian, Paula Mark, Jamie Meighen-Sei, Liza Bradulina, and Martha Salcedo. The CAST crew Reza Nemovi and Noel Esparza-Duran which allowed me to use the CAST facilities and provided me with the necessary help for my research.

My many students that I mentored throughout these years at Caltech: Leo Zhang, Sulekha Kishore, Cailyn Smith, Jade Millan, Julia Donovan, Asta Wu, Aaron Feldman, Ray Sun, and many others.

I also had the privilege to take classes and be inspired by professor such as Prof. Konstantin Zuev, Prof. Joanna Austin, Prof. Joel Burdick, Prof. Venkat Chandrasekaran, and Prof. Eugene Lavretsky.

I would like to thank the Keck Institute of Space Science (KISS) family, especially 💙 Michele Judd and Harriet Brettle, the staff and affiliates of KISS, for providing invaluable opportunities and guidance as an affiliate of KISS.

To my collaborators from the 🚀 Aerospace Corporation such as Phaedrus Leeds, Darren W. Rowen, and Benjamin P. Bycroft.

I am profoundly grateful to my family for their endless support and inspiration. My parents, 💙 Cristina and 💙 Valentin, have always been supportive of my interests in engineering. My mom, an engineer with a passion for science fiction, ignited my early fascination with space exploration. My father, who uniquely combines two degrees in engineering and law and served many years as a prison warden, has always been my cornerstone of support and instilled in me the principles of fairness. I would like to thank my grandmother, Bimba, for always encouraging me to be confident, and to my aunt, Tanti Milica, for guiding me through life. Both of them always encourged me to be an engineer like themselves and I am now a 4th generation engineer in my family. I would like to thank my husband, 💙 Patrick Spieler, for his endless support, help, and engaging discussions. He has been my rock and my best friend throughout this journey.

This leads me to thanking the number of funding sources that have supported my research, including Defense Advanced Research Projects Agency (DARPA), Amazon AI4Science Fellowship, and the Aerospace Corporation.

ABSTRACT

Autonomous robots are widely recognized as highly valuable and are expected to become increasingly prevalent. They will play a critical role across a wide range of terrestrial applications in complex, unstructured environments, as well as in space, supporting infrastructure and exploration on various bodies throughout the solar system and beyond. Looking ahead, autonomous robots will play a crucial role in the search for extraterrestrial life by enabling exploration of remote and extreme environments beyond Earth. As robots need to approach more complex tasks, the ability to rapidly perceive, understand, make real-time decisions, and operate at speed requires advances in perception-driven controls, improved predictability, and robustness to disturbances. To enable these capabilities, the first part of this thesis proposes an innovative approach to enhancing ground vehicle mobility by integrating a vision-based control algorithm that adapts to changes in real-time. Our approach improves the vehicle's ability to assess and respond to complex terrains in real-time by leveraging visual information through visual foundation models and meta-learning. Our controller has provable guarantees of exponential stability and was validated on board two ground vehicles. Next, an extension of the previously mentioned method applied to detecting objects in space using a visual foundation model is presented. Our method was successfully demonstrated in space in early 2025 aboard the EdgeNode Lite spacecraft. Efficient operation comes from the synergy of suitable autonomy and control with a suitable robot body. Following this consideration, the second part of the thesis presents the design and control of multi-degrees of freedom robots designed for mobility in complex environments. It presents a nonlinear tracking controller with adaptation to improve the walking performance of walking-flying robots. This is illustrated by our implementation on Leonardo, the first robot to combine walking with flying to create a new type of locomotion, which we showcase in complex acrobatic movements such as slacklining and skateboarding. In a second case study, we aim to further understand and improve biped walking by introducing a bipedal robot designed to be lightweight, easily manufactured, and easily repaired, serving as a platform for testing learning-based controllers. We introduce and demonstrate the performance of two controllers: a model-based and a learning-based control. This work highlights the importance of tightly integrated perception, control, and electromechanical design in achieving robust autonomy: on Earth, in orbit, and beyond.

PUBLISHED CONTENT AND CONTRIBUTIONS

[1] **E. S. Lupu***, F. Xie*, J. A. Preiss, J. Alindogan, M. Anderson, and S.-J. Chung. "MAGIC-VFM: Meta-learning Adaptation for Ground Interaction Control with Visual Foundation Models". In: *IEEE Transactions of Robotics* 10.03 (2024), p. C03021. DOI: 10.1109/TRO.2024.3475212.
E.S.L. proposed the concept of the project, formulated theoretical proofs, implemented the experiments on two hardware platforms, analyzed simulation and hardware data, and participated in the writing of the manuscript.

[2] H. Grauer*, **E. S. Lupu***, C. Lee, D. Rowen, B. Bycroft, P. Leeds, J. Brader, and S.-J. Chung. "Vision-Based Detection of Uncooperative Targets and Components on Small Satellites". In: *38th Annual Small Satellite Conference* (2024), pp. 5321–5327. DOI: 10.1126/scirobotics.abf8136. URL: https://arxiv.org/abs/2408.12084.
E.S.L. developed the concept of the project, designed the close-range detection algorithm using knowledge distillation and implemented the algorithm on the flight hardware, ran experiments, analyzed data, and contributed to the writing of the manuscript.

[3] K. Kim*, P. Spieler*, **E. S. Lupu**, A. Ramezani, and S.-J. Chung. "A Bipedal Walking Robot that can Fly, Slackline, and Skateboard". In: *Science Robotics* 6.59 (2021), eabf8136. DOI: 10.1126/scirobotics.abf8136.
E.S.L. developed the nonlinear control system for the walking of the robot, the convergence proof of the control algorithm, ran experiments, and contributed to the writing of the manuscript.

[4] R. C. Foust, **E. S. Lupu**, Y. K. Nakka, S.-J. Chung, and F. Y. Hadaegh. "Autonomous In-orbit Satellite Assembly from a Modular Heterogeneous Swarm". In: *Acta Astronautica* 169 (2020), pp. 191–205. DOI: https://doi.org/10.1016/j.actaastro.2020.01.006.
E.S.L. designed the low-level controller that executed the trajectories from the high-level planner, ran experiments, as well as contributed to the writing of the manuscript.

[5] X. Shi, P. Spieler, E. Tang, **E. S. Lupu**, P. Tokumaru, and S.-J. Chung. "Adaptive Nonlinear Control of Fixed-Wing VTOL with Airflow Vector Sensing". In: *IEEE International Conference on Robotics and Automation (ICRA)* (2020), pp. 5321–5327. DOI: 10.1109/ICRA40945.2020.9197344.
E.S.L. ran experiments, analyzed data, and contributed to the writing of the manuscript.

[6] K. Matsuka, **E. S. Lupu**, Y. Nakka, R. Foust, S.-J. Chung, and F. Y. Hadaegh. "Distributed Multi-target Relative Pose Estimation for Cooperative Spacecraft Swarm". In: *10th International Workshop on Satellite Constellations and Formation Flying* (2019). URL: https://www.researchgate.net/

publication/334623422_Distributed_multi-target_relative_pose_estimation_for_cooperative_spacecraft_swarm.
E.S.L. implemented the algorithm on hardware and developed the detection algorithm using AruCo markers, ran experiments, as well as contributed to the writing of the manuscript.

[7] R. Foust, **E. S. Lupu**, Y. Nakka, D. B. Elliott, I. S. Crowell, S.-J. Chung, and F. Y. Hadaegh. "Ultra-soft Electromagnetic Docking with Applications to In-orbit Assembly". In: *International Astronautical Congress, 2018* (2018).
E.S.L. developed the electromagnets and the docking ports and their control system, designed the low-level controller that executed the trajectories from the high-level planner, ran experiments, and contributed to the writing of the manuscript.

[8] Y. Nakka, R. Foust, **E. S. Lupu**, D. B. Elliott, I. S. Crowell, S.-J. Chung, and F. Y. Hadaegh. "Six Degree-of-Freedom Spacecraft Dynamics Simulator for Formation Control Research". In: *AAS/AIAA Astrodynamics Specialist Conference* (2018). URL: https://www.researchgate.net/publication/327067426_Six_Degree-of-Freedom_Spacecraft_Dynamics_Simulator_for_Formation_Control_Research.
E.S.L. developed the electrical system, designed the control hardware and software for the thrusters and reaction wheels, ran experiments, and contributed to manuscript writing.

The * denotes co-first.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Number *Page*

Chapter 1

INTRODUCTION

1.1 Motivation

Autonomous robots are widely recognized as highly valuable and are expected to become increasingly pervasive in the near future. They will play a critical role across a wide range of terrestrial applications in complex, unstructured environments, as well as in space, supporting infrastructure and exploration on various bodies throughout the solar system and beyond. Looking ahead, autonomous robots will one day play a crucial role in the search for extraterrestrial life by enabling exploration of remote and extreme environments beyond Earth.

Vision-based Autonomy and Real-time Adaptation

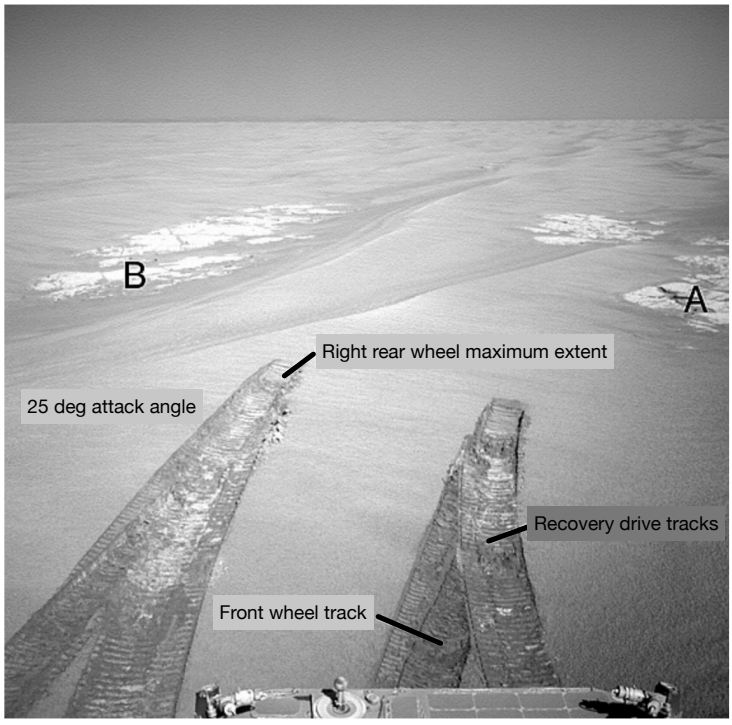


Figure 1.1: Image on-board the Opportunity rover on Mars: "Navcam image acquired on sol 2226 of the sol 2220 high slippage and sinkage location on the western side of a ripple." [1]

Planetary exploration presents unique challenges for robotic mobility due to the diverse and often treacherous terrain that can pose serious risks to robot safety. Such an example is the Opportunity rover, which experienced mobility issues caused by excessive wheel sinkage while crossing sand dunes. This incident highlights the limitations of current mobility systems on other planets that rely on pre-defined terrain models and and lack the ability to adapt to real-time environmental changes. In addition, the communication delay between Earth and Mars makes real-time teleoperation infeasible. While Mars is currently within reach with different rovers, we must explore further into the Solar Systems, to icy moons like Europa, Titan, and Enceladus [2], as well as asteroids and comets.

To enable autonomous operation in such unstructured settings, two key technologies are essential: **vision-based autonomy** and **real-time adaptation**. Vision-based autonomy provides robots with the ability to perceive and interpret their environment, while real-time adaptation allows them to dynamically adjust their behavior in response to terrain variations, hazards, and other environmental changes. Together, these capabilities are **critical and essential** for achieving robust, intelligent mobility in the next generation of space exploration missions.

Learning-based Control for Multi Degrees of Freedom Robots

Figure 1.2: Animals with outstanding mobility capabilities. **A.** Ibex climbing a dam. **B.** Ostrich, the fastest bipedal animal on Earth, reaching speeds of 48 kmph (30 mph) **C.** A cheetah is capable of moving at more than 104 kmph (64 mph).

Efficient operation comes from the synergy of suitable autonomy and control with the suitable robot body. A straightforward way to enhance the mobility of a robot is through bio-mimicry. Animals have evolved to have different forms that are adapted to their environment. Many of them have outstanding mobility capabilities that allow them to navigate through complex terrains with ease, as shown in Figure 1.2. In particular, legs give animals the ability to traverse rough terrains, climb obstacles, and navigate through confined spaces. Legs are not the only form of mobility that

animals have evolved. Recent studies show a close ontogenetic[1] and evolutionary relationships between legs and wings, such as of Drosophila [3]. The ability to fly gave insects and birds a three dimensional mobility that is unmatched by any other form of navigation. Learning from those evolutionary traits, robots can be designed to be capable of traversing through complex terrains with the same agility as animals.

1.2 Objectives and Scope

The *first* objective of this research is to explore the synergy between vision and control to enhance the autonomy of vehicles driving on unknown terrains. We focus on real-time adaptability and on methods that have theoretical guarantees. This study is driven by the hypothesis that a robot should not operate only on a small set of states such as position and velocity, but also on a high-dimensional representation of the environment. On that aspect, there is a gap in the literature on how to leverage vision information with control theory methods, while guaranteeing exponential stability.

The *second* objective is driven by the understanding that, while autonomy algorithms are key to enable robots to navigate through complex terrains, the physical form of the robot is also critical. This study explores how to design robots with multiple degrees of freedom—and develop corresponding control and learning algorithms that enable agile locomotion in complex environments, comparable to that observed in nature.

The *third* objective is to explore the deployment of learning-based algorithms onboard spacecraft. This study is driven by the hypothesis that such methods can significantly enhance autonomy, yet they remain largely unvalidated in space applications due to limitations in onboard computation, power, and communication. There is a clear gap in the literature on how to effectively adapt and implement these algorithms within the stringent constraints of space environments.

1.3 Contributions

The first part of the thesis (Chapters 2-4 and Figure 1.3**A**) proposes an innovative approach to enhance ground vehicle mobility by integrating a vision-based control algorithm with real time adaptability. By leveraging visual information through visual foundation models (VFMs) and meta-learning, our approach aims to improve

[1] of or relating to the origin and development of individual organisms

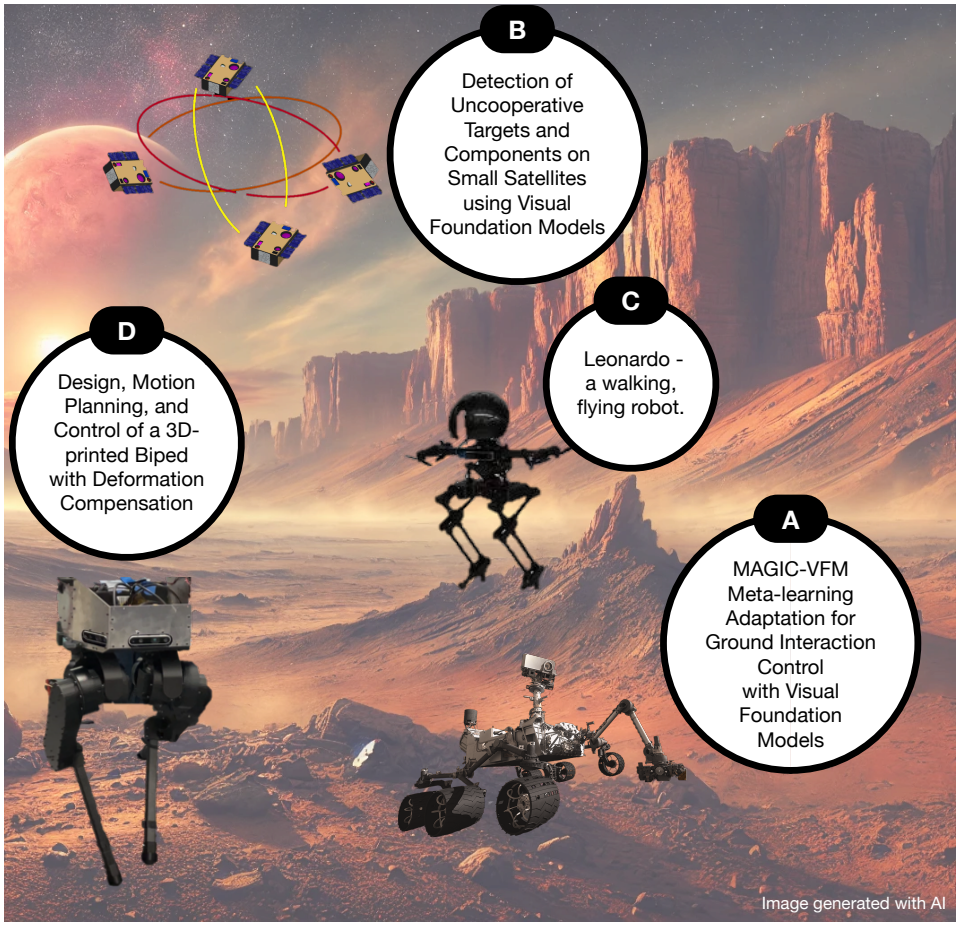


Figure 1.3: Outline of the main contributions of the thesis. **A.** MAGIC-VFM algorithm for ground vehicles. **B.** Visual foundation models-based detection algorithm for spacecraft applications. **C.** Leonardo robot capable of walking and flying. **D.** Biped robot for learning-based control.

the vehicle's ability to assess and respond to complex terrains in real time. Compared to the state-of-the-art, our algorithm is the *first* to propose a method that combines vision with adaptive control and has provable safety guarantees of exponential stability. We validated our method extensively on-board two different ground vehicles in various terrains. The results show that our approach outperforms state-of-the-art methods in terms of adaptability to different terrains. We have extended the VFM framework for a segmentation task using knowledge distillation and implemented it on board a spacecraft flying in space, as shown in Chapter 6 and Figure 1.3**B** Based on publicly available data, our algorithm is the *first* to demonstrate both training and testing of a deep neural network on-board an NVIDIA Jetson GPU in space, as shown in Section 6.5. The second part of the thesis (Chapters 7–8) explores the design and control of robots with multiple degrees of freedom. We first introduce

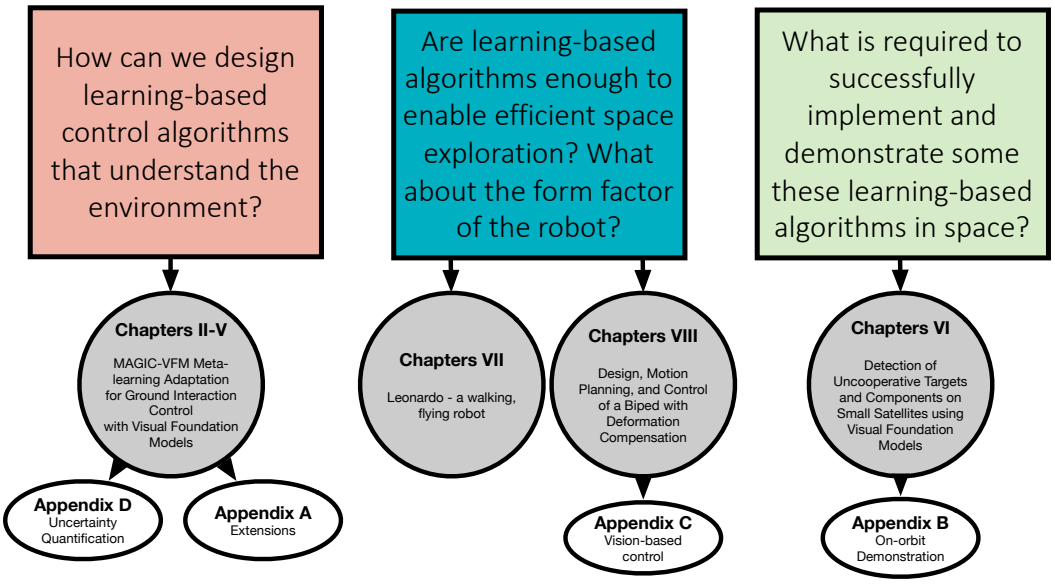


Figure 1.4: Three questions that are investigated by this thesis and thesis organization (chapters and appendices that address these questions).

Leonardo (Figure 1.3**C**), the *first* robot to integrate walking and flying, enabling a novel form of locomotion. Thanks to its high control authority provided by propellers, Leonardo can perform agile and previously unachievable maneuvers, like slacklining and skateboarding. We then present an open-source custom-designed bipedal robot (Figure 1.3**D**), developed to be lightweight and easily manufacturable. This platform serves as a testbed for advancing learning-based control strategies for robots with high degrees of freedom. Preliminary results on learning-based control for this biped are included in Appendix C. In summary, the three contributions outlined above aim to address the questions illustrated in Figure 1.4.

1.4 Thesis Outline

As shown in Figure 1.4, the thesis is outlined as follows:

- Chapters 2-4 introduce the MAGIC$^{\text{VFM}}$ Meta-learning Adaptation for Ground Interaction Control with Visual Foundation Models algorithm [4], with two extensions of using visual foundation models and adaptive control for motion planning (Chapter 5).
- Chapter 6 presents a learning-based algorithm for detecting spacecraft parts using visual foundation models and knowledge distillation [5]. This work is extended in Appendix B where we show the improvements made to the algorithm to be able to fly it in space on-board the EdgeNode Lite spacecraft.

For the space environment, the algorithm processes thermal images rather than red-green-blue (RGB) images.

- In Chapter 7, we introduce Leonardo [6], a robot capable of both walking and flying. We show the design, control and experimental results of the robot performing complex maneuvers such as slacklining and skateboarding.
- In Chapter 8, we present the design, motion planning and control of a lightweight and easy to manufacture biped with deformation compensation. This work is extended in Appendix C where some preliminary results on the learning-based control of this biped are outlined.
- In Appendix A, we extend some of the control formulations in Chapters 2-4 to a racing car for the Indy Autonomous Challenge and present some experimental results of the Gradient-based Adaptive Policy Selection (GAPS) [7] algorithm on-board an RC vehicle with front-wheels steering.
- In Appendix D, we introduce a model-based reinforcement learning framework that uses Bayesian Neural Networks to quantify uncertainty in the dynamics of a system and use it in planning.
- Appendix E provides an overview of the main concepts in control and robotics kinematics used throughout the thesis.

BIBLIOGRAPHY

[1] Raymond Arvidson et al. “Opportunity Mars Rover Mission: Overview and Selected Results from Purgatory Ripple to Traverses to Endeavour Crater”. In: *Journal Geophysical Research: Planets* 116.E7 (Feb. 2011).

[2] Tiago Vaquero et al. “EELS: Autonomous Snake-like Robot with Task and Motion Planning Capabilities for Ice World Exploration”. In: *Science Robotics* 9.88 (2024), eadh8332.

[3] Neil H. Shubin, Clifford J. Tabin, and Sean B. Carroll. “Fossils, Genes and the Evolution of Animal Limbs”. In: *Nature* 388 (1997), pp. 639–648.

[4] Elena-Sorina Lupu, Fengze Shi, James Preiss, Jedidiah Alindogan, Matthew Anderson, and Soon-Jo Chung. “MAGIC-VFM: Meta-learning Adaptation for Ground Interaction Control with Visual Foundation Models”. In: *IEEE Transactions on Robotics* (2024).

[5] Hannah Grauer, Elena Sorina Lupu, Connor Lee, Darren Rowen, Benjamen Bycroft, Phaedrus Leeds, John Brader, and Soon-Jo Chung. “Vision-Based Detection of Uncooperative Targets and Components on Small Satellites”. In: *38th Annual Small Satellite Conference* (2024), pp. 5321–5327.

[6] Kyunam Kim, Patrick Spieler, Elena-Sorina Lupu, Alireza Ramezani, and Soon-Jo Chung. “A Bipedal Walking Robot that can Fly, Slackline, and Skateboard”. In: *Science Robotics* 6.59 (2021), eabf8136.

[7] Yiheng Lin, James A. Preiss, Emile Anand, Yingying Li, Yisong Yue, and Adam Wierman. “Online Adaptive Policy Selection in Time-Varying Systems: No-Regret via Contractive Perturbations”. In: *Advances in Neural Information Processing Systems*. Vol. 36. Curran Associates, Inc., 2023, pp. 53508–53521.

Chapter 2

META-LEARNING ADAPTATION FOR GROUND INTERACTION CONTROL WITH VISUAL FOUNDATION MODELS (MAGIC-VFM)

> "The study of vision must include not only the study of how to extract from images the various aspects of the world that are useful to us, but also an inquiry into the nature of the internal representations by which we capture this information and thus make it available as a basis for decision about our thoughts and actions." (D. Marr, Vision: A computation Investigation into the Human Representation and Processing of Visual Information)

This chapter is based on the publication:

Elena-Sorina Lupu*, Fengze Shi*, James Preiss, Jedidiah Alindogan, Matthew Anderson, and Soon-Jo Chung. "MAGIC-VFM: Meta-learning Adaptation for Ground Interaction Control with Visual Foundation Models". In: *IEEE Transactions on Robotics* (2024)

Acknowledgments: I would like to acknowledge the co-authors of the paper for their contributions to this work, as follows: Fengze Shi for his help on the theoretical work, especially the proofs of Theorems 1 and 3 in Chapter 3, Prof. James Preiss for his help with the simulation results and manuscript writing, Jedidiah Alindogan and Matt Anderson for their help with the hardware experiments and manuscript writing, and Prof. Soon-Jo Chung for theoretical proofs, guidance, manuscript writing, and suggestions.

2.1 Motivation

Control of off-road vehicles is challenging due to the complex dynamic interactions with the terrain. Accurate modeling of these interactions is important to optimize driving performance, but the relevant physical phenomena, such as slip, are too complex to model from first principles. Therefore, we present an offline meta-

learning algorithm to construct a rapidly-tunable model of residual dynamics and disturbances. Our model processes terrain images into features using a visual foundation model (VFM), then maps these features and the vehicle state to an estimate of the current actuation matrix using a deep neural network (DNN). We then combine this model with composite adaptive control to modify the last layer of the DNN in real time, accounting for the remaining terrain interactions not captured during offline training. We provide mathematical guarantees of stability and robustness for our controller in Chapter 3 and demonstrate the effectiveness of our method through simulations and hardware experiments with a tracked vehicle and a car-like robot in Chapter 4. We evaluate our method outdoors on different slopes with varying slippage and actuator degradation disturbances, and compare against an adaptive controller that does not use the VFM terrain features. We show significant improvement over the baseline in both hardware experimentation and simulation.

2.2 Introduction

Autonomous Ground Vehicles (AGVs) are gaining popularity across numerous domains including agriculture applications [1–3], wilderness search and rescue missions [4–7], and planetary exploration [8]. In many of these scenarios, the AGVs operate on rugged surfaces where the ability to follow a desired trajectory degrades. To reliably operate in these environments with minimal human intervention, AGVs must understand the environment and adapt to it in real time. Slippage is one of the primary challenges encountered by ground vehicles while operating on loose terrain. For rovers exploring other planets, slippage can slow down their progress and even halt their scientific objectives. For instance, the Opportunity rover recorded significant slippage and sinking of its wheels during the Mars Day 2220 [9] while traversing sand ripples. During its climb, the slip, calculated based on visual odometry [10], was high, and thus the drive was halted and rerouted by the ground operators.

To better understand the effects of terradynamics, researchers have designed sophisticated models [11] that inform the design, simulation, and control of ground vehicles. However, these models have numerous assumptions and are often limited when the vehicles are operated at their performance boundaries (e.g., steering at high speeds and instances of non-uniform resistive forces like stumps and stones). In addition, designing controllers that consider these complex models is challenging. For control, kinematic models, such as Dubins, are often employed due to their simplicity and intuitive understanding. However, these models are not able to capture the

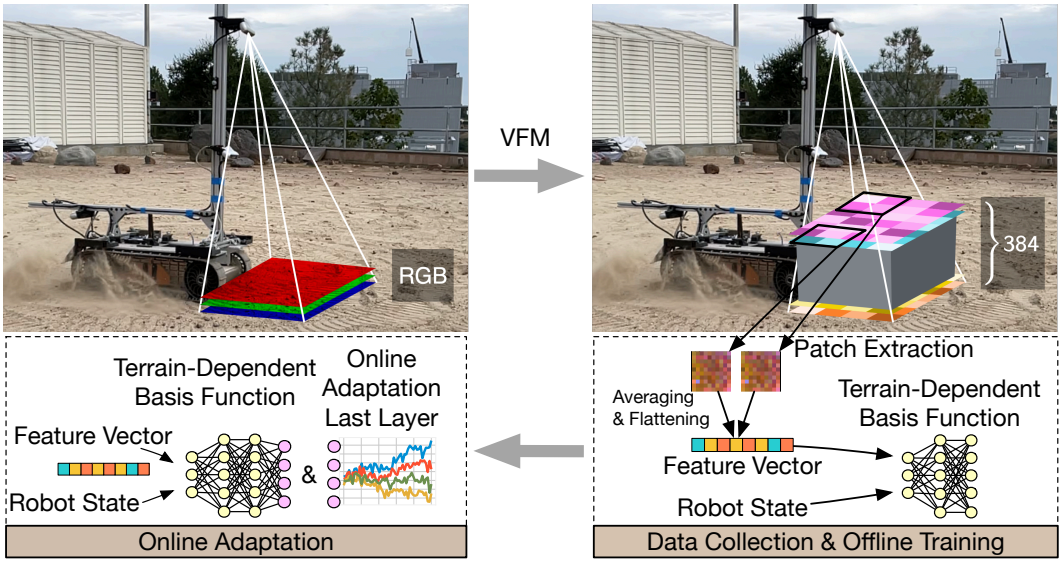


Figure 2.1: MAGIC$^{\text{VFM}}$: An offline meta-learning algorithm to build a residual dynamics and disturbance model using both Visual Foundation Models (VFM) and vehicle states. This model is integrated with composite adaptive control to adapt to changes in both the terrain and vehicle dynamics conditions in real time. See https://youtu.be/sxM73ryweRA

complicated dynamics between the vehicle and the ground, nor other disturbances such as internal motor dynamics or wheel or track degradation. To increase the performance of ground vehicles, more comprehensive models are necessary.

Controllers that stabilize a ground vehicle and track desired trajectories amidst a variety of disturbances are crucial for achieving optimal vehicle performance. Oftentimes, the bottleneck is not in the controller design per se, but rather in the choice and complexity of the model utilized by the controller. Recently, reinforcement learning (RL) has shown significant promise in facilitating the development of efficient controllers through experiential learning [12–14]. The combination of meta-learning [15–22] and adaptive control [23–31] demonstrates considerable potential in accurately estimating unmodeled dynamics, efficiently addressing domain shift challenges and real-time adaptation to new environments [18, 22, 32, 33]. Despite this progress, incorporating a suitable model into a controller/policy is still an active area of research, especially when combined with theoretical and safety guarantees.

Learning sophisticated unmodeled dynamics based only on a limited set of vehicle states is ill-posed given that the operating environment is infinite dimensional. To accurately represent a complete dynamics model, including learned residual terms

for control, it is imperative to leverage as much information about the environment as possible. For instance, visual information can inform the model about the type of terrain in which the vehicle is operating. Previous work includes segmentation-based models that assign a discrete terrain type to each area in the image. This information is further employed in planning and control [34]. However, in off-road applications, categorizing terrains into a limited number of classes such as snow, mud, sand, etc. is not sufficient. There are infinite subcategories within each terrain type, each presenting distinct effects on the vehicle. In addition, two terrains can appear similar yet induce different dynamic behaviors on the robot (e.g., deep and shallow sand). Therefore, finding the most accurate and robust representation of the environment is indispensable for vehicle control over complex terrain.

Contributions

To address these limitations, we present MAGIC$^{\text{VFM}}$ (**M**eta-learning **A**daptation for **G**round **I**nteraction **C**ontrol with **V**isual **F**oundation **M**odels), an approach that integrates a VFM with meta-learning and composite adaptive control, thereby enabling ground vehicles to navigate and adapt to complex terrains in real time. Our method is well-suited for any ground vehicle equipped with the following: 1) sensors to measure the internal robot state, 2) exteroceptive sensors that can capture the terrain such as cameras, 3) the availability of a pre-trained VFM, and 4) the necessary computation hardware to evaluate the VFM in real-time. Our contributions are:

- the first stable *learning-based adaptive controller* that incorporates *visual foundation models* for terrain adaptation in real time;
- an offline meta-learning algorithm that uses continuous trajectory data to train and learn the terrain disturbance as a function of visual terrain information and vehicle states;
- mathematical guarantees of exponential stability and robustness against model errors for adaptive control with visual and state input that works in conjunction with our meta-learning offline training algorithm;
- the development of a position, attitude, and velocity tracking control formulation with the control influence matrix adaptation that can handle a variety of other perturbations in real-time such as unknown time-varying track or motor degradation and arbitrary time-varying disturbances.

We validate the effectiveness of our method both through simulation and in hardware on two heterogeneous robotic platforms, demonstrating its performance outdoors, on slopes with different slippage, as well as under track degradation disturbances.

Notation

Unless otherwise noted, all vector norms are Euclidean and all matrix norms are the Euclidean operator norm. We denote the floor operator by $\lfloor\cdot\rfloor$. Given $\mathbf{A} \in \mathbb{R}^{n\times m\times p}$ and $\mathbf{b} \in \mathbb{R}^p$, the notation $(\mathbf{Ab})$ is defined as $\sum_{i=1}^{p} \mathbf{A}_i b_i$. The notation $\|\mathbf{x}\|_{\mathbf{P}}$ for positive semi-definite matrix $\mathbf{P}$ defines the weighted inner product $\sqrt{\mathbf{x}^\top \mathbf{P}\mathbf{x}}$. For a function $f : X \mapsto Y$ where X and Y are metric spaces with metrics d_X and d_Y, we define $\|f\|_{\text{Lip}} = \max_{x,x'\in X} d_Y(f(x), f(x'))/d_X(x, x')$. For a measurable set X, we denote the set of probability measures on X by $\triangle X$, and if a uniform distribution on X exists, we denote it by $\mathcal{U}X$. When clear from context, we overload the notation $[i, j]$ to denote the integer sequence $i, \ldots, j$. All matrices and vectors are written in bold.

2.3 Related Work

The term *meta-learning*, first coined in [35], most often refers to learning protocols in which there is an underlying set of related learning tasks/environments, and the learner leverages data from previously seen tasks to adapt rapidly to a new task [16, 17, 36, 37]. The goal is to adapt more rapidly than would be possible for a standard learning algorithm presented with the new task in isolation. In robotics, meta-learning has been used to accurately adapt to highly dynamic environments [18, 22, 32, 33]. Online meta-learning [27, 38–40] includes two phases: offline meta-training and online adaptation. In the offline phase, the goal is to learn a model that performs well across all environments using a meta-objective. Given limited real-world data, the online adaptation phase aims to use online learning, such as adaptive control [23], to adapt the offline-learned model to a new environment in real time.

Some examples of meta-learning algorithms from the literature are Model-Agnostic Meta-Learning (MAML) [17] with its online extension [40], Meta-learning via Online Changepoint Analysis (MOCA) [41], and Domain Adversarially Invariant Meta-Learning (DAIML) used in Neural-Fly [27]. For task-centered datasets, MAML [17] trains the parameters of a model to achieve optimal performance on a new task with minimal data, by updating these parameters through one or more gradient steps based on that task's dataset. In continuous problems, tasks often lack

clear segmentation, resulting in the agent being unaware of task transitions. Therefore, MOCA [42] proposes task unsegmented meta-learning via online changepoint analysis. DAIML [27] proposes an online meta-learning-based approach where a shared representation is learned offline (for example, using data from different wind conditions for a quadrotor), with a linear, low-dimensional part updated online through adaptive control.

Our method builds on the previous work [27] on the integration of adaptive control and offline meta-learning to build a comprehensive model for ground vehicles. We develop a meta-learning algorithm that uses continuous trajectories from a robot driving on different terrains to learn a representation of the dynamics residual common across these terrains. This representation is a DNN that encodes the terrain information through vision, together with a set of linear parameters that adapt online at runtime. These linear parameters can be interpreted as the last layer of the DNN [27] and are terrain independent but encapsulate the remaining disturbances not captured during training.

Embedding Visual Information in Classical Control and Reinforcement Learning

One of the early works on including visual information for control is visual servoing [43], a technique mainly used for robot manipulation. Recently, vision-based reinforcement learning (VRL) has demonstrated the capability to control agents in simulated environments[44], as well as robots in real environments, with applications to ground robots [45–47] and robot manipulation [48–50]. This capability is achieved by leveraging high-fidelity models in robotics simulators [51, 52], imitation learning from human demonstrations or techniques to bridge the sim-to-real gap of the learned policy [14]. Nevertheless, a limitation of VRL is that the generated policy remains uninterpretable and does not have safety and robustness guarantees. To address the uninterpretability aspect, recent advancements in Inverse Reinforcement Learning (IRL) offer promising algorithms for interpreting terrain traversability as a reward map, thus enhancing the understanding of the environments [53, 54]. Despite progress in combining vision with Reinforcement Learning (RL), incorporating a suitable terrain model into a policy is still an open area of research, especially when combined with theoretical guarantees and safety properties.

To this end, we derive a nonlinear adaptive tracking controller for AGVs that uses a learned ground model with vision information incorporated in the control influ-

ence matrix. Our method processes camera images that are then passed through a VFM to synthesize the relevant features. These features, together with the robot's state, are incorporated into the ground-robot interaction model learned offline using meta-learning. For this adaptive controller, we prove exponential convergence to a bounded error ball.

Visual Foundation Models in Robotics

A foundation model is a large-scale machine learning model trained on a broad dataset that can be adapted, fine-tuned, or built upon for a variety of applications. Self-supervised learned VFMs, such as Dino and DINOv2 [55], are foundation models that are based on visual transformers [56, 57]. These models are trained to perform well on several downstream tasks, including image classification, semantic segmentation, and depth estimation. In robotics, these foundation models are starting to gain popularity in tasks such as image semantic segmentation [58, 59], traversability estimation [54], and robot manipulation [60, 61]. One of their key advantages lies in the robustness against variations in lighting and occlusions [62], as well as their ability to generalize well across different images of the same context. By consuming raw images as inputs, these self-supervised learning foundation models possess the potential to learn all-purpose visual features if pre-trained on a large quantity of data.

Adaptation to Ground Disturbances

Adaptive control [23–31, 63] is a control method with provable convergence guarantees in which a set of linear parameters is adapted online to compensate for disturbances at runtime. Typically, these linear parameters are multiplied by a basis function, which can be constant (as in the case of integral control), derived from physics [64], or represented using Radial Basis Functions (RBFs) [65] or DNNs [27]. First introduced in [23, 66], composite adaptation combines online parameter estimation and tracking-error adaptive control. A rigorous robustness/stability analysis for composite adaptation with a connection to deep meta-learning was first derived in [27] for flight control applications.

Ground vehicles (including cars, tracked vehicles, and legged robots) should be adaptive to changes in the terrain conditions. This adaptability is essential for optimal performance and safety in diverse environments [67–72]. In [73], an adaptive energy-aware prediction and planning framework for vehicles navigating over terrains with varying and unknown properties was proposed and demonstrated in

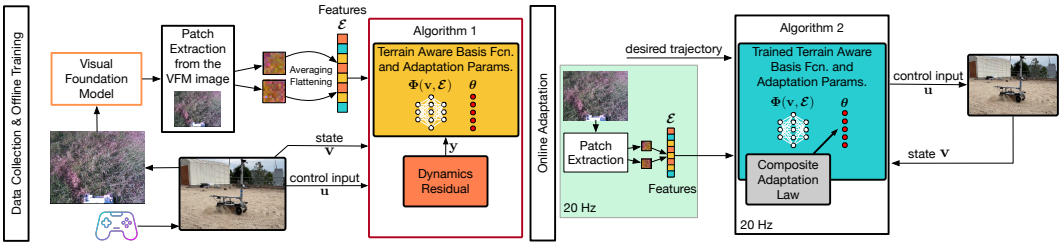


Figure 2.2: Terrain-aware Architecture: offline data collection and training (Algorithm 1), followed by real-time adaptive control (Algorithm 2) running onboard the robot.

simulation. [74] proposes a deep meta-learning framework for learning a global terrain traversability prediction network that is integrated with a sampling-based model predictive controller, while [75] develops a probabilistic traction model with uncertainty quantification using both semantic and geometric terrain features. In [76], a meta-learning-based approach to adapt probabilistic predictions of rover dynamics with Bayesian regression is used. While these approaches succeeded in developing different models for control and planning, incorporating a suitable model that fits the adaptive control framework is an open area of research.

In this paper, we establish an adaptive controller that can handle a broad range of real-time perturbations, such as unknown time-varying track or motor degradation, under controllability assumptions, arbitrary time-varying disturbances, and model uncertainties. This work can be viewed as a generalization and improvement of [27] with a new control matrix adaptation method using visual information and improved stability results. Our adaptive capability acts as an enhancement to the offline trained basis function, further improving tracking performance under challenging conditions.

2.4 Methods

Residual Dynamics Representation using VFM

We consider an uncertain dynamical system model

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) + \mathbf{d}, \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ denotes the state, $\mathbf{u} \in \mathbb{R}^m$ denotes the control input, $\mathbf{f} : \mathbb{R}^{n+m+1} \mapsto \mathbb{R}^n$ denotes the nominal dynamics model, and $\mathbf{d}$ is an unknown disturbance that is possibly time-varying and state- and environment-dependent. Our algorithm approximates $\mathbf{d}$ by

$$\mathbf{d} = \boldsymbol{\Phi}(\mathbf{x}, \mathbf{u}, \boldsymbol{\mathcal{E}})\boldsymbol{\theta}(t) + \boldsymbol{\delta}, \tag{2.2}$$

where $\boldsymbol{\theta}(t) = \begin{bmatrix}\theta_1 \dots \theta_{n_\theta}\end{bmatrix}^\top \in \mathbb{R}^{n_\theta}$ denotes a time-varying vector of linear parameters that are adapted online by our algorithm, $\boldsymbol{\delta} \in \mathbb{R}^n$ is a representation error, and $\boldsymbol{\mathcal{E}} \in \mathbb{R}^p$ is a feature vector representation of the terrain surrounding the robot computed by a VFM. From the perspective of the adaptive control part of our method, the precise form of $\boldsymbol{\mathcal{E}}$ is not required in our work. We can think of $\boldsymbol{\mathcal{E}}$ as computed by some arbitrary Lipschitz function from the robot's sensors to $\mathbb{R}^p$. We provide details on the particular form of $\boldsymbol{\mathcal{E}}$ used in our empirical sections (Sec. 4.2-4.4). The feature mapping $\boldsymbol{\Phi}(\mathbf{x}, \mathbf{u}, \boldsymbol{\mathcal{E}}) : \mathbb{R}^{n+m+p} \mapsto \mathbb{R}^{n \times n_\theta}$ is learned in the offline training phase of our algorithm. Our method supports arbitrary parameterized families of continuous functions, but in practice we focus on the case where $\boldsymbol{\Phi}$ is a DNN. In this case, $\boldsymbol{\theta}$ can be regarded as the weights of the last layer of the DNN (2.2), which continuously adapt in real-time. The online adaptation is necessary in real scenarios, because two environments (i.e., terrains) might have the same representation $\boldsymbol{\mathcal{E}}$, but induce different dynamic behaviors onto the robot, as well as for other types of disturbances not captured in the feature mapping $\boldsymbol{\Phi}$.

Further, we assume that the disturbance $\mathbf{d}$ is affine in the control input $\mathbf{u}$, taking the form

$$\mathbf{d} \approx \boldsymbol{\Phi}^{\mathbf{w}}(\mathbf{x}, \boldsymbol{\mathcal{E}})\boldsymbol{\theta}\mathbf{u} = \sum_{i=1}^{n_\theta} \theta_i \boldsymbol{\Phi}_i^{\mathbf{w}}(\mathbf{x}, \boldsymbol{\mathcal{E}})\mathbf{u}, \tag{2.3}$$

where the dependence on a parameter vector $\mathbf{w}$ (the DNN weights) is made explicit, and the basis function has the form $\boldsymbol{\Phi}^{\mathbf{w}}(\mathbf{x}, \boldsymbol{\mathcal{E}}) : \mathbb{R}^{n+p} \mapsto \mathbb{R}^{n \times m \times n_\theta}$ with the individual matrix-valued components denoted by $\boldsymbol{\Phi}_i^{\mathbf{w}}(\mathbf{x}, \boldsymbol{\mathcal{E}}) : \mathbb{R}^{n+p} \mapsto \mathbb{R}^{n \times m}$. The control affine assumption is motivated by two factors. First, in our main application of ground vehicles with desired-velocity inputs, input-affine disturbances more accurately capture terrain interactions such as slippage (Sec. 3.3), internal dynamics, and wheel or track degradations. Second, this assumption simplifies the exponential convergence proof for our adaptive controller given in Theorem 1. However, we emphasize that Theorem 1 can be extended to more general forms of disturbances by input-output stability combined with contraction theory [77, 78]. Lastly, we define the dynamics residual that is used in both the online and offline phase of our method as $\mathbf{y} = \dot{\mathbf{x}} - \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$.

Offline Meta-Learning Phase

In Fig. 2.2, we illustrate the structure of our proposed solution to learn offline the basis function $\boldsymbol{\Phi}^{\mathbf{w}}(\mathbf{x}, \boldsymbol{\mathcal{E}})$ in (2.3) and to make real-time adjustments using composite adaptive control (Sec. 2.4). Our method is divided into two steps. First, the

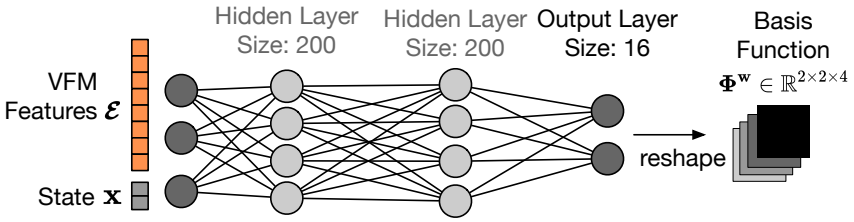


Figure 2.3: The structure of the DNN used for the basis function $\mathbf{\Phi}^{\mathbf{w}}$ in the controller synthesis from (3.14), (3.15) applied to a tracked vehicle.

robot captures relevant ground information during offline data collection, followed by training a DNN with terrain and state information to approximate the residual dynamics $\mathbf{y}$. Second, the trained model is deployed onboard the robot and updated online to compensate for the residual dynamics not captured in offline training.

Dataset

To learn the basis function $\mathbf{\Phi}^{\mathbf{w}}$ in (2.3), we collect a dataset of the robot operating on a diverse set of terrains. The dataset includes paired ground images from the onboard camera and state information measured using onboard sensors. The images are processed through a VFM, resulting in the representation $\mathcal{E}$, as discussed in Sec. 2.4.

This dataset contains $N \in \mathbb{N}$ trajectories. Each trajectory is an uninterrupted driving session on the order of a few minutes. Therefore, a single trajectory may contain significant dynamics-altering terrain transitions, such as between grass and concrete, but it will not contain dramatic transitions such as from a desert to a forest, or from midday to night. For notational simplicity only, we assume all trajectories have equal length $\ell \in \mathbb{N}$. Let $\mathbf{x}_t^n, \mathbf{u}_t^n, \mathcal{E}_t^n, \mathbf{y}_t^n$, respectively, denote the t^{th} state, input, VFM representation, and residual dynamics derivative of the n^{th} trajectory.

Model Architecture

In designing the parameterized function class for the basis function $\mathbf{\Phi}^{\mathbf{w}}$, we prioritize simplicity and efficiency to enable fast inference for real-time control. Therefore, we select a fully connected DNN with two hidden layers (Fig. 2.3), which takes as input both the robot's state and visual features from the VFM. We employ layer-wise spectral normalization to constrain the Lipschitz constant of the DNN. Spectral normalization is crucial for ensuring smooth control outputs and limiting pathological

Algorithm 1 Offline Meta-Learning with Continuous Trajectories

Input
  Dataset of N trajectories of length ℓ, window length distribution L, regularization target $\boldsymbol{\theta}_{\mathrm{r}}$ and weight λ_{r}, minibatch size K, initial DNN weights $\mathbf{w}$.
Output: Final optimized DNN weights $\mathbf{w}$ of $\boldsymbol{\Phi}$.
while not converged **do**
  Sample (with replacement) size-K minibatches of:
    - trajectory indices $\{n_k\}_{k=1}^K$,
    - window lengths $\{\hat{\ell}_k\}_{k=1}^K$,
    - start times $\{s_k\}_{k=1}^K$.
  Solve (2.4) for each $\theta^\star_{n_k,\hat{\ell}_k,s_k}$ in the minibatch
    (in closed form, allowing $J_{n_k,\hat{\ell}_k,s_k}$ gradient flow[1]).
  Take optimizer step on $\mathbf{w}$ w.r.t minibatch cost (2.4)

$$\sum_{k=1}^K J_{n_k,\hat{\ell}_k,s_k}(\mathbf{w}).$$

  Spectral Normalizaton: $\mathbf{W}_i \leftarrow \frac{\mathbf{W}_i}{\|\mathbf{W}_i\|}$ for all $i \in [d]$.
end while

behavior outside the training domain [79]. Details of spectral normalization are given in Sec. 2.4.

Optimization

Our method is built around the assumption that two terrains with similar visual features $\boldsymbol{\mathcal{E}}$ will usually, but not always, induce similar dynamics. We account for this observation with a meta-learning method that allows the linear part $\boldsymbol{\theta}$ to vary over the training data while the feature mapping weights $\mathbf{w}$ remain fixed. In particular, we assume that the linear part $\boldsymbol{\theta}$ is slowly time-varying within a single trajectory in the training data but $\boldsymbol{\theta}$ may change arbitrarily much between two trajectories. The slowly time-varying assumption implies that within a sufficiently short window into a full trajectory, $\boldsymbol{\theta}$ is approximately constant. Therefore, we optimize the weights $\mathbf{w}$ of the basis function for data-fitting accuracy when the best-fit constant $\boldsymbol{\theta}$ is computed for random short windows into the trajectories.

Due to our linear adaptation model structure, we observe that for a particular trajectory index $n \in [1 \dots N]$, window length $\hat{\ell} \in [1 \dots \ell]$, and starting timestep $s \in [1 \dots \ell - \hat{\ell} + 1]$, the best-fit value

$$\begin{aligned}\boldsymbol{\theta}^{\star}_{n,\hat{\ell},s}(\mathbf{w}) &=: \arg\min_{\boldsymbol{\theta}} \sum_{t=s}^{s+\hat{\ell}-1} \left\| \mathbf{y}_t^n - (\boldsymbol{\Phi}^{\mathbf{w}}\boldsymbol{\theta})\mathbf{u}_t^n \right\|_2^2 + \lambda_r \|\boldsymbol{\theta} - \boldsymbol{\theta}_r\|_2^2, \\ &= \sum_{t=s}^{s+\hat{\ell}-1} \left((\boldsymbol{\Phi}^{\mathbf{w}}\boldsymbol{\theta})^\top (\boldsymbol{\Phi}^{\mathbf{w}}\boldsymbol{\theta}) + \lambda_r \mathbf{I}_{n_\theta} \right)^{-1} \left((\boldsymbol{\Phi}^{\mathbf{w}}\boldsymbol{\theta})^\top \mathbf{y}_t^n + \lambda_r \boldsymbol{\theta}_r \right)\end{aligned}$$

is the solution to an L_2-regularized linear least-squares problem and is a closed-form, continuous function of the feature mapping parameter $\mathbf{w}$, where $\boldsymbol{\Phi}^{\mathbf{w}}$ is shorthand for $\boldsymbol{\Phi}^{\mathbf{w}}(\mathbf{x}_t^n, \boldsymbol{\mathcal{E}}_t^n)$, $\lambda_r \in \mathbb{R}$ is the regularization parameter, and $\boldsymbol{\theta}_r \in \mathbb{R}^{n_\theta}$ is the regularization target, chosen arbitrary. The $\|\boldsymbol{\theta} - \boldsymbol{\theta}_r\|_2^2$ regularization term ensures that the closed-form solution is unique. We can now define our overall optimization objective. Let L denote a distribution over trajectory window lengths: $L \in \triangle[1, \ell]$. We minimize the meta-objective

$$J(\mathbf{w}) = \mathop{\mathbb{E}}_{n,\hat{\ell},s} \left[\sum_{t=s}^{s+\hat{\ell}-1} \left\| \mathbf{y}_t^n - \left(\boldsymbol{\Phi}^{\mathbf{w}}(\mathbf{x}_t^n, \boldsymbol{\mathcal{E}}_t^n) \boldsymbol{\theta}^{\star}_{n,\hat{\ell},s}(\mathbf{w}) \right) \mathbf{u}_t^n \right\|_2^2 \right] \coloneqq \mathop{\mathbb{E}}_{n,\hat{\ell},s} \left[J_{n,\hat{\ell},s}(\mathbf{w}) \right],$$

where the expectation is shorthand for $n \sim \mathcal{U}[1, N]$, $\hat{\ell} \sim L$, and $s \sim \mathcal{U}[1, \ell - \hat{\ell} + 1]$. We incorporate the closed-form computation of the best-fit linear component $\boldsymbol{\theta}^{\star}_{n,\hat{\ell},s}$ in the computational graph of our optimization (meaning we take gradients through the least squares solution in Line 9), as opposed to treating the trajectories of $\boldsymbol{\theta}$ as optimization variables. Given this, we obtain a simpler algorithm.

Our offline training procedure is given in Algorithm 1. It consists of stochastic first-order optimization on the objective $J(\mathbf{w})$ and spectral normalization to enforce the Lipschitz constraint on $\boldsymbol{\Phi}$. In particular, let $\mathbf{w} = (\mathbf{W}_1, \dots, \mathbf{W}_d, \overline{\mathbf{w}})$, with d being the number of layers, where $\mathbf{W}_1, \dots, \mathbf{W}_d$ are the dimensionally compatible weight matrices of the DNN, such that the product $\mathbf{W}_1 \cdots \mathbf{W}_d$ exists, and $\overline{\mathbf{w}}$ are the remaining bias parameters. It holds that $\|\boldsymbol{\Phi}^{\mathbf{w}}\|_{\text{Lip}} \leq \|\mathbf{W}_1 \cdots \mathbf{W}_d\|$ for neural networks with 1-Lipschitz nonlinearities. Therefore, we can enforce that $\|\boldsymbol{\Phi}^{\mathbf{w}}\|_{\text{Lip}} \leq 1$, by enforcing that $\|\mathbf{W}_i\| \leq 1$ for all $i \in [1 \dots d]$. This is implemented in Algorithm 1 of Algorithm 1. Note that finding less conservative ways to enforce $\|\boldsymbol{\Phi}^{\mathbf{w}}\|_{\text{Lip}} \leq 1$ for DNN is an active area of research.

Algorithm 2 Rapid Terrain-Informed Online Adaptation for Model Mismatch and Tracking Error

1: **Input**
2: Optimized feature mapping $\mathbf{\Phi}$ from Algorithm 1, importance weight for prediction $\mathbf{R}$, damping constant λ, initial adaptive gain $\mathbf{\Gamma}_0$, reference trajectory $\mathbf{x}_r$.
3: Initialize $\hat{\boldsymbol{\theta}}$ with an user-defined regularization target $\boldsymbol{\theta}_\mathrm{r}$.
4: **while** running **do**
5: Evaluate the VFM on the input image and get the features $\boldsymbol{\mathcal{E}}$.
6: Get state $\mathbf{x}$ and evaluate the DNN $\mathbf{\Phi}(\mathbf{x}, \boldsymbol{\mathcal{E}})$.
7: Compute the tracking error vector $\mathbf{s}$ (e.g., using (3.13)).
8: Compute the control input $\mathbf{u}$ (e.g., using (3.14)).
9: Compute the dynamics residual $\mathbf{y}$ (e.g., using (3.16))
10: Compute the adaptation parameter derivative $\dot{\hat{\boldsymbol{\theta}}}$ $\dot{\hat{\boldsymbol{\theta}}} = -\lambda\hat{\boldsymbol{\theta}} - \mathrm{predict}(\mathbf{\Gamma}, \mathbf{y}, \mathbf{\Phi}, \hat{\boldsymbol{\theta}}, \mathbf{u}) + \mathrm{track}(\mathbf{\Gamma}, \mathbf{s}, \mathbf{\Phi}, \hat{\boldsymbol{\theta}}, \mathbf{u})$.
11: Compute the adaptation gain derivative $\dot{\mathbf{\Gamma}}$ (e.g., using (3.15) or (3.22)).
12: Integrate with system timestep Δ_t

$$\hat{\boldsymbol{\theta}} \leftarrow \hat{\boldsymbol{\theta}} + \Delta_t \dot{\hat{\boldsymbol{\theta}}}, \quad \mathbf{\Gamma} \leftarrow \mathbf{\Gamma} + \Delta_t \dot{\mathbf{\Gamma}}.$$

13: **end while**

For the remaining sections, the parameters $\mathbf{w}$ of the basis function $\mathbf{\Phi}$ are fixed. Therefore, we drop the superscript and refer to $\mathbf{\Phi}^{\mathbf{w}}$ as $\mathbf{\Phi}$, for simplicity of notation.

Online $\boldsymbol{\theta}$ Adaptation and Tracking Control

This section introduces the online adaptation running onboard the robot to adapt the linear part $\boldsymbol{\theta}$ of the dynamics model. The algorithm uses composite adaptation [80] and is given in Algorithm 2. The parameter vector $\boldsymbol{\theta}$ is initialized with a user-defined regularization target $\boldsymbol{\theta}_\mathrm{r}$ (Line 3). Then, in each cycle of the main loop, the robot processes the data from its visual sensor through the VFM to generate the feature vector $\boldsymbol{\mathcal{E}}$ (Line 5). The feature mapping $\mathbf{\Phi}$ is then evaluated using the robot's current state $\mathbf{x}$ and the feature vector $\boldsymbol{\mathcal{E}}$ (Line 6). We compute the error vector $\mathbf{s}$ between the reference trajectory $\mathbf{x}_r$ and the actual state $\mathbf{x}$, for example, using (3.13), as well as the control input $\mathbf{u}$ (Line 8).

These previously computed variables are passed to the composite adaptive controller. In this way, model mismatches and other disturbances not captured during training (Algorithm 1) can be adapted in real-time. For each sampled measurement

[1] We use the machine learning framework PyTorch that implements the linear least squares solution with gradient flow.

(interaction with the environment), the adaptation parameter vector $\hat{\boldsymbol{\theta}}$ is updated using the composite adaptation rule in Line (10), which is designed to decrease both the tracking error and the prediction error.

Each term of Line (10) provides a specific functionality: the first term in Line (10) implements the so-called "exponential forgetting" to allow $\boldsymbol{\theta}$ to change more rapidly when the best-fit parameters are time-varying. The second term is gradient descent on the $\mathbf{R}^{-1}$-weighted squared prediction error with respect to $\hat{\boldsymbol{\theta}}$, where $\mathbf{R}$ is a positive definite matrix. The third term minimizes the trajectory tracking error. In Line 11, we introduce $\boldsymbol{\Gamma}$, which is our adaptation gain, and its derivative $\dot{\boldsymbol{\Gamma}}$ can be defined from least-squares with exponential forgetting [64, 80] or reminiscent of a Kalman Filter like in [27]. Thus, this composite adaptation is used to ensure both small tracking errors and low model mismatch. The functions 'predict' and 'track' are defined in Sec. 3.3. The stability and robustness properties of Algorithm 2 are presented in Sec. 3.3.

BIBLIOGRAPHY

[1] Mario M. Foglia and Giulio Reina. “Agricultural Robot for Radicchio Harvesting”. In: *Journal of Field Robotics* 23.6-7 (July 2006), pp. 363–377.

[2] Pablo Gonzalez-De-Santos, Roemi Fernández, Delia Sepúlveda, Eduardo Navas, and Manuel Armada. “Unmanned Ground Vehicles for Smart Farms”. In: *Agronomy - Climate Change & Food Security*. London: Intechopen, 2020. Chap. 6, pp. 73–95.

[3] “Agricultural Robots for Field Operations: Concepts and Components”. In: *Biosystems Engineering* 149 (2016), pp. 94–111.

[4] Jeffrey Delmerico, Elias Mueggler, Julia Nitsch, and Davide Scaramuzza. “Active Autonomous Aerial Exploration for Ground Robot Path Planning”. In: *IEEE Robot. Autom. Letters* 2.2 (Apr. 2017), pp. 664–671.

[5] Zendai Kashino, Goldie Nejat, and Beno Benhabib. “Aerial Wilderness Search and Rescue with Ground Support”. In: *Journal Intelligent Robotic Syst.* 99.1 (July 2020), pp. 147–163.

[6] Hailong Qin et al. “Autonomous Exploration and Mapping System Using Heterogeneous UAVs and UGVs in GPS-denied Environments”. In: *IEEE Trans. Veh. Technol.* 68.2 (Feb. 2019), pp. 1339–1350.

[7] Abhijit Gadekar et al. “Rakshak: A Modular Unmanned Ground Vehicle for Surveillance and Logistics Operations”. In: *Cognitive Robotics* 3 (Mar. 2023), pp. 23–33.

[8] Vandi Verma et al. “Autonomous Robotics is Driving Perseverance Rover’s Progress on Mars”. In: *Science Robotics* 8.80 (July 2023).

[9] Raymond Arvidson et al. “Opportunity Mars Rover Mission: Overview and Selected Results from Purgatory Ripple to Traverses to Endeavour Crater”. In: *Journal Geophysical Research: Planets* 116.E7 (Feb. 2011).

[10] Mark Maimone, Yang Cheng, and Larry Matthies. “Two Years of Visual Odometry on the Mars Exploration Rovers”. In: *Journal of Field Robotics* 24.3 (Mar. 2007), pp. 169–186.

[11] Jo Yung Wong. *Theory of Ground Vehicles*. John Wiley & Sons, 2001.

[12] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. “Deep Reinforcement Learning in a Handful of Trials Using Probabilistic Dynamics Models”. In: *Proc. 32nd Int. Conf. Neural Information Processing Systems*. Dec. 2018, pp. 4759–4770.

[13] Matt Peng, Banghua Zhu, and Jiantao Jiao. “Linear Representation Meta-Reinforcement Learning for Instant Adaptation”. In: *Proc. Int. Conf. Learn. Representations*. 2021.

[14] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World". In: *IEEE/RSJ Int. Conf. Intelligent Robots Systems*. Sept. 2017, pp. 23–30.

[15] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. "Meta-learning with Memory-Augmented Neural Networks". In: *Proc. 33rd Int. Conf. Machine Learning*. Vol. 48. June 2016, pp. 1842–1850.

[16] Timothy M Hospedales, Antreas Antoniou, Paul Micaelli, and Amos J. Storkey. "Meta-Learning in Neural Networks: A Survey". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 44 (Sept. 2021), pp. 5149–5169.

[17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *Proc. 34th Int. Conf. Machine Learning*. Vol. 70. Aug. 2017, pp. 1126–1135.

[18] Anusha Nagabandi et al. "Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning". In: *Int. Conf. Learning Representations (ICLR)* (May 2019).

[19] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. "Model-based Reinforcement Learning via Meta-Policy Optimization". In: *Conf. on Robot Learning*. 2018, pp. 617–629.

[20] Ian Goodfellow et al. "Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems* 27 (2014), pp. 2672–2680.

[21] Yaroslav Ganin et al. "Domain-Adversarial Training of Neural Networks". In: *Journal of Machine Learning Research* 17.1 (May 2016), pp. 2096–2030.

[22] Christopher D. McKinnon and Angela P. Schoellig. "Meta Learning With Paired Forward and Inverse Models for Efficient Receding Horizon Control". In: *IEEE Robot. Autom. Letters* 6.2 (2021), pp. 3240–3247.

[23] Jean-Jacques E. Slotine and Weiping Li. *Applied Nonlinear Control*. Englewood Cliffs, N.J: Prentice Hall, 1991.

[24] Petros A Ioannou and Jing Sun. *Robust Adaptive Control*. Vol. 1. Prentice-Hall Upper Saddle River, NJ, 1996.

[25] Miroslav Krstic, Petar V Kokotovic, and Ioannis Kanellakopoulos. *Nonlinear and Adaptive Control Design*. John Wiley & Sons, Inc., 1995.

[26] Kumpati S Narendra and Anuradha M Annaswamy. *Stable Adaptive Systems*. Courier Corporation, 2012.

[27] Michael O'Connell et al. "Neural-Fly enables rapid learning for agile flight in strong winds". In: *Science Robotics* 7.66 (May 2022).

[28] Fu-Chuang Chen and Hassan K Khalil. “Adaptive Control of a Class of Nonlinear Discrete-Time Systems Using Neural Networks”. In: *IEEE Trans. Autom. Control* 40.5 (1995), pp. 791–801.

[29] Jay A. Farrell and Marios M. Polycarpou. *Adaptive Approximation Based Control*. John Wiley & Sons, Ltd, 2006.

[30] Jun Nakanishi, Jay A. Farrell, and Stefan Schaal. “A Locally Weighted Learning Composite Adaptive Controller with Structure Adaptation”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst.* Vol. 1. Sept. 2002, 882–889 vol.1.

[31] Jintasit Pravitra, Kasey A. Ackerman, Chengyu Cao, Naira Hovakimyan, and Evangelos A. Theodorou. “L1-Adaptive MPPI Architecture for Robust and Agile Control of Multirotors”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Sys. (IROS)*. Oct. 2020, pp. 7661–7666.

[32] Xingyou Song et al. “Rapidly Adaptable Legged Robots via Evolutionary Meta-Learning”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)* (Oct. 2020). arXiv: 2003.01239 [cs.RO].

[33] Suneel Belkhale et al. “Model-Based Meta-Reinforcement Learning for Flight With Suspended Payloads”. In: *IEEE Robot. Autom. Letters* 6.2 (Apr. 2021), pp. 1471–1478.

[34] Brandon Rothrock et al. “SPOC: Deep Learning-based Terrain Classification for Mars Rover Missions”. In: *AIAA SPACE 2016*. Amer. Inst. Aeronaut. Astronautics (AIAA) Forum, Sept. 2016.

[35] Jürgen Schmidhuber. “Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook”. Diploma Thesis. Technische Universität Munchen, Germany, May 1987.

[36] Guanya Shi, Kamyar Azizzadenesheli, Michael O’Connell, Soon-Jo Chung, and Yisong Yue. “Meta-Adaptive Nonlinear Control: Theory and Algorithms”. In: *Advances Neural Information Processing Systems* (2021).

[37] Wenli Xiao, Tairan He, John Dolan, and Guanya Shi. “Safe Deep Policy Adaptation”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 17286–17292.

[38] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. “Online Meta-Learning”. In: *Proc. 36th International Conference on Machine Learning*. June 2019, pp. 1920–1930.

[39] Maruan Al-Shedivat et al. “Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments”. In: *6th Int. Conf. on Learning Representations*. 2018.

[40] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. “Online Meta-Learning”. In: *Proc. 36th Int. Conf. on Machine Learning*. Vol. 97. June 2019, pp. 1920–1930.

[41] James Harrison, Apoorva Sharma, Chelsea Finn, and Marco Pavone. “Continuous Meta-Learning without Tasks”. In: *Advances in Neural Information Processing Syst.* Vol. 33. Curran Associates, Inc., 2020, pp. 17571–17581.

[42] James Harrison, Apoorva Sharma, Chelsea Finn, and Marco Pavone. “Continuous Meta-Learning without Tasks”. In: *arXiv:1912.08866* (2019). arXiv: 1912.08866.

[43] Bernard Espiau, François Chaumette, and Patrick Rives. “A New Approach to Visual Servoing in Robotics”. In: *IEEE Trans. Robot. Autom.* 8.3 (1992), pp. 313–326.

[44] David Ha and Jürgen Schmidhuber. “Recurrent World Models Facilitate Policy Evolution”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 2451–2463.

[45] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. “Learning Robust Perceptive Locomotion for Quadrupedal Robots in the Wild”. In: *Science Robotics* 7.62 (2022).

[46] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. “DayDreamer: World Models for Physical Robot Learning”. In: *Proc. 6th Conf. on Robot Learning*. Vol. 205. Dec. 2023, pp. 2226–2240.

[47] Xuxin Cheng, Kexin Shi, Ananye Agarwal, and Deepak Pathak. “Extreme Parkour with Legged Robots”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 11443–11450.

[48] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. “End-to-End Training of Deep Visuomotor Policies”. In: *Journal of Machine Learning Research* 17.39 (2016), pp. 1–40.

[49] Peter R. Florence, Lucas Manuelli, and Russ Tedrake. “Self-Supervised Correspondence in Visuomotor Policy Learning”. In: *IEEE Robot. Autom. Letters* 5 (2019), pp. 492–499.

[50] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex X. Lee, and Sergey Levine. “Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control”. In: *CoRR* abs/1812.00568 (2018).

[51] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A Physics Engine for Model-based Control”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst.* Oct. 2012, pp. 5026–5033.

[52] Jemin Hwangbo, Joonho Lee, and Marco Hutter. “Per-Contact Iteration Method for Solving Contact Dynamics”. In: *IEEE Robot. Autom. Letters* 3.2 (2018), pp. 895–902.

[53] Lu Gan, Jessy W. Grizzle, Ryan M. Eustice, and Maani Ghaffari. “Energy-Based Legged Robots Terrain Traversability Modeling via Deep Inverse Reinforcement Learning”. In: *IEEE Robot. Autom. Letters* 7.4 (Oct. 2022), pp. 8807–8814.

[54] Jonas Frey, Matias Mattamala, Nived Chebrolu, Cesar Cadena, Maurice Fallon, and Marco Hutter. "Fast Traversability Estimation for Wild Visual Navigation". In: *Proc. Robot.: Sci. Syst.* 2023. arXiv: 2305.08510 [cs.RO].

[55] Maxime Oquab et al. "DINOv2: Learning Robust Visual Features without Supervision". In: *Transactions on Machine Learning Research* (2024).

[56] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Syst.* Vol. 30. 2017, pp. 6000–6010.

[57] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021.

[58] Mark Hamilton, Zhoutong Zhang, Bharath Hariharan, Noah Snavely, and William T. Freeman. "Unsupervised Semantic Segmentation by Distilling Feature Correspondences". In: *Int. Conf. Learning Representations*. 2022.

[59] Gian Erni, Jonas Frey, Takahiro Miki, Matias Mattamala, and Marco Hutter. "MEM: Multi-Modal Elevation Mapping for Robotics and Learning". In: *2023 IEEE/RSJ International Conf. Intell. Robot. Syst. (IROS)*. 2023, pp. 11011–11018.

[60] Yanjie Ze et al. "GNFactor: Multi-Task Real Robot Learning with Generalizable Neural Feature Fields". In: *Proceedings of The 7th Conference on Robot Learning*. Vol. 229. PMLR, June 2023, pp. 284–301.

[61] Yixuan Wang et al. "D^3Fields: Dynamic 3D Descriptor Fields for Zero-Shot Generalizable Rearrangement". In: *8th Annual Conference on Robot Learning*. 2024.

[62] Muzammal Naseer et al. "Intriguing Properties of Vision Transformers". In: *Advances in Neural Information Processing Syst.* 2021.

[63] Anuradha M. Annaswamy and Alexander L. Fradkov. "A Historical Perspective of Adaptive Control and Learning". In: *Annu. Reviews Control* 52 (Oct. 2021), pp. 18–41.

[64] Xichen Shi, Patrick Spieler, Ellande Tang, Elena-Sorina Lupu, Phillip Tokumaru, and Soon-Jo Chung. "Adaptive Nonlinear Control of Fixed-Wing VTOL with Airflow Vector Sensing". In: *IEEE Int. Conf. Robot. Autom.* May 2020, pp. 5321–5327.

[65] Visakan Kadirkamanathan and Simon Fabri. "Stable Nonlinear Adaptive Control With Growing Radial Basis Function Networks". In: *5th IFAC Symp. on Adaptive Syst. Ctrl. Signal Processing* 28.13 (June 1995), pp. 245–250.

[66] Jean-Jacques E. Slotine and Weiping Li. "Composite Adaptive Control of Robot Manipulators". In: *Automatica* 25.4 (1989), pp. 509–519.

[67] Cuauhtémoc Acosta Lúa, Domenico Bianchi, and Stefano Di Gennaro. "Nonlinear Observer-Based Adaptive Control of Ground Vehicles with Uncertainty Estimation". In: *Journal the Franklin Inst.* 360.18 (2023), pp. 14175–14189.

[68] Ardashir Mohammadzadeh and Hamid Taghavifar. "A Novel Adaptive Control Approach for Path Tracking Control of Autonomous Vehicles Subject to Uncertain Dynamics". In: *Proc. Institution Mechanical Engineers, Part D: J. Automobile Engineering* 234.8 (2020), pp. 2115–2126.

[69] Mohsen Sombolestan, Yiyu Chen, and Quan Nguyen. "Adaptive Force-based Control for Legged Robots". In: *2021 IEEE/RSJ Intern. Conf. Intell. Robot. Syst. (IROS)*. Sept. 2021, pp. 7440–7447.

[70] K. Tsujita, K. Tsuchiya, and A. Onat. "Adaptive Gait Pattern Control of a Quadruped Locomotion Robot". In: *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.* Vol. 4. 2001, 2318–2325 vol.4.

[71] Plamen Petrov and Fawzi Nashashibi. "Modeling and Nonlinear Adaptive Control for Autonomous Vehicle Overtaking". In: *IEEE Trans. Intell. Transport. Syst.* 15.4 (2014), pp. 1643–1656.

[72] Subhan Khan and Jose Guivant. "Fast Nonlinear Model Predictive Planner and Control for an Unmanned Ground Vehicle in the Presence of Disturbances and Dynamic Obstacles". In: *Scientific Reports* 12.1 (July 2022), p. 12135.

[73] Marco Visca, Roger Powell, Yang Gao, and Saber Fallah. "Deep Meta-learning Energy-Aware Path Planner for Unmanned Ground Vehicles in Unknown Terrains". In: *IEEE Access* 10 (2022), pp. 30055–30068.

[74] Junwon Seo, Taekyung Kim, Seongyong Ahn, and Kiho Kwak. "META-Verse: Meta-Learning Traversability Cost Map for Off-Road Navigation". In: *arXiv:2307.13991* (July 2023).

[75] Xiaoyi Cai, Michael Everett, Lakshay Sharma, Philip Osteen, and Jonathan How. "Probabilistic Traversability Model for Risk-Aware Motion Planning in Off-Road Environments". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst.* (2023).

[76] Somrita Banerjee, James Harrison, P. Michael Furlong, and Marco Pavone. "Adaptive Meta-Learning for Identification of Rover-Terrain Dynamics". In: *CoRR* abs/2009.10191 (2020). arXiv: 2009.10191.

[77] Soon-Jo Chung, Saptarshi Bandyopadhyay, Insu Chang, and Fred Y Hadaegh. "Phase Synchronization Control of Complex Networks of Lagrangian Systems on Adaptive Digraphs". In: *Automatica* 49.5 (2013), pp. 1148–1161.

[78] Hiroyasu Tsukamoto, Soon-Jo Chung, and Jean-Jaques E. Slotine. "Contraction Theory for Nonlinear Stability Analysis and Learning-based Control: A Tutorial Overview". In: *Annu. Reviews Control* 52 (Oct. 2021), pp. 135–169.

[79] Guanya Shi et al. "Neural Lander: Stable Drone Landing Control Using Learned Dynamics". In: *Int. Conf. on Robot. Autom.* Mar. 2019, pp. 9784–9790.

[80] Jean-Jacques Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice Hall, 1991.

Chapter 3

ANALYSIS AND PROOFS OF MAGIC$^{\text{VFM}}$ FOR GROUND ROBOTS

3.1 Chapter Overview

In this chapter, we apply the methods from Algorithm 1 and 2 to a skid-steering tracked vehicle (Fig. 3.1) (Sec. 3.2 through 3.3) and to an Ackermann-steering vehicle (Sec. 3.4 through 3.5). The tracked vehicle uses skid-steering to maneuver over the ground, with its tracks moving at different speeds depending on the sprocket's angular velocity. Due to the slip between the sprocket and the tracks and between the tracks and the ground, modeling the full dynamics becomes very complex. We therefore derive its 3 DOF dynamics model (3.6) with its corresponding simplified model of the form (3.7). To this simplified model, we apply an adaptive controller with learned ground information of the form (3.14) and (3.15). The car-like vehicle uses the Ackermann steering geometry, which ensures that all wheels turn around the same center, thus minimizing wheel wear. For this vehicle type, we derive a 3-DOF dynamics model (3.24) using the bicycle model and an adaptive controller using Algorithm 2.

3.2 Tracked Vehicle Dynamics Model

We define a fixed reference frame $\mathcal{I}$ and a moving reference frame $\mathcal{B}$ attached to the body of the tracked vehicle, as seen in Fig. 3.1.

Consider the 3-DOF dynamics model with the generalized coordinates $\mathbf{q} := [p_x^{\mathcal{I}}, p_y^{\mathcal{I}}, \psi] \in \mathbb{R}^3$, where $p_x^{\mathcal{I}}$ and $p_y^{\mathcal{I}}$ are the inertial positions and ψ is the yaw angle from $\mathcal{I}$ to $\mathcal{B}$, as follows

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{B}(\mathbf{q})\boldsymbol{\tau}_u + \mathbf{F}_r(\mathbf{q}, \dot{\mathbf{q}}), \tag{3.1}$$

where $\mathbf{M} \in \mathbb{R}^{3\times3}$ is the inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{3\times3}$ is the Coriolis and centripetal matrix, $\mathbf{B}(\mathbf{q}) \in \mathbb{R}^{3\times2}$ is the control actuation matrix, $\mathbf{F}_r(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^3$ are the dissipative track forces due to surface-to-soil interaction, and $\boldsymbol{\tau}_u \in \mathbb{R}^2$ is the control torque.

Developing a tracking controller for the system modeled using (3.1) is difficult due to underactuation. To address this complexity, previous work [1, 2] introduced a nonholonomic constraint for (3.1), which reduces the number of state variables.

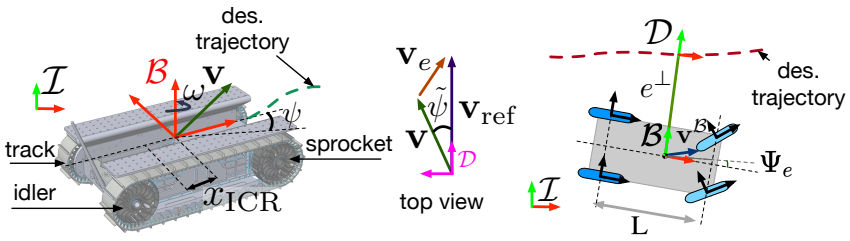


Figure 3.1: The frames of reference for the tracked vehicle, its corresponding velocities, and the main driving components (left), a velocity vector diagram used for the proof of Theorem 2 (middle), and the car model notations (right). For both vehicles, we assume the center of mass and the body frame are at the same location.

The following constrains the ratio of the lateral body velocity $\dot{p}_y^{\mathcal{B}}$ to the angular velocity ω

$$\dot{p}_y^{\mathcal{B}} + x_{\text{ICR}}\omega = 0, \tag{3.2}$$

where x_{ICR} is the ICR and $\omega = \dot{\psi}$. We embed this constraint into (3.1), as follows

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{B}(\mathbf{q})\boldsymbol{\tau}_u + \mathbf{F}_r(\mathbf{q}, \mathbf{q}) + \mathbf{A}(\mathbf{q})^\top \lambda_c, \tag{3.3}$$

with λ_c being the Lagrange multiplier corresponding to the equality constraint in (3.2). By assuming x_{ICR} as constant, $\mathbf{A}(\mathbf{q}) \in \mathbb{R}^{1\times 3}$ is defined as follows, in which $\mathbf{p}$ from (3.2) is expressed in the $\mathcal{I}$ frame

$$\begin{bmatrix} -\sin\psi & \cos\psi & x_{\text{ICR}} \end{bmatrix} \cdot \begin{bmatrix} \dot{p}_x^{\mathcal{I}} & \dot{p}_y^{\mathcal{I}} & \omega \end{bmatrix} = \mathbf{A}(\mathbf{q})\dot{\mathbf{q}} = 0. \tag{3.4}$$

To remove the constraint force from (3.3), an orthogonal projection operator $\mathbf{S}(\mathbf{q}) \in \mathbb{R}^{3\times 2}$ is defined, whose columns are in the nullspace of $\mathbf{A}^\top(\mathbf{q})$, and thus $\mathbf{S}(\mathbf{q})^\top \mathbf{A}(\mathbf{q})^\top = \mathbf{0}$ [3, 4].

$$\mathbf{S}(\mathbf{q}) = \begin{bmatrix} \cos(\psi) & x_{\text{ICR}}\sin(\psi) \\ \sin(\psi) & -x_{\text{ICR}}\cos(\psi) \\ 0 & 1 \end{bmatrix}. \tag{3.5}$$

We select this projection operator conveniently to transform the velocities in the $\mathcal{I}$ frame to $\mathbf{v} = [v_x^{\mathcal{B}}, \omega]^\top$, with $v_x^{\mathcal{B}}$ being the projection of the inertial velocity onto the body x-forward axis. The reduced form can be written as [5]

$$\begin{aligned} \dot{\mathbf{q}}(t) &= \mathbf{S}(\mathbf{q})\mathbf{v}(t), \\ \dot{\mathbf{v}}(t) &= \widetilde{\mathbf{M}}^{-1}(\widetilde{\mathbf{B}}(\mathbf{q})\boldsymbol{\tau}_u - \widetilde{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{v}(t) + \widetilde{\mathbf{F}}_r(\mathbf{q}, \dot{\mathbf{q}})), \end{aligned} \tag{3.6}$$

with the reduced matrices

$$\widetilde{\mathbf{M}} = \mathbf{S}^\top(\mathbf{q})\mathbf{M}\mathbf{S}(\mathbf{q}) = \begin{bmatrix} m & 0 \\ 0 & I_z + m x_{\mathrm{ICR}}^2 \end{bmatrix},$$
$$\widetilde{\mathbf{F}}_r = \mathbf{S}^\top(\mathbf{q})\mathbf{F}_r, \quad \widetilde{\mathbf{B}}(\mathbf{q}) = \mathbf{S}^\top(\mathbf{q})\mathbf{B}(\mathbf{q}),$$
$$\widetilde{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{S}^\top\mathbf{M}\dot{\mathbf{S}} = \begin{bmatrix} 0 & m x_{\mathrm{ICR}}\omega \\ -m x_{\mathrm{ICR}}\omega & 0 \end{bmatrix},$$

where m is the mass of the robot and I_z is the inertia of the robot about the rotational degree of freedom.

Simplified Vehicle Dynamics Model with Velocity Input

Due to the limited access to the robot's internal control software, specifically the absence of direct torque command capabilities, we are only able to utilize velocity inputs. Consequently, we have chosen to simplify the system in (3.6) with the velocity modeled as a first-order time delay

$$\dot{\mathbf{q}}(t) = \mathbf{S}(\mathbf{q})\mathbf{v}(t), \quad \dot{\mathbf{v}}(t) = \mathbf{A}_{\mathrm{n}}\mathbf{v}(t) + \mathbf{B}_{\mathrm{n}}\mathbf{u}(t), \tag{3.7}$$

where $\mathbf{u} = [u_v, u_\omega]$ are velocity inputs and

$$\mathbf{A}_{\mathrm{n}} = \begin{bmatrix} -\frac{1}{\tau_v} & 0 \\ 0 & -\frac{1}{\tau_\omega} \end{bmatrix}, \quad \mathbf{B}_{\mathrm{n}} = \begin{bmatrix} \frac{k_1}{\tau_v} & 0 \\ 0 & \frac{k_2}{\tau_\omega} \end{bmatrix}. \tag{3.8}$$

The simplified system is dynamically equivalent to (3.6). We identify the process gains k_1, k_2 and the process time constants τ_v, τ_ω using system identification on hardware. The robot is symmetric and rotates around the origin, therefore x_{ICR} is assumed 0.

3.3 Adaptive Tracking Controller for a Tracked Vehicle

First, we explain why using a control matrix adaptation is suitable for adapting to longitudinal and rotational slips, as well as the internal dynamics of a tracked vehicle. Next, we design a composite adaptive controller for the system in (3.7) and prove its exponential convergence to a bounded error ball. Note that our adaptive controller can be applied to any system of the form (3.1).

Motivation for Control Matrix Adaptation

The longitudinal slip κ is defined [6] as

$$\kappa = -\frac{v_x^{\mathcal{B}} - \Omega_{\mathrm{tr}} r_{\mathrm{tr}}}{v_x^{\mathcal{B}}}, \tag{3.9}$$

where Ω_{tr} is the angular velocity of the tracks, r_{tr} is the track wheel radius, and $v_x^{\mathcal{B}}$ is the projection of the inertial velocity onto the body x-forward axis. Let $\Omega_{\text{tr}} r_{\text{tr}}$ be our velocity control input u_v. Then (3.9) can be written as $v_x^{\mathcal{B}} = \frac{1}{1+\kappa} u_v$. Analyzing the extreme cases, we notice that if $\kappa = 0$ (no longitudinal slip), the velocity of the vehicle will match the velocity input into the tracks. In comparison, if $\kappa \to \infty$, the forward velocity of the vehicle will tend toward zero. Similar reasoning can be applied to the rotational slip. Therefore, adapting for a coefficient that multiplies the control input (the track speeds) ensures tracking of the body's forward and angular velocity.

In addition, adapting the control matrix also contributes to compensating for the unknown internal dynamics of the robot, because the velocity control input is the setpoint to an internal proportional-derivative-integral controller, which outputs motor torques to the tracked vehicle. Lastly, adapting the control matrix effectively compensates for track degradation, manifested as a slowdown in the sprocket's angular velocity.

Reference Trajectories

We define a 2 dimensional *feasible* trajectory characterized by the desired position and velocity $\mathbf{p}_d^{\mathcal{I}}(t)$, $\mathbf{v}_d^{\mathcal{I}}(t)$ in the inertial frame $\mathcal{I}$. The position error is $\tilde{\mathbf{p}}^{\mathcal{I}} = \mathbf{p}^{\mathcal{I}} - \mathbf{p}_d^{\mathcal{I}}(t)$, where $\mathbf{p}^{\mathcal{I}} = [p_x^{\mathcal{I}}, p_y^{\mathcal{I}}]$, and ψ_{d} is the desired yaw angle. Let the following reference velocities be

$$\mathbf{v}_{\text{ref}}^{\mathcal{I}} = \mathbf{v}_d^{\mathcal{I}} - \mathbf{K}_p \tilde{\mathbf{p}}^{\mathcal{I}}, \quad v_{\text{ref,x}} = \begin{bmatrix} \cos(\psi) \\ \sin(\psi) \end{bmatrix} \cdot \mathbf{v}_{\text{ref}}^{\mathcal{I}}, \tag{3.10}$$

$$\omega_{\text{ref}} = \dot{\psi}_{\text{ref}} - k_\psi (\psi - \psi_{\text{ref}}), \tag{3.11}$$

where the reference angle is given as

$$\psi_{\text{ref}} = \begin{cases} \arctan\left(\frac{\mathbf{v}_{\text{ref,y}}^{\mathcal{I}}}{\mathbf{v}_{\text{ref,x}}^{\mathcal{I}}}\right), & \text{if } \|\mathbf{v}_{\text{ref}}^{\mathcal{I}}\|_2^2 > v_\epsilon \\ \psi_{\text{d}}, & \text{otherwise.} \end{cases} \tag{3.12}$$

Note that the reference trajectory is not fully pre-planned; it includes feedback terms that are only defined during the execution of the trajectory. Both $\mathbf{K}_p \in \mathbb{R}^{2\times 2}$ and $k_\psi \in \mathbb{R}$ are positive gains, with $\mathbf{K}_p = \text{diag}(k_{px}, k_{py})$, and v_ϵ is a small velocity constant used to ensure the robot can track time-varying position trajectories, as well as turn in place. We define $\mathbf{v}_{\text{ref}} = [v_{\text{ref,x}}, \omega_{\text{ref}}]^\top$, which is our reference trajectory further used in the control synthesis.

Controller Synthesis

We design a composite adaptive controller $\mathbf{u}(t)$ and show that this composite tracking and adaptation error exponentially converges to a bounded error ball. First, we start by defining the tracking error variable $\mathbf{s}$ as

$$\mathbf{s} = \mathbf{v} - \mathbf{v}_{\text{ref}} = \begin{bmatrix} v_x^{\mathcal{B}} - v_{\text{ref,x}}, & \omega - \omega_{\text{ref}} \end{bmatrix}^\top. \tag{3.13}$$

We then derive the tracking controller for the system in (3.7)

$$\mathbf{u} = -(\mathbf{B}_\text{n} + \boldsymbol{\Phi}\hat{\boldsymbol{\theta}})^{-1}[\mathbf{K}\mathbf{s} + \mathbf{A}_\text{n}\mathbf{v}_{\text{ref}} - \dot{\mathbf{v}}_{\text{ref}}], \tag{3.14}$$

where $\mathbf{K} \in \mathbb{R}^{2\times 2}$ is a positive gain matrix, with $\mathbf{K} = \text{diag}(k_{dx}, k_{dw})$, $\boldsymbol{\Phi} \in \mathbb{R}^{2\times 2\times n_\theta}$ is the output of the DNN basis function (Fig. 2.3) evaluated with the feature vector $\boldsymbol{\mathcal{E}}$ and state $\mathbf{v}$, and $\hat{\boldsymbol{\theta}} \in \mathbb{R}^{n_\theta}$ is the estimated parameter vector of the true parameter vector $\boldsymbol{\theta}$. Recall from Sec. 2.4 that the learned basis function $\boldsymbol{\Phi}$ and the true adaptation parameters $\boldsymbol{\theta}$ were introduced to model the disturbance $\mathbf{d} \approx (\boldsymbol{\Phi}\boldsymbol{\theta})\mathbf{u} = \sum_{i=1}^{n_\theta} \theta_i \boldsymbol{\Phi}\mathbf{u}$. For our model of the skid-steer vehicle, we chose $n_\theta = 4$, matching the number of terms in our control matrix $\mathbf{B}_\text{n}$. Choosing n_θ too large can introduce redundant parameters and choosing n_θ too small could make the function class insufficiently expressive.

Theorem 1 *By applying the controller in* (3.14) *to the dynamics that evolve according to* (3.7)*, with the composite adaptation law, for each* $i \in [1, n_\theta]$

$$\begin{aligned} \dot{\hat{\theta}}_i &= -\lambda\hat{\theta}_i - \underbrace{\gamma_i \mathbf{u}^\top \boldsymbol{\Phi}_i^\top \mathbf{R}^{-1}\left(\sum_{i=1}^{n_\theta} \boldsymbol{\Phi}_i \mathbf{u}\hat{\theta}_i - \mathbf{y}\right)}_{\text{predict}} + \underbrace{\gamma_i \mathbf{s}^\top \boldsymbol{\Phi}_i \mathbf{u}}_{\text{track}}, \\ \dot{\gamma}_i &= -2\lambda\gamma_i + q_i + \gamma_i \mathbf{u}^\top \boldsymbol{\Phi}_i^\top \mathbf{R}^{-1} \boldsymbol{\Phi}_i \mathbf{u}\gamma_i, \end{aligned} \tag{3.15}$$

where $\gamma_i > 0$, $q_i > 0$, *for each* $i \in [1, n_\theta]$, *then the tracking errors* $\mathbf{s}$ *and the parameter error* $\tilde{\boldsymbol{\theta}}$ *will exponentially converge to a bounded error ball.*

Proof 1 *Defining the true control matrix as* $\mathbf{B} := \mathbf{B}_\mathrm{n} + \mathbf{\Phi}\boldsymbol{\theta}$*, we obtain the following closed loop system using* (3.7)

$$\dot{\mathbf{v}} = \mathbf{A}_\mathrm{n}\mathbf{v} - (\mathbf{B}_\mathrm{n} + \mathbf{\Phi}\boldsymbol{\theta})(\mathbf{B}_\mathrm{n} + \mathbf{\Phi}\hat{\boldsymbol{\theta}})^{-1}[\mathbf{K}\mathbf{s} + \mathbf{A}_\mathrm{n}\mathbf{v}_\mathrm{ref} - \dot{\mathbf{v}}_\mathrm{ref}] + \boldsymbol{\delta},$$

where $\boldsymbol{\delta}$ *is a representation error, previously introduced in* (2.2)*. Let* $\tilde{\boldsymbol{\theta}} = \hat{\boldsymbol{\theta}} - \boldsymbol{\theta}$ *be the error adaptation vector. Further, using the composite variable* $\mathbf{s}$ *in* (3.13)*, the closed-loop system becomes*

$$\mathbf{s} = \mathbf{A}_\mathrm{n}\mathbf{s} - \mathbf{K}\mathbf{s} - (\mathbf{\Phi}\tilde{\boldsymbol{\theta}})\mathbf{u} + \boldsymbol{\delta} = \mathbf{A}_\mathrm{n}\mathbf{s} - \mathbf{K}\mathbf{s} - \sum_{i=1}^{n_\theta} \mathbf{\Phi}_i\mathbf{u}\tilde{\theta}_i + \boldsymbol{\delta}.$$

For the prediction term in (3.15)*, we compute the dynamics residual derivative* $\mathbf{y}$ *determined for the bounded and adversarial noise* $\bar{\boldsymbol{\epsilon}}$ *as*

$$\mathbf{y} = \mathrm{LPF(s)}\dot{\mathbf{v}} - \mathbf{f}(\mathbf{v}, \mathbf{u}, t) = (\mathbf{\Phi}\boldsymbol{\theta})\mathbf{u} + \bar{\boldsymbol{\epsilon}}, \tag{3.16}$$

where premultiplying the noisy measurement $\dot{\mathbf{v}}$ *by* $\mathrm{LPF}(s)$ *with the Laplace transform variable* s *indicates low-pass filtering. Using the Lyapunov function*

$$\mathcal{V} = \mathbf{s}^\top\mathbf{s} + \sum_{i=1}^{n_\theta} \tilde{\theta}_i\gamma_i^{-1}\tilde{\theta}_i,$$

we compute its derivative as follows

$$\begin{aligned}
\dot{\mathcal{V}} &= 2\mathbf{s}^\top\dot{\mathbf{s}} + 2\sum_{i=1}^{n_\theta} \tilde{\theta}_i\gamma_i^{-1}\dot{\tilde{\theta}}_i + \sum_{i=1}^{n_\theta} \tilde{\theta}_i\frac{d}{dt}\left(\gamma_i^{-1}\right)\tilde{\theta}_i \\
&= -2\mathbf{s}^\top\left[(\mathbf{K} - \mathbf{A}_\mathrm{n})\mathbf{s} + \sum_{i=1}^{n_\theta} \mathbf{\Phi}_i\mathbf{u}\tilde{\theta}_i\right] \\
&+ 2\sum_{i=1}^{n_\theta} \gamma_i^{-1}\tilde{\theta}_i(\gamma_i\mathbf{s}^\top\mathbf{\Phi}_i\mathbf{u} - \gamma_i\mathbf{u}^\top\mathbf{\Phi}_i^\top\mathbf{R}^{-1}\sum_{j=1}^{n_\theta} \mathbf{\Phi}_j\mathbf{u}\tilde{\theta}_j - \lambda\tilde{\theta}_i) \\
&+ \sum_{i=1}^{n_\theta} \tilde{\theta}_i(2\gamma_i^{-1}\lambda - \gamma_i^{-1}q_i\gamma_i^{-1} - \mathbf{u}^\top\mathbf{\Phi}_i^\top\mathbf{R}^{-1}\mathbf{\Phi}_i\mathbf{u})\tilde{\theta}_i \\
&+ \underbrace{2\left(\mathbf{s}^\top\boldsymbol{\delta} + \sum_{i=1}^{n_\theta} \tilde{\theta}_i\left(\mathbf{u}^\top\mathbf{\Phi}_i^\top\mathbf{R}^{-1}\bar{\epsilon} - \gamma_i^{-1}\lambda\theta_i - \gamma_i^{-1}\dot{\theta}_i\right)\right)}_{\textit{error terms}}.
\end{aligned}$$

After further manipulation, the time derivative of the Lyapunov function becomes

$$\begin{aligned}
\dot{\mathcal{V}} &= -2\mathbf{s}^\top(\mathbf{K} - \mathbf{A}_\mathrm{n})\mathbf{s} - \sum_{i=1}^{n_\theta} \tilde{\theta}_i(q_i\gamma_i^{-2} + \mathbf{u}^\top\mathbf{\Phi}_i^\top\mathbf{R}^{-1}\mathbf{\Phi}_i\mathbf{u})\tilde{\theta}_i \\
&- 2\left(\sum_{i=1}^{n_\theta} \tilde{\theta}_i\mathbf{u}^\top\mathbf{\Phi}_i^\top\right)\mathbf{R}^{-1}\left(\sum_{j=1}^{n_\theta} \mathbf{\Phi}_j\mathbf{u}\tilde{\theta}_j\right) + \textit{error terms}.
\end{aligned} \tag{3.17}$$

Next, we will bound the terms in (3.17) as follows. There exists $\alpha \in \mathbb{R}_+$ such that

$$\begin{aligned} -2(\mathbf{K} - \mathbf{A}_{\mathrm{n}}) &\preceq -2\alpha \mathbf{I}, \\ -\left(q_i \gamma_i^{-2} + \mathbf{u}^\top \mathbf{\Phi}_i^\top \mathbf{R}^{-1} \mathbf{\Phi}_i \mathbf{u}\right) &\leq -2\alpha \gamma_i^{-1}, \quad \forall i \in [1, n_\theta]. \end{aligned} \tag{3.18}$$

We assume that $\|\boldsymbol{\delta}\|$, $\|\bar{\boldsymbol{\epsilon}}\|$, and $\dot{\theta}_i$ are small and bounded, and that the true value θ_i is bounded. Furthermore, the DNN $\mathbf{\Phi}_i$ is bounded since we use spectral normalization and the input domain is bounded. We then define an upper bound for the error terms as

$$\bar{d} = \sup_t \left(|\boldsymbol{\delta}| + \left| \sum_{i=1}^{n_\theta} \left(\mathbf{u}^\top \mathbf{\Phi}_i^\top \mathbf{R}^{-1} \bar{\boldsymbol{\epsilon}} - \gamma_i^{-1} \lambda \theta_i - \gamma_i^{-1} \dot{\theta}_i \right) \right| \right), \tag{3.19}$$

Note that this is a conservative estimate (the worst-case disturbance of all future time t), and hence can be made smaller using a shorter time range. Furthermore, even for a relatively large value of $\mathbf{\Phi}$, $\bar{d}$ can be made small using a larger value of $\mathbf{R}$ and a smaller value of $\bar{\boldsymbol{\epsilon}}$. We define the matrix $\mathcal{M}$, for $i \in [1, n_\theta]$

$$\mathcal{M} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathrm{diag}(\gamma_i^{-1}) \end{bmatrix}. \tag{3.20}$$

By applying the Comparison Lemma [7] and using a contraction theory like argument [8, 9], we can then prove the tracking error and adaptation parameters error exponentially converge to the bounded error ball

$$\lim_{t \to \infty} \| \begin{bmatrix} \mathbf{s} \\ \tilde{\theta}_i \end{bmatrix} \| \leq \frac{\bar{d}}{\alpha \lambda_{\min}(\mathcal{M})} := \bar{b}, \tag{3.21}$$

where $\lambda_{\min}$ is the minimum eigenvalue of a square matrix.

It follows from [7, 10] that the input-to-state stability (ISS) and bounded input and bounded output (BIBO) stability in the sense of finite-gain $\mathcal{L}_p$ [7] is proven for $\bar{b} \in \mathcal{L}_{pe}$, resulting in its bounded output $\mathbf{s}, \tilde{\boldsymbol{\theta}} \in \mathcal{L}_{pe}$, where the $\mathcal{L}_p$ norm in the extended space $\mathcal{L}_{pe}$, $p \in [1, \infty]$ is

$$\begin{aligned} \|(\mathbf{u})_\tau\|_{\mathcal{L}_p} &= \left(\int_0^\tau \|\mathbf{u}(t)\|^p dt \right)^{1/p} < \infty, \quad p \in [1, \infty) \\ \|(\mathbf{u})_\tau\|_{\mathcal{L}_\infty} &= \sup_{t \geq 0} \|(\mathbf{u}(t))_\tau\| < \infty \end{aligned}$$

and $(\mathbf{u}(t))_\tau$ is a truncation of $\mathbf{u}(t)$, i.e., $(\mathbf{u}(t))_\tau = \mathbf{0}$ for $t \geq \tau$, $\tau \in [0, \infty)$ while $(\mathbf{u}(t))_\tau = \mathbf{u}(t)$ for $0 \leq t \leq \tau$.

The exponential convergence proof in Theorem 1 shows that the online algorithm (Algorithm 2) will drive $\hat{\boldsymbol{\theta}}$ to a value within a bounded error ball of the offline least-squares solution used in the meta-learning algorithm (Algorithm 1) for a sufficiently long window of data. In contrast with [11, 12], (3.14) and (3.15) admits adaptation through the $\mathbf{B}$ control influence matrix. For stability purposes, under the assumption of a diagonal $\boldsymbol{\Gamma}$, the adaptation law equation resembles the Riccati equation of the $\mathcal{H}_\infty$ filtering [13]. This tends to increase the adaptation gain, making it more responsive to measurements.

The parameters of the adaptation law (3.15) are $\boldsymbol{\Gamma} = \text{diag}(\gamma_1, \ldots, \gamma_{n_\theta})$, $\mathbf{R}$, λ, and $\mathbf{Q} = \text{diag}(q_i)$. $\boldsymbol{\Gamma}$ is a positive definite matrix that influences the convergence rate of the estimator, and a sufficiently large initial $\boldsymbol{\Gamma}_0$ should be chosen to obtain a suitable convergence rate. $\mathbf{Q}$ is a positive definite gain added to the gain update law, λ is a damping factor, and $\mathbf{R}$ is a gain added to the prediction component of the adaptation law. Without this gain, the prediction term and the tracking error-based term could not be tuned separately.

Next, we assume the adaptation gain Γ in (3.15) has cross terms. Under this more general setting, we prove the exponential convergence of both $\tilde{\boldsymbol{\theta}}$ and $\mathbf{s}$ to a bounded error ball.

Proposition 1 *By applying the controller in* (3.14) *to the dynamics in* (3.7)*, with the composite adaptation law*

$$\dot{\hat{\boldsymbol{\theta}}} = -\lambda\hat{\boldsymbol{\theta}} - \underbrace{\boldsymbol{\Gamma}\mathbf{H}^\top\mathbf{R}^{-1}(\mathbf{H}\hat{\boldsymbol{\theta}} - \mathbf{y})}_{\text{predict}} + \underbrace{\boldsymbol{\Gamma}\mathbf{H}^\top\mathbf{s}}_{\text{track}}, \tag{3.22a}$$

$$\dot{\boldsymbol{\Gamma}} = -2\lambda\boldsymbol{\Gamma} + \mathbf{Q} - \boldsymbol{\Gamma}\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{H}\boldsymbol{\Gamma}, \tag{3.22b}$$

where $\lambda > 0$, $\boldsymbol{\Gamma} \in \mathbb{R}^{n_\theta \times n_\theta}$, $\mathbf{Q}^{n_\theta \times n_\theta}$ *and* $\mathbf{R} \in \mathbb{R}^{2\times 2}$ *are positive definite matrices and* $\mathbf{H} \in \mathbb{R}^{n\times n_\theta}$*, the tracking errors* $\mathbf{s}$ *and the parameter error* $\tilde{\boldsymbol{\theta}}$ *exponentially converge to a bounded error ball defined in [12].*

Proof 2 *We define the matrix* $\mathbf{H} = \begin{bmatrix}\mathbf{h}_1 \ldots \mathbf{h}_{n_\theta}\end{bmatrix} \in \mathbb{R}^{n\times n_\theta}$*, where the columns* $\mathbf{h}_i = \boldsymbol{\Phi}_i\mathbf{u}$*, for each* $i \in [1, n_\theta]$*. By observing that the disturbance in* (2.3) *can be defined as* $\mathbf{d} \approx \sum_{i=1}^{n_\theta} \boldsymbol{\Phi}_i\theta_i\mathbf{u} := \mathbf{H}\boldsymbol{\theta}$*, the proof of exponential convergence for additive disturbance adaptation from [12] can be directly applicable for the multiplicative disturbance adaptation.*

Note that the exponential convergence proof for Proposition 1 using Lyapunov theory holds for both when the last term of the gain adaptation law (3.22b) is positive and when the last term is negative. A negative sign makes the update law (3.22b) resemble the covariance update law of the Kalman filter. However, using a positive sign will make the closed-loop system converge faster for the same constants. Our controller in Theorem 1 behaves similar to the second case with the assumption that $\mathbf{\Gamma}$ is diagonal.

Lastly, for completeness, we show exponential convergence to a bounded error ball for the position and the attitude error.

Theorem 2 *By Theorem 1,* $\mathbf{s}$ *converges to a bounded error ball* (3.21) *defined as* $\bar{b}$. *Therefore, we hierarchically show that* $\psi \to \psi_{\text{ref}}$ *and* $\mathbf{p} \to \mathbf{p}_d$ *exponentially fast to a bounded error ball for bounded reference velocity.*

Proof 3 *We define the error* $\tilde{\psi} = \psi - \psi_{\text{ref}}$. *Using* (3.11), *we obtain* $\dot{\tilde{\psi}} + k_\psi \tilde{\psi} \leq \bar{b}$, *and with the Comparison Lemma, we prove that the error* $\tilde{\psi}$ *converges to the bounded error ball* $\frac{\bar{b}}{k_\psi}$. *To give intuition about the following position tracking error proof, we use Fig. 3.1. We define* $\mathbf{v} = \mathbf{v}_{\text{ref}} + \mathbf{v}_e$ *in vector form, where* $\mathbf{v}_e$ *is the velocity error. We further express these quantities in the reference frame* $\mathcal{D}$ *and note that, by Theorem 1, we have proved the convergence* $v_x^{\mathcal{B}} = v_{\text{ref,x}} + \bar{b}$ *as* $t \to \infty$. *Therefore, we obtain*

$$\begin{aligned}\mathbf{v}_e^{\mathcal{D}} &= -\begin{bmatrix} v_{\text{ref,x}}^{\mathcal{D}} \\ 0 \end{bmatrix} + \begin{bmatrix} \cos(\tilde{\psi}) \\ \sin(\tilde{\psi}) \end{bmatrix} \left(\begin{bmatrix} \cos(\tilde{\psi}) \\ \sin(\tilde{\psi}) \end{bmatrix} \cdot \begin{bmatrix} v_{\text{ref,x}}^{\mathcal{D}} \\ 0 \end{bmatrix} + \bar{b} \right) \\ &= v_{\text{ref,x}}^{\mathcal{D}} \begin{bmatrix} \cos^2(\tilde{\psi}) - 1 \\ \sin(\tilde{\psi}) \cos(\tilde{\psi}) \end{bmatrix} + \bar{b} \begin{bmatrix} \cos(\tilde{\psi}) \\ \sin(\tilde{\psi}) \end{bmatrix}.\end{aligned}$$

We compute and bound the norm, as follows

$$\|\mathbf{v}_e^{\mathcal{D}}\|_2 \leq v_{\max} \left| \sin\left(\frac{\bar{b}}{k_\psi} \right) \right| \sqrt{2} + \bar{b}, \tag{3.23}$$

where $v_{\max}$ *is our assumption for the existence of an upper bound for the reference velocity. From* (3.10) *and* (3.23), *it is straightforward to see that the position error is also bounded using the Comparison Lemma.*

3.4 Ackermann Steering Vehicle Dynamics Model

We define a fixed reference frame $\mathcal{I}$, a moving reference frame $\mathcal{B}$ attached to the center of mass of the car, and a desired frame $\mathcal{D}$ attached to the desired trajectory as seen in Fig. 3.1. Similar to (3.4), a non-holonomic constraint holds: $\begin{bmatrix}\sin\psi & -\cos\psi & 0\end{bmatrix} \cdot \begin{bmatrix}\dot{p}_x^{\mathcal{I}} & \dot{p}_y^{\mathcal{I}} & 0\end{bmatrix}$, where $\dot{p}_x^{\mathcal{I}}$ and $\dot{p}_y^{\mathcal{I}}$ are the velocities in the inertial frame $\mathcal{I}$ and ψ is the yaw angle from $\mathcal{B}$ to $\mathcal{I}$. For the tracked vehicle discussed in Sec. 3.2, the instantaneous center of rotation x_{ICR} is assumed to be 0 with $\dot{p}_y^{\mathcal{B}} = 0$ in (3.2) because the tracked vehicle is not designed for highly aggressive maneuvers.

A car, on the other hand, can be drifting, and thus the side velocity plays a much more important role, which is considered in our control design. Let $v_x^{\mathcal{B}}$ and $v_y^{\mathcal{B}}$ be the linear velocities in the body frame and ω the angular velocity around the vertical z-axis of the $\mathcal{B}$ frame. The dynamic model can be expressed as [14]

$$\begin{aligned} m(\dot{v}_x^{\mathcal{B}} - \omega v_y^{\mathcal{B}}) &= F_{xr} + F_{xf}\cos(u_\delta) - F_{yf}\sin(u_\delta), \\ m(\dot{v}_y^{\mathcal{B}} + \omega v_x^{\mathcal{B}}) &= F_{yr} + F_{xf}\sin(u_\delta) + F_{yf}\cos(u_\delta), \\ I_z\dot{\omega} &= \frac{L}{2}F_{xf}\sin(u_\delta) + \frac{L}{2}F_{yf}\cos(u_\delta) - \frac{L}{2}F_{yr}, \end{aligned} \tag{3.24}$$

where L is the wheelbase length, F_{xf} and F_{xr} are the front and rear tire forward forces, m is the vehicle mass, I_z is the vehicle inertia about the vertical axis intersecting the center of mass, and the lateral forces are $F_{yf} \approx C_y\alpha_f$, $F_{yr} \approx C_y\alpha_r$, where C_y is the tire cornering stiffness [15], and α_r and α_f are two tire slip angles, defined as in [14]. The tire cornering stiffness coefficient is terrain- and wheel-dependent. Either an accurate estimate or online adaptation is necessary when designing a tracking controller. Note that a more slippery ground has a lower C_y.

We decouple the controller for the longitudinal velocity from the controller for the lateral and angular velocity and apply our MAGIC$^{\text{VFM}}$ algorithm to the lateral and angular motion. Note that the forward velocity dynamics is nonlinear. Therefore, for simplicity, similar to the tracked vehicle, we model the forward velocity as a first-order time delay system $\dot{v}_x^{\mathcal{B}} = -\tau_v^{-1}(v_x^{\mathcal{B}} - u_v)$, with the time constant τ_v identified through system identification. We then design an exponentially stabilizing PD tracking controller for this linear system. For the lateral and angular motion, we linearize (3.24) around a zero steering angle and assume a small tire slip angle. The resulting linear time-varying system dynamics in the $\mathcal{B}$ frame is written by using $\mathbf{x} := [v_y^{\mathcal{B}}, \omega]$ and the disturbance model (2.3)

$$\dot{\mathbf{x}} = \mathbf{A}_{\text{n}}(v_x^{\mathcal{B}}(t))\mathbf{x} + \mathbf{B}_{\text{n}}u_\delta + (\boldsymbol{\Phi}(\mathbf{x}, \boldsymbol{\mathcal{E}})\boldsymbol{\theta})\, u_\delta, \tag{3.25}$$

where

$$\mathbf{A}_{\mathrm{n}}(v_x^{\mathcal{B}}(t)) = \begin{bmatrix} -\frac{2C_y}{mv_x^{\mathcal{B}}} & -v_x^{\mathcal{B}} \\ 0 & -\frac{L^2C_y}{2v_x^{\mathcal{B}}I_z} \end{bmatrix}, \mathbf{B}_{\mathrm{n}} = \begin{bmatrix} \frac{C_y}{m} \\ \frac{LC_y}{2I_z} \end{bmatrix},$$

and $\boldsymbol{\Phi}(\mathbf{x}, \mathcal{E}) = \begin{bmatrix} \boldsymbol{\phi}_1 & \boldsymbol{\phi}_2 \end{bmatrix}^\top$, with the estimated component $\hat{\boldsymbol{\theta}}$ adapted online. The definition of $\boldsymbol{\Phi}(\mathbf{x}, \mathcal{E}) \in \mathbb{R}^{2\times n_\theta}$ and $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$ are the same as their definitions for the tracked vehicle. The adaptation component accounts for model mismatches as well as for the linearization errors in (3.24).

3.5 Adaptive Tracking Controller for Ackermann Steering

We apply MAGIC$^{\mathrm{VFM}}$ to compensate for the sideslip when the robot is performing fast turning maneuvers. Thus, our adaptive control algorithm is applied only to the lateral and angular controller, although it can be applied for the linear velocity, as well. We define the path error $\mathbf{e} = [e^{\parallel}, e^{\perp}]$ with the longitudinal and lateral error components, as seen in Fig. 3.1

$$\mathbf{e} := \mathbf{p}^{\mathcal{D}} - \mathbf{p}_d^{\mathcal{D}} = \mathbf{R}_{\mathcal{I}}^{\mathcal{D}}(\mathbf{p}^{\mathcal{I}} - \mathbf{O}_{\mathcal{D}}^{\mathcal{I}}) \tag{3.26}$$

where $\mathbf{O}_{\mathcal{D}}^{\mathcal{I}}$ is the origin of the desired frame $\mathcal{D}$ expressed in $\mathcal{I}$, $\mathbf{p} = [p_x, p_y]$ is the position of the robot, $\mathbf{p}_d = [p_{x,d}, p_{y,d}]$ is the desired position from the trajectory, and $\mathbf{R}_{\mathcal{I}}^{\mathcal{D}}$ is the rotation from the inertial frame $\mathcal{I}$ to the desired frame $\mathcal{D}$. Next, we compute the time derivative of the path error (3.26) as follows

$$\begin{bmatrix} \dot{e}^{\parallel} \\ \dot{e}^{\perp} \end{bmatrix} = \mathbf{R}_{\mathcal{I}}^{\mathcal{D}}\mathbf{R}_{\mathcal{B}}^{\mathcal{I}}\mathbf{v}^{\mathcal{B}} - \mathbf{v}_d^{\mathcal{D}} + \dot{\mathbf{R}}_{\mathcal{I}}^{\mathcal{D}}(\mathbf{R}_{\mathcal{D}}^{\mathcal{I}}\mathbf{p}^{\mathcal{D}}), \tag{3.27}$$

where $\mathbf{R}_{\mathcal{B}}^{\mathcal{I}}$ is the rotation from the body frame $\mathcal{B}$ to the inertial frame $\mathcal{I}$, $\mathbf{v}_d^{\mathcal{D}} = [v_{d,x}^{\mathcal{D}}, 0]$ is the derivative of the desired position taken in $\mathcal{I}$, and expressed in $\mathcal{D}$, and $\mathbf{v}^{\mathcal{B}} = [v_x^{\mathcal{B}}, v_y^{\mathcal{B}}]$ is the velocity of the robot in the $\mathcal{B}$ frame. From (3.27), the perpendicular error derivative becomes

$$\dot{e}^{\perp} = \begin{bmatrix} \sin(\psi_e) \\ \cos(\psi_e) \end{bmatrix} \cdot \mathbf{v}^{\mathcal{B}} - \dot{\psi}_d e^{\parallel}, \tag{3.28}$$

where $\psi_e = \psi - \psi_d$ is the angle error between the actual orientation and the desired orientation. In addition, each component in (3.26) is

$$e^{\perp} = \begin{bmatrix} \sin(\psi_e) \\ \cos(\psi_e) \end{bmatrix} \cdot \tilde{\mathbf{p}}^{\mathcal{B}}, \quad e^{\parallel} = \begin{bmatrix} \cos(\psi_e) \\ -\sin(\psi_e) \end{bmatrix} \cdot \tilde{\mathbf{p}}^{\mathcal{B}}, \tag{3.29}$$

where $\tilde{\mathbf{p}}^{\mathcal{B}} = \mathbf{p}^{\mathcal{B}} - \mathbf{p}_d^{\mathcal{B}}$ is the position error expressed in $\mathcal{B}$. Because we model the dynamics decoupled and linearized, from (3.28), we obtain

$$\dot{e}^{\perp} = v_y^{\mathcal{B}} + v_x^{\mathcal{B}} \psi_e. \tag{3.30}$$

Further, we differentiate (3.30) and substitute (3.25), as follows

$$\ddot{e}^{\perp} = -\frac{2C_y}{m v_x^{\mathcal{B}}} v_y^{\mathcal{B}} - v_x^{\mathcal{B}} \omega + \frac{C_y}{m} u_\delta + (\boldsymbol{\phi}_1 \boldsymbol{\theta})\, u_\delta + \dot{v}_x^{\mathcal{B}} \psi_e + v_x^{\mathcal{B}} \omega_e. \tag{3.31}$$

Now we design a tracking controller for the lateral motion of the vehicle. Let $s^{\perp} = \dot{e}^{\perp} + k_p e^{\perp}$, with $k_p \in \mathbb{R}^+$ a positive constant. Then, using (3.31), $\dot{s}^{\perp}$ is

$$\dot{s}^{\perp} = -\frac{2C_y}{m v_x^{\mathcal{B}}} v_y^{\mathcal{B}} - v_x^{\mathcal{B}} \omega + \frac{C_y}{m} u_\delta + (\boldsymbol{\phi}_1 \boldsymbol{\theta})\, u_\delta + \dot{v}_x^{\mathcal{B}} \psi_e + v_x^{\mathcal{B}} \omega_e + k_p \dot{e}^{\perp}.$$

We then design the following adaptive controller

$$u_\delta = -\hat{b}_n^{-1} \left(k_v s^{\perp} - \frac{2C_y}{m v_x^{\mathcal{B}}} v_y^{\mathcal{B}} + \dot{v}_x^{\mathcal{B}} \psi_e - v_x^{\mathcal{B}} \omega_d + k_p \dot{e}^{\perp} \right), \tag{3.32}$$

where $\hat{b}_n = \frac{C_y}{m} + \boldsymbol{\phi}_1 \hat{\boldsymbol{\theta}}$. Letting $\tilde{\boldsymbol{\theta}} = \hat{\boldsymbol{\theta}} - \boldsymbol{\theta}$, the closed-loop system of $s^{\perp}$ becomes

$$\dot{s}^{\perp} + k_v s^{\perp} = \left(\boldsymbol{\phi}_1 \tilde{\boldsymbol{\theta}}\right) u_\delta. \tag{3.33}$$

Note that the controller in (3.32) resembles (3.14), which was derived for the tracked vehicle. Using the same proof as in Sec. 3.3, we show that the tracking error $s^{\perp}$ and $\tilde{\boldsymbol{\theta}}$ exponentially converge to a bounded error ball. Next, we analyze the stability of the internal states $v_y^{\mathcal{B}}(t)$ and $\psi_e(t)$ under the exact error definitions (3.28)-(3.29).

Theorem 3 *If $|s^{\perp}(t)| \leq e^{-\gamma t}|s_0^{\perp}| + \frac{\epsilon}{\gamma}$, for positive constants γ and ϵ, and s_0 being the initial value, under the local assumption of $-\frac{\pi}{2} < \psi_e < \frac{\pi}{2}$ and a positive $v_x^{\mathcal{B}}$, then $\tilde{p}_y^{\mathcal{B}}$, $v_y^{\mathcal{B}}$, and ψ_e exponentially tend to bounds.*

Proof 4 *Our forward velocity controller ensures $v_x^{\mathcal{B}}$ converges to the desired forward velocity, as shown in Theorem 1 for the tracked vehicle. Thus, we can approximate $\tilde{p}_x^{\mathcal{B}} \approx 0$ in (3.29). Hence, (3.28) and (3.29) are simplified as*

$$e^{\perp} \approx \cos\psi_e \tilde{p}_y^{\mathcal{B}}, \quad \dot{e}^{\perp} \approx \sin\psi_e (v_x^{\mathcal{B}} + \dot{\psi}_d \tilde{p}_y^{\mathcal{B}}) + \cos\psi_e v_y^{\mathcal{B}}. \tag{3.34}$$

Note that under no disturbance ($\epsilon \approx 0$) and since $s^{\perp} = \dot{e}^{\perp} + k_p e^{\perp}$, $e^{\perp}$ and $\dot{e}^{\perp}$ exponentially converge to 0. Assuming a feasible reference trajectory (nonzero desired side velocity, $v_{y,d}^{\mathcal{B}}$), since $\cos\psi_e$ and $v_x^{\mathcal{B}}$ are nonzero values, we have $\tilde{p}_y^{\mathcal{B}}$,

$\tilde{v}_y^{\mathcal{B}}$, *and* ψ_e *converge to 0. If* ϵ *is a small nonzero value, assuming* $\inf_t(\cos\psi_e) = \bar{\psi}$, *we can show that* $|\tilde{p}_y^{\mathcal{B}}|$ *exponentially converges to a small error bound, as follows*

$$\lim_{t\to\infty} |\tilde{p}_y^{\mathcal{B}}| \leq \frac{\epsilon}{\bar{\psi} k_p \gamma}. \tag{3.35}$$

We further assume that $|\tilde{p}_y^{\mathcal{B}}(t)| \approx e^{-\gamma_p t}|\tilde{p}_y^{\mathcal{B}}(0)| + \frac{|d(t)|}{k_p\gamma}$, *where* γ_p *is a positive constant and* $d(t)$ *is a function with a small Lipschitz constant* ϵ_y. *With this assumption, we can show* $v_y^{\mathcal{B}}$ *exponentially converges to the bound* $\frac{\epsilon_y}{k_p\gamma}$. *Then, assuming* $\inf_t v_x^{\mathcal{B}} = \bar{v}$, *with positive* $\bar{v}$, *we apply the triangle inequality for* $\dot{e}^{\perp}$ *as follows*

$$|\sin\psi_e(v_x^{\mathcal{B}} + \dot{\psi}_d\tilde{p}_y^{\mathcal{B}})| \leq |\dot{e}^{\perp}| + |\cos\psi_e v_y^{\mathcal{B}}|. \tag{3.36}$$

Taking the limit and denoting $\bar{v} - \frac{\epsilon \sup_t |\dot{\psi}_d|}{\bar{\psi} k_p \gamma}$ *as* $\underline{v}$, *we show that* ψ_e *exponentially converges to a bounded error as*

$$\lim_{t\to\infty} |\psi_e| \leq \arcsin\left(\frac{2\epsilon}{\underline{v}\gamma} + \frac{\bar{\psi}\epsilon_y}{\underline{v} k_p \gamma}\right). \tag{3.37}$$

Note that γ *and* k_p *can be chosen to make the error bounds sufficiently small. The proof above is based on* (3.28). *Using the simplified version of* (3.30), *the bound simplifies to* $\lim_{t\to\infty}|\psi_e| \leq \frac{2\epsilon}{\underline{v}\gamma} + \frac{\epsilon_y}{\underline{v}k_p\gamma}$.

BIBLIOGRAPHY

[1] Natalia Strawa, Dmitry I. Ignatyev, Argyrios C. Zolotas, and Antonios Tsourdos. "On-Line Learning and Updating Unmanned Tracked Vehicle Dynamics". In: *Electronics* 10.2 (Jan. 2021).

[2] Raymond H. Byrne and Chaouki Tanios Abdallah. "Design of a Model Reference Adaptive Controller for Vehicle Road Following". In: *Mathematical and Computer Modelling* 22.4 (1995), pp. 343–354.

[3] Michael Mistry and Ludovic Righetti. "Operational Space Control of Constrained and Underactuated Systems". In: *Proc. Robotics: Science Syst.* June 2011.

[4] Farhad Aghili. "A Unified Approach for Inverse and Direct Dynamics of Constrained Multibody Systems Based on Linear Projection Operator: Applications to Control and Simulation". In: *IEEE Transactions on Robotics* 21.5 (Oct. 2005), pp. 834–849.

[5] Luca Caracciolo, Alessandro De Luca, and Stefano Iannitti. "Trajectory Tracking Control of a Four-Wheel Differentially Driven Mobile Robot". In: *IEEE Int. Conf. Robot. Autom.* (May 1999).

[6] Hans Pacejka. *Tire and vehicle dynamics, 2nd Edition*. Elsevier, 2006.

[7] Hassan K. Khalil. *Nonlinear Systems, 3rd Edition*. Prentice Hall, 2002.

[8] Winfried Lohmiller and Jean-Jacques E. Slotine. "On Contraction Analysis for Non-linear Systems". In: *Automatica* 34.6 (1998), pp. 683–696.

[9] Hiroyasu Tsukamoto, Soon-Jo Chung, and Jean-Jaques E. Slotine. "Contraction Theory for Nonlinear Stability Analysis and Learning-based Control: A Tutorial Overview". In: *Annu. Reviews Control* 52 (Oct. 2021), pp. 135–169.

[10] Soon-Jo Chung, Saptarshi Bandyopadhyay, Insu Chang, and Fred Y Hadaegh. "Phase Synchronization Control of Complex Networks of Lagrangian Systems on Adaptive Digraphs". In: *Automatica* 49.5 (2013), pp. 1148–1161.

[11] Jean-Jacques Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice Hall, 1991.

[12] Michael O'Connell et al. "Neural-Fly enables rapid learning for agile flight in strong winds". In: *Science Robotics* 7.66 (May 2022).

[13] Dan Simon. "Optimal State Estimation". In: John Wiley & Sons, Inc., 2006. Chap. 11, pp. 331–371.

[14] Gabriel M. Hoffmann, Claire J. Tomlin, Michael Montemerlo, and Sebastian Thrun. "Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing". In: *Amer. Control Conference*. July 2007, pp. 2296–2301.

[15] Jo Yung Wong. *Theory of Ground Vehicles*. John Wiley & Sons, 2001.

Chapter 4

IMPLEMENTATION AND RESULTS OF MAGIC$^{\text{VFM}}$

4.1 Chapter Overview

In this chapter, we present the simulation and experimental results of our proposed MAGIC$^{\text{VFM}}$ framework. We first start by analyzing the features extracted by the Visual Foundation Model (VFM) to understand the suitability of the model for our application. We then present the simulation results in Section 4.3 to validate the learning and control strategy. We then discuss the hardware experiments in Section 4.4 to demonstrate the effectiveness of our adaptive controller on a tracked vehicle and a car-like vehicle. Moreover, we present the results of the experiments conducted on slopes in JPL's Mars Yard in Section 4.6 to validate the performance of our adaptive controller on different terrains. Lastly, we show the performance of our algorithm for the DARPA Learning Introspective Control (LINC) competition.

4.2 Empirical Results: Analysis of VFM Suitability

For our empirical work, we selected DINO V1 [1] as the VFM. DINO maps a high-resolution red-green-blue (RGB) image to a lower-resolution image where each pixel is a high-dimensional feature vector that depends on the *entire* input image, not just the corresponding input patch. More precisely, let $\xi \in \mathbb{N}$ be the patch dimension and $\xi_f \in \mathbb{N}$ the feature vector dimension. Given an RGB image $\mathcal{I}_{\text{RGB}}$, the transformation is

$$\mathcal{I}_{\text{RGB}} : h \times w \times 3 \rightarrow \mathcal{I}_{\text{VFM}} : \left\lfloor \frac{h}{\xi} \right\rfloor \times \left\lfloor \frac{w}{\xi} \right\rfloor \times \xi_f, \tag{4.1}$$

where h and w are the image height and width. We then extract prominent patches from $\mathcal{I}_{\text{VFM}}$ to form the terrain representation $\boldsymbol{\mathcal{E}}$, which will be further used in the $\boldsymbol{\Phi}$ from (2.3). For our experiments, we select a set of patches on right and left of the tracks/wheels of the vehicle, as emphasized in Fig. 2.1.

DINO is optimized for a self-supervised learning objective and was shown to yield feature mappings useful for a variety of downstream tasks. This VFM is trained on the ImageNet dataset, which also includes diverse ground terrains but mainly in the context of buildings, plants or landscapes, instead of terrain-only images. Therefore, in this section, we verify that DINO is able to clearly discriminate between different terrain types in terrain-only images before deploying it in our control setting.

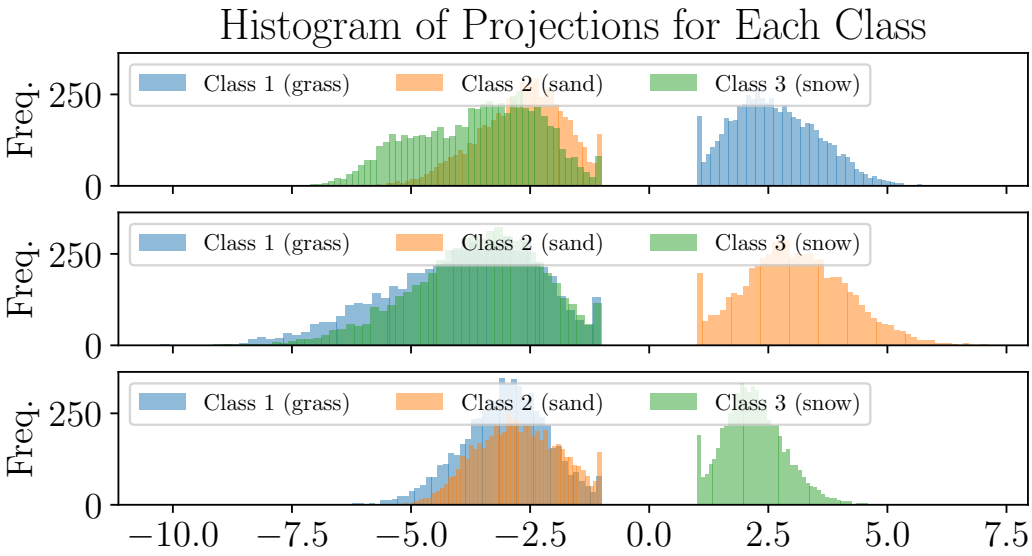


Figure 4.1: DINO VFM discriminative ability for different terrains. We show the histograms of the projection values onto the separating hyperplane normal computed using Support Vector Classifier for 3 sets of classes with 5 images each (each row presents the separation margin between one class type and the other 2 classes). Note that the spikes at -1 and 1 are an artifact of the high dimensionality and the small dataset we used.

We first measure DINO's discriminative ability by examining the margins of linear classifiers between terrain classes in the high-dimensional feature space $\mathbb{R}^{\xi_f}$. We consider three terrain types: grass, sand, and snow. We collect five example images for each class and convert each image to a set of feature vectors using DINO. Then, using the known class labels, we fit a multi-class linear classifier for the feature vectors using the One-vs-Rest Support Vector Classifier (OVR-SVC) method [2]. We then project the feature vectors onto the one-vs-rest separating hyperplane normals. Let the separating hyperplane have the equation $\mathbf{w}_h \cdot \mathbf{x} + b_h = 0$, where $\mathbf{w}_h \in \mathbb{R}^{\xi_f}$ is the vector normal to hyperplane and $\mathbf{b}_h \in \mathbb{R}$ is the bias term. Let $\boldsymbol{\mathcal{E}}$ be represented by just one patch, and thus have size ξ_f. The projected patch $\boldsymbol{\mathcal{E}}$ onto the separating hyperplane normal is defined as $p_h = \mathbf{w}_h \cdot \boldsymbol{\mathcal{E}}$. The histogram of these projected values for each patch in the image is shown in Fig. 4.1. By comparing the SVC margin (the separation between -1 and 1) to the width of the histograms, we confirm that the classes are highly separable.

We next examine the distribution of the features across a sequence of images, taken while navigating from flagstones (irregular-shaped flat rocks) to gravel in the Mars Yard [3] at NASA Jet Propulsion Laboratory (JPL). The top row of Fig. 4.2 displays

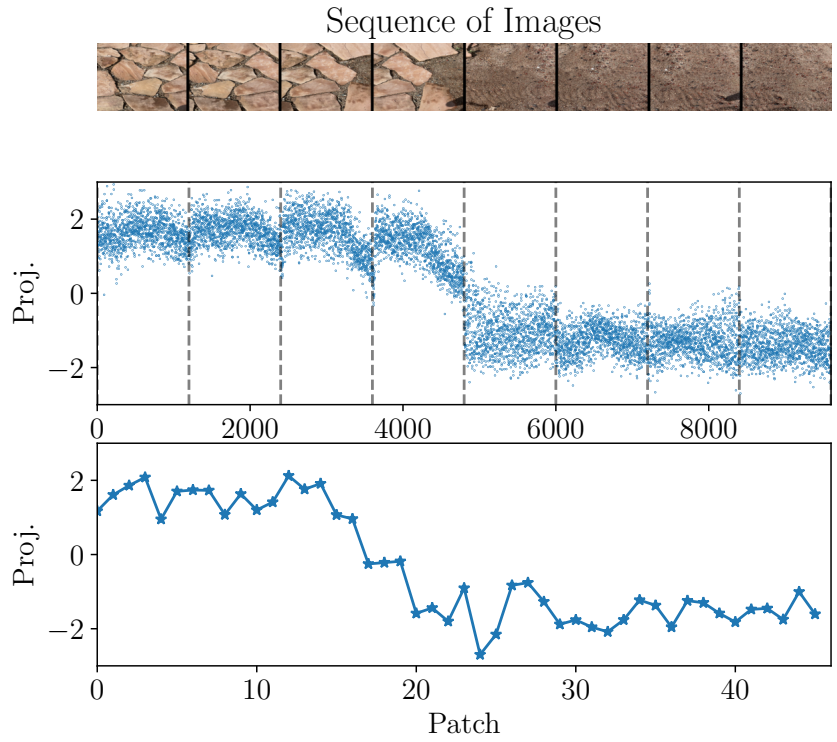
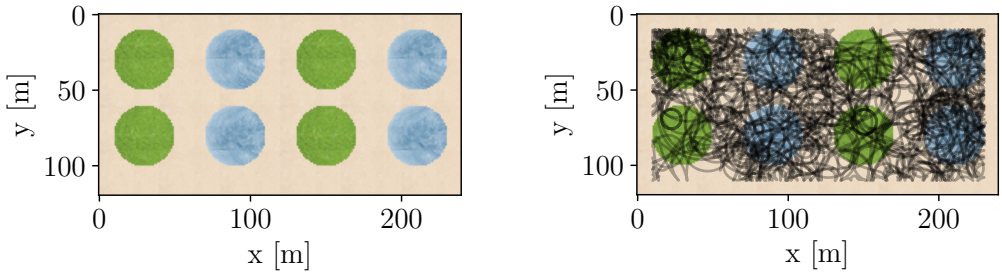


Figure 4.2: Projection of sequential flagstones and gravel features onto an OVR-SVC separating hyperplane normal. The middle plot shows the projection of all the patch features from the top 8 figures, while the bottom plot shows the projection of a central patch taken from 45 sequential images of flagstones and gravel.

8 out of a total of 45 images extracted from a video. Each image is processed through the DINO VFM, yielding 1200 patches of dimension $\xi_f = 384$ per image (computed using (4.2)). We apply OVR-SVC on the patches from one flagstone and one gravel image and project all patches from our chosen 8 images onto the SVC separating hyperplane normal. This projection reveals a bimodal distribution in the 3^{rd} and 4^{th} images due to the presence of both flagstones and gravel. In the bottom subplot of Fig. 4.2, we simulate a scenario where the robot traverses the area covered in all 45 images sequentially. For each image, we focus on a central patch of size ξ_f and project these features onto the separating hyperplane normal. This projection shows a consistent and continuous trend as the robot transitions from flagstone to gravel surfaces. This observation ensures the continuity of the VFM with respect to the camera motion.

Overall, these results provide positive empirical evidence that the DINO VFM is suitable for fine-grained discrimination of terrain types in images containing only terrain, and thus suitable for use in our setting.



(a) (b)

Figure 4.3: Simulation Environment (a) Environment with 3 different types of terrain (sand, grass, and ice), which represent areas of differing slip coefficients (b) Generated trajectories for training.

Table 4.1: Control and adaptation coefficients for the $\mathbf{\Phi}$ constant and $\mathbf{\Phi}$ DNN controllers for the simulation environment.

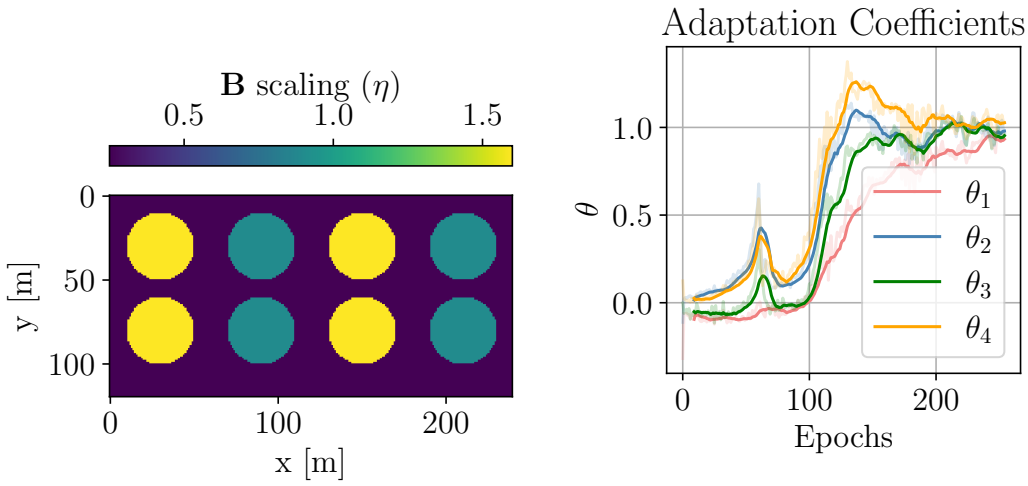
Ctrl. Type	k_{dx}	$k_{d\omega}$	$\mathbf{\Gamma}_0$ diag	$\mathbf{R}$ diag	$\mathbf{Q}$ diag	λ
$\mathbf{\Phi}$ = ct.	0.05	0.1	0.01	0.1	1.0	0.01
$\mathbf{\Phi}$ = NN	0.05	0.1	0.01	0.1	1.0	0.01

4.3 Simulation Results

Simulation Study Settings

To validate our learning and control strategy, we developed a simulation environment (Fig. 4.3) that enables detailed visualizations of the algorithm behavior. This environment is designed to incorporate a variety of terrains, thereby allowing us to test the adaptability of our model across diverse conditions without requiring extensive real world data collection. The dynamics for the simulator were modeled via (3.7), and the controller of (3.14) and (3.15) with the coefficients in Table 4.1 was used to track user-defined velocity trajectories $\mathbf{v}_{\text{ref}}$ generated at random. The environment contains three distinct terrain types (Fig. 4.3a). Each terrain type induces a different level of slip, modeled as a scaling of the nominal control matrix $\mathbf{B}_{\text{n}} \in \mathbb{R}^{2\times 2}$ in (3.7) such that $\mathbf{B}_{\text{n}}$ is replaced by $\eta\mathbf{B}_{\text{n}}$ and the dynamics matrix $\mathbf{A}_{\text{n}} \in \mathbb{R}^{2\times 2}$ is kept the same as in (3.7). The particular values of η are illustrated in Fig. 4.4a.

To construct a dataset, we simulate $N = 1$ long trajectory of 150 000 discrete time steps, with randomized piecewise-constant velocity inputs. For acquiring these features, we utilize the DINO VFM on images of the terrains from Fig. 4.3a. As explained in Sec. 4.2, this model processes high-resolution terrain images into a more compact, lower resolution embedding. This reduced resolution representation is overlaid across the entire map. Specifically, let $m_s \times n_s$ be the size of the simulated



(a) (b)

Figure 4.4: Simulation (a) Perturbed control matrix (η) on the different types of terrains. (b) Convergence of the adaptation coefficients for the simulation dataset.

map, which is 120×240 in our case. Let each DINO feature image have the size computed as in (4.2), for a background image of size 480×640, a patch size of $\xi = 16$, and the feature size $\xi_f = 384$.

$$\mathcal{I}_{\text{VFM}} : \left\lfloor \frac{480}{16} \right\rfloor \times \left\lfloor \frac{640}{16} \right\rfloor \times 384 = 30 \times 40 \times 384. \tag{4.2}$$

Then, we tile each of the DINO feature images across the entire map vertically 4 times $\left(\left\lfloor \frac{120}{30} \right\rfloor\right)$ and horizontally 6 times $\left(\left\lfloor \frac{240}{40} \right\rfloor\right)$ and extract and record the relevant terrain features underneath the robot. For training, we collect random trajectories, generated by sampling control inputs $\mathbf{u}$ from a uniform distribution, and integrate forward the dynamics in (3.7) in order to cover a large portion of the simulated map, as seen in Fig. 4.3b. The dataset contains the DINO features extracted from underneath the robot and the robot's velocities, and as labels the residual dynamics derivative $\mathbf{y}$, computed as in (3.16). Using this dataset, we then train the basis function $\mathbf{\Phi}$, whose architecture can be seen in Fig. 2.3, using Algorithm 1. The convergence of the adaptation coefficients to $\boldsymbol{\theta}_r$ is presented in Fig. 4.4b, taken as an average over the last training minibatch K. This compact representation of the terrain is then integrated with online adaptive control (Algorithm 2).

Simulation Study Objectives

In exploring the capabilities of our model, we investigate how prior knowledge of the terrain contributes to improved tracking accuracy for an adaptive controller. We

thus test our algorithm across 3 scenarios: a) We assess the model performance in an environment identical to the one used during training to understand its effectiveness with in-distribution data (Fig. 4.5). b) We test the algorithm under simulated nighttime conditions to gauge performance when the ground is identical, but the lighting conditions are different (Fig. 4.6). c) We challenge the model by presenting it with two environments that have similar visual features to those in the training data set, but exhibit different dynamic behaviors. Furthermore, we adopt an adversarial approach by exposing the robot to completely novel environments that are not encountered during training (Fig. 4.8).

In-distribution Performance

To quantify if prior knowledge of the terrain improves tracking accuracy, the robot is tested in-distribution using the same environment as in the training dataset. The first row of Fig. 4.5 shows 39 random trials (black) and the single exemplar path (shades of purple, colored by the $\mathcal{L}_2$ error between the actual and desired states). These random trials are used to compute error statistics in row 6. The second row displays the robot's adaptation coefficients for the purple trajectory as it navigates through this environment. When the basis function lacks terrain awareness and is set as constant matrices (4.3),

$$\mathbf{\Phi} = \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \right\}, \tag{4.3}$$

there is significant fluctuation in the adaptation coefficients during the transition between different terrains. Conversely, when the DNN basis function is used, the adaptation coefficients remain relatively stable, while the DNN output itself varies with each terrain type, as shown in the third row of Fig. 4.5. The fourth row showcases the components of the product between the DNN basis function $\mathbf{\Phi}$ and the adaptation vector $\hat{\boldsymbol{\theta}}$. Though the output of our controller is slightly noisier, the adaptation of the product $\mathbf{\Phi}\hat{\boldsymbol{\theta}}$ is significantly faster. The fifth row shows the normalized error between the robot's actual and desired states. For the constant basis function, most of the tracking error occurs at terrain transitions. When the basis function is terrain-informed, the error is negligible, even at terrain transitions. Finally, the last row shows the spread of the cumulative error across 40 distinct experimental runs, each initiated at a random starting point and orientation, but of the same duration (the black and the purple trajectories in Row 1). The results of the simulation show that our terrain-informed DNN-based tracking controller reduces

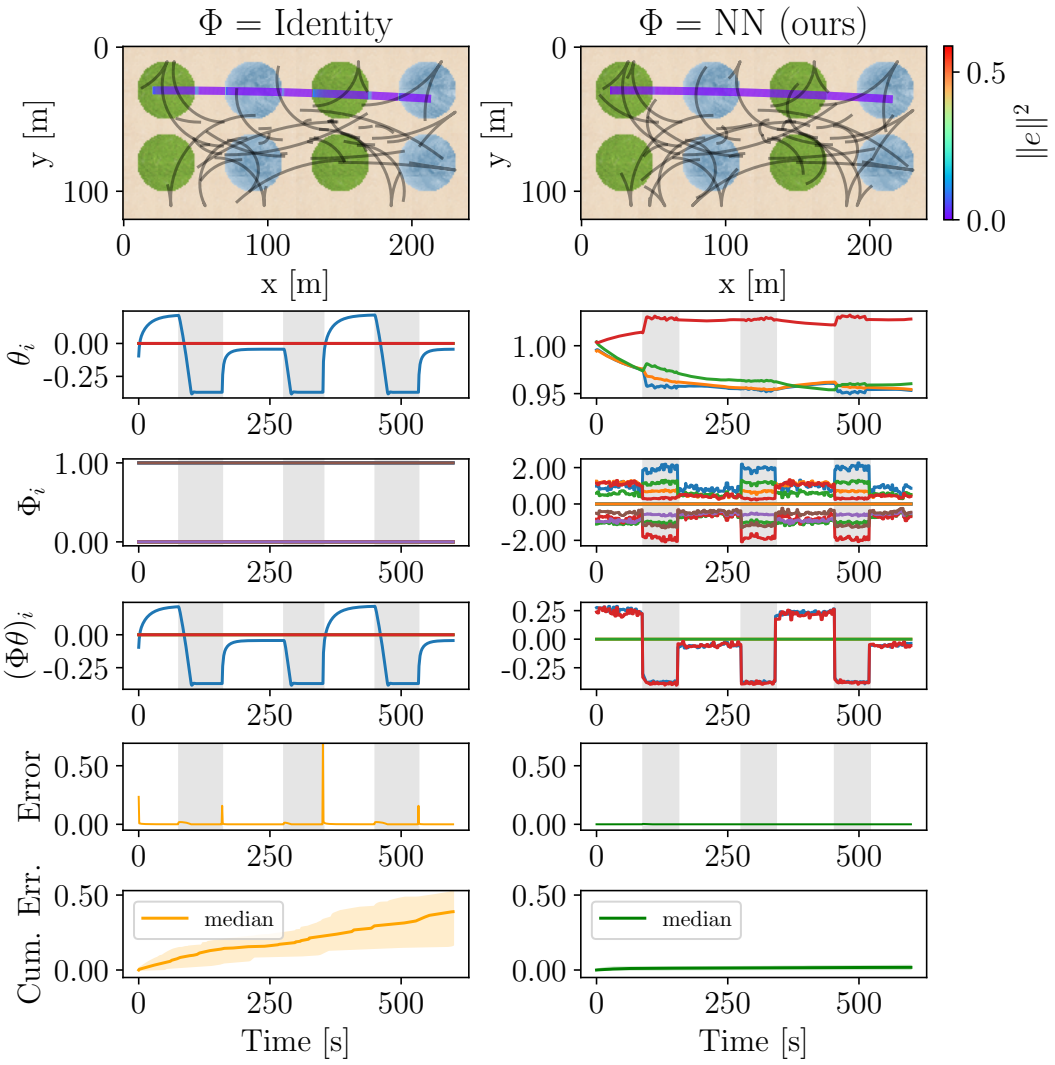


Figure 4.5: Results for in-distribution data from the simulation model. Each experiment was run 40 times on the terrains from row 1. The left column contains the performance for the baseline controller, while the right column contains the performance for our method. The second row contains the adaptation coefficients $\hat{\theta}_i$, while the third row emphasizes the basis function $\mathbf{\Phi}_i$. For the baseline, $\mathbf{\Phi}_i$ is constant, while in our method, $\mathbf{\Phi}_i$ varies as a function of the terrain and state. The last row presents the cumulative error, where the thick colored line represents the median, and the shaded region encompasses the range from the 25$^{\text{th}}$ to the 75$^{\text{th}}$ percentile.

the cumulative error by approximately 90.1% when compared to the constant $\mathbf{\Phi}$, defined as in (4.3).

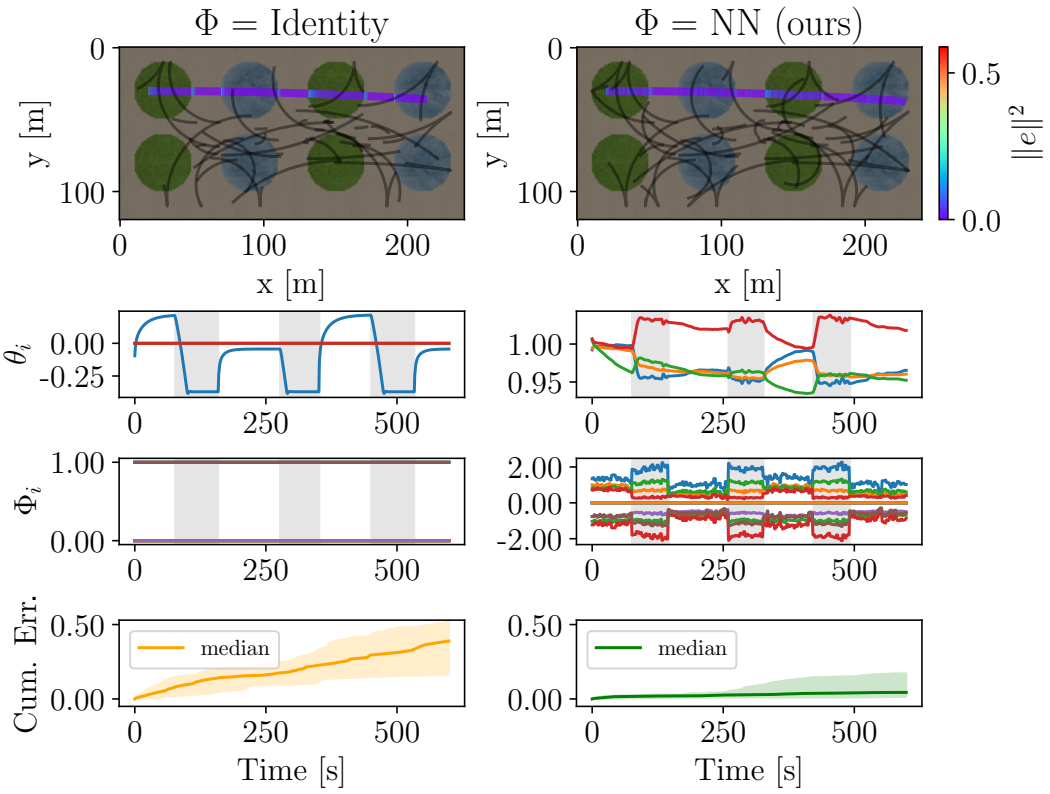


Figure 4.6: Simulation results in the simulated nighttime environment. Despite different lighting conditions, the cumulative error is kept small by the terrain-informed DNN.

Nighttime Out-of-distribution Performance

To test the robustness of our framework to varying lighting conditions, we extend our simulated experiments with a nighttime environment by uniformly darkening (changing the brightness) of each image representing the environment (Fig. 4.6). While the adaptation coefficients exhibit more variation compared to those in the standard, in-distribution scenario, the DNN still demonstrates good accuracy in predicting the environment from the darkened images. This outcome emphasizes the robustness of VFM, underscoring its ability to adapt effectively to varying lighting conditions. Importantly, even in these altered night conditions, the cumulative error remains low.

We highlight the importance of adaptation by comparing the tracking error under two scenarios: with adaptation and without adaptation, as shown in Fig. 4.7. In the "no adaptation case," we maintain $\boldsymbol{\theta}$ as a constant, initialized to $\boldsymbol{\theta}_r$. This comparison effectively demonstrates the benefits of adaptation, emphasizing its value even in situations where the basis function accurately predicts the terrain.

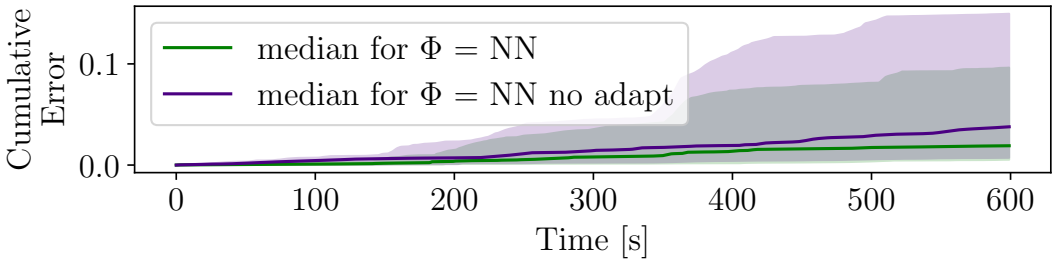


Figure 4.7: Comparison of the cumulative error between adaptation and "no adaptation" for the simulated nighttime experiment. In both cases, the DNN version of $\mathbf{\Phi}$ is used. This figure emphasizes the benefit of doing online adaptation.

Adversarial Environment Performance

In our final test (Fig. 4.8), we introduced two adversarial environments for the robot, manipulating two visually similar environments by altering their respective η coefficient of the $\mathbf{B}$ matrix. This emulates the real world where pits of deep sand appear very similar to shallow sand, but have a significantly different effect on the dynamics of the robot. Additionally, we modified the appearance of the simulated ice environment to create a distinct visual difference, while also slightly changing the effect of ice on the dynamics.

In the adversarial environment, the adaptation coefficients exhibit greater changes than for the in-distribution and night-time simulations. In addition, we observe that the DNN basis function demonstrates good performance, validating its effectiveness in handling out-of-distribution data. This effectiveness is likely attributed to the zero-shot capability inherent in the VFM. Lastly, it is important to note that the overall cumulative error remained lower compared to scenarios where the basis function lacked terrain information, further demonstrating the benefit and robustness of our approach in varied and challenging conditions, even for out-of-distribution data.

4.4 Overview Hardware Experiments

We focus on the hardware implementation and experimental validation of our MAGIC$^{\text{VFM}}$ adaptive controller discussed in Sec. 2.4-4.2 on a tracked vehicle whose dynamics are modeled in (3.7) and a car with Ackermann steering with the dynamics modeled as in (3.25). We present how our adaptive controller effectively addresses various perturbations such as terrain changes, severe track degradation, and unknown internal robot dynamics.

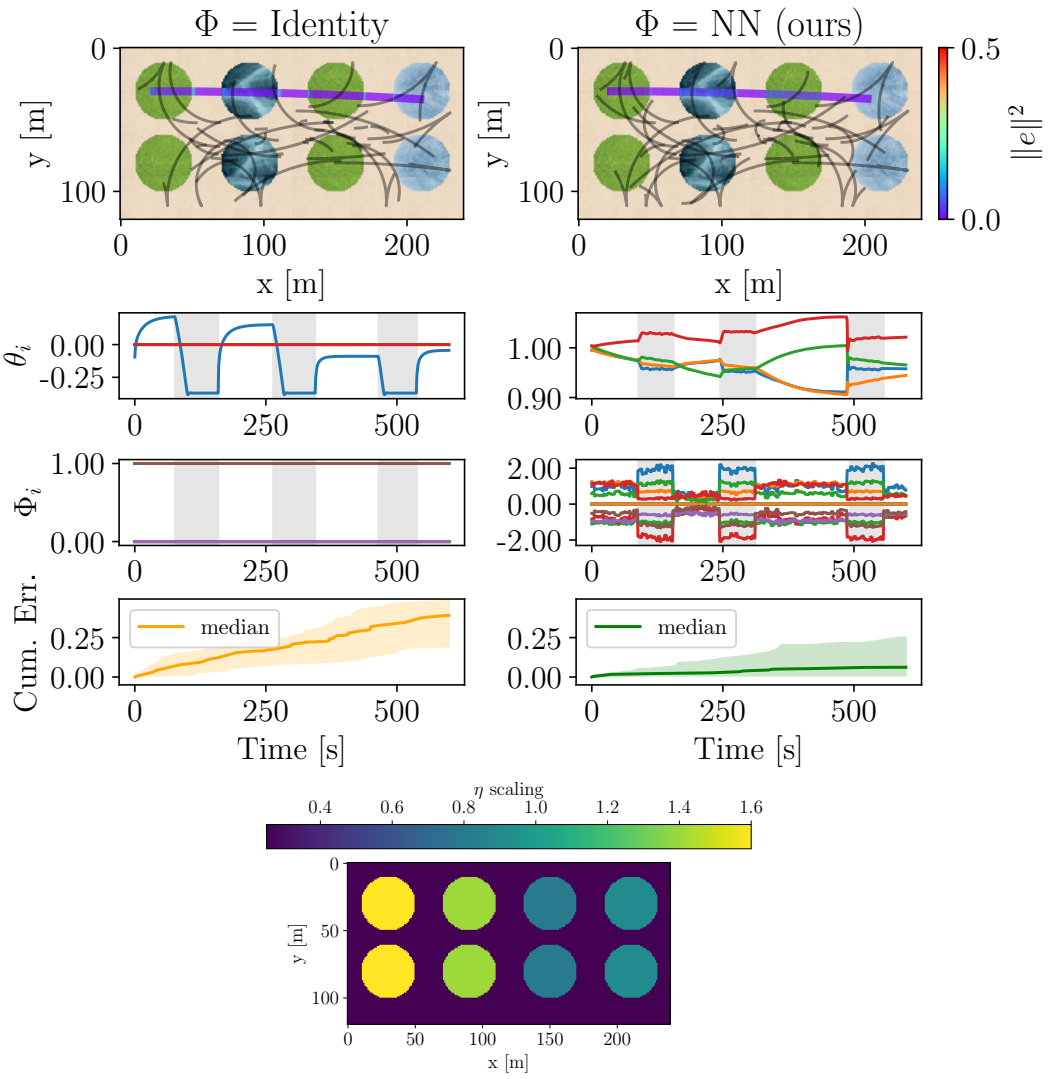


Figure 4.8: Simulation results with adversarial environment (different structures that look like ice) and different η scaling coefficient for the control matrix for identical terrain.

4.5 Robot Hardware and Software Stack

Experiments were carried out using a GVR-Bot [4] and a modified Traxxas X-Maxx, both shown in Fig. 4.9. Both vehicles are equipped with an NVIDIA Jetson Orin, RealSense D457 cameras (GVR-Bot: two forward facing and one rear facing, Traxxas: single forward facing) and a VectorNav VN100 IMU.

State estimation is provided onboard using OpenVINS [5], which fuses the camera data with an IMU to estimate the platform's position, attitude, and velocity. Our MAGIC$^{\text{VFM}}$ controller, as presented in (3.14), (3.15) for the tracked vehicle and in (3.32) for the car-like vehicle, and Theorem 1, runs at 20 Hz. It is implemented in

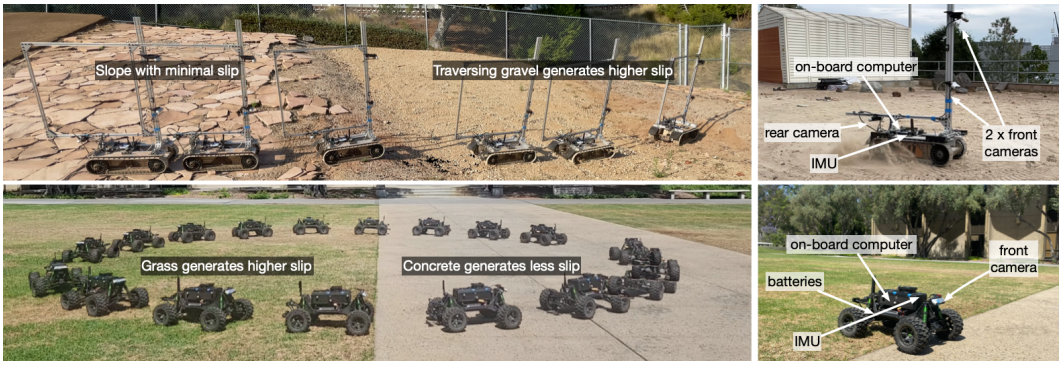


Figure 4.9: Setup for hardware experiments (a) The GVR-Bot traversing two slopes with different textures and terrain-induced dynamic behaviors at the JPL Mars Yard. (b) The GVR-Bot with the sensing and compute units highlighted. Note that the forward facing camera is used for state estimation, while the top camera is used for taking terrain images for MAGIC$^{\text{VFM}}$. The rear camera is not used in this work. (c) The Traxxas robot traversing two different terrains that induce different dynamic behaviors. (d) The Traxxas robot with its main sensing and compute units highlighted.

Python using the Robot Operating System (ROS) as the middleware to communicate with the robot's internal computer.

4.6 Experiments on Slopes in JPL's Mars Yard

The GVR-Bot only accepts velocity commands as the track velocities are regulated using an internal PID controller, which is inaccessible to the user. While this justifies our first-order modelling (3.7) using velocities, these experimental results validate that MAGIC$^{\text{VFM}}$ successfully learns the unknown internal dynamics. To verify the performance of our MAGIC$^{\text{VFM}}$ controller (Sec. 2.4 and 3.1) on different terrains, the GVR-Bot was driven on the slopes of the Mars Yard [3] at the Jet Propulsion Laboratory (JPL). Fig. 4.9 shows the two selected slopes, both chosen for their appropriate angle and visually different terrain type that induce different dynamic terrain-based behaviors.

Offline Training

Training data was collected by driving the GVR-Bot via direct tele-operation for a total of 20 minutes on the slopes. This trajectory was designed to include segments of transition between different slopes as well as periods of single slope operation. We utilize this dataset for training our terrain-dependent basis function as outlined in Algorithm 1. By leveraging the strengths of a pre-trained VFM, we develop the

Table 4.2: Training hyperparameters for Algorithm 1 for the tracked vehicle. β is the learning rate for the optimization in Line 11 of Algorithm 1, $\boldsymbol{\theta}_r$ is the regularization target, $\ell_{\min}$ and $\ell_{\max}$ are the bounds for the distribution over trajectory window lengths, λ_{r} is the regularization term for (2.4), K is the minibatch size, and n_θ is the size of the adaptation vector.

β	$\boldsymbol{\theta}_r$	$\ell_{\min}$	$\ell_{\max}$	λ_{r}	K	n_θ
0.001	$\mathbf{1}_4$	1.2 [s]	30 [s]	0.1	70	4

lightweight DNN basis function head used in the adaptive controller of (3.14), (3.15). This function processes inputs comprising of the mean of two visual feature patches from the GVR-Bot's right and left tracks and the robot's velocity taken from the onboard state estimator. The VFM-based DNN ($\boldsymbol{\Phi}$) structure incorporates two hidden layers, each consisting of 200 neurons, as seen in Fig. 2.3. The output has size 16, which is then reconfigured into dimensions $n \times m \times n_\theta$, where $n = 2$ is the state size, $m = 2$ is the control input size, and $n_\theta = 4$ is the size of the adaptation vector that matches the number of terms in the control matrix. The hyperparameters for the training algorithm are shown in Table 4.2.

Online Adaptation

At runtime, the downward-facing camera[1] is used to capture images of the terrain at 20 Hz. These images are then processed by the VFM explained in Sec. 4.2 to extract the features. The extracted features are then concatenated with the robot's velocity and are then fed into the DNN basis function $\boldsymbol{\Phi}$. This function, together with an online-adapting vector, is then employed to dynamically adjust the residual $\mathbf{B}$ matrix (3.14), (3.15) in real time to account for the different terrains.

The benefits of the terrain-informed basis function can been seen by comparing the performance of a constant and non-constant basis function controller as the robot traverses slopes. Both controllers are based on (3.14) and the adaptation law in (3.15). The first controller uses a constant basis function, defined in (4.3). We choose this structure for the constant $\boldsymbol{\Phi}$ to capture both the direct and cross-term effects on the robot's velocity. The second controller uses a terrain-dependent DNN basis function trained as explained in Sec. 4.6. The control coefficients for both controllers are presented in Table 4.3. The initial adaptation vector $\boldsymbol{\theta}_0 = \boldsymbol{\theta}(0)$ for

[1]To mitigate the purple tint in the RGB images (a common issue for Intel Realsense cameras), the RGB cameras were outfitted with neutral density filters to maintain the integrity of the VFM features.

Table 4.3: Control coefficients for both controllers ($\mathbf{\Phi}$ is constant and $\mathbf{\Phi}$ is a DNN) for the tracked vehicle.

k_{px}	k_{py}	k_{dx}	$k_{d\omega}$	k_{ψ}	$\mathbf{\Gamma}_0$ diag	$\mathbf{Q}$ diag	$\mathbf{R}$ diag	λ
0.8	0.8	0.5	1.6	2.3	0.2	0.1	5.0	0.01

Table 4.4: Statistics for the tracked vehicle on Mars Yard slopes.

Controller	Tracking error (RMS [m])
$\mathbf{\Phi}$ constant	0.130 ± 0.038
$\mathbf{\Phi}$ DNN (ours)	0.061 ± 0.022

the constant basis function is $\mathbf{0}_{n_\theta}$, while $\boldsymbol{\theta}(0)$ for the terrain-dependent basis function is the converged value from Algorithm 1.

Each experiment was carried out five times, with the results detailed in Fig. 4.10. For repeatability, we used a rake to re-distribute the gravel on the slopes between runs and alternated back and forth between running the two controllers. For this experiment, the desired trajectory is a straight line that spans the entire length of the two slopes (see Fig. 4.9).

Fig. 4.10 shows that when the robot traverses the first slope (flagstone resulting in minimal slippage), both controllers have comparable tracking errors. However, a notable change in performance appears when the robot transitions to the second slope, which has an increased tendency for the soil to slump down the hill, causing slippage. In Table 4.4, we present the RMSE between the actual position and the desired position computed as $\sqrt{\frac{1}{L}\sum_{i=1}^{L}||\mathbf{p}_i^I - \mathbf{p}_{di}^I||_2^2}$, where L is the length of the trajectory. The results demonstrate that the integration of a VFM in an adaptive control framework enhances tracking performance, yielding an average improvement of 53%.

Computational Load

A significant bottleneck in deploying VFMs onboard robots is the computational requirements of the inference stage of the models, especially as typical controllers need to run at 10s-100s Hz. To minimize the inference time and allow high controller rates, we employ the smallest visual transformer architecture of the DINO V1, consisting of 21 million network parameters. This architecture allows us to run the controller at 20 Hz on the Graphics Processing Unit (GPU) on-board an NVIDIA Jetson Orin.

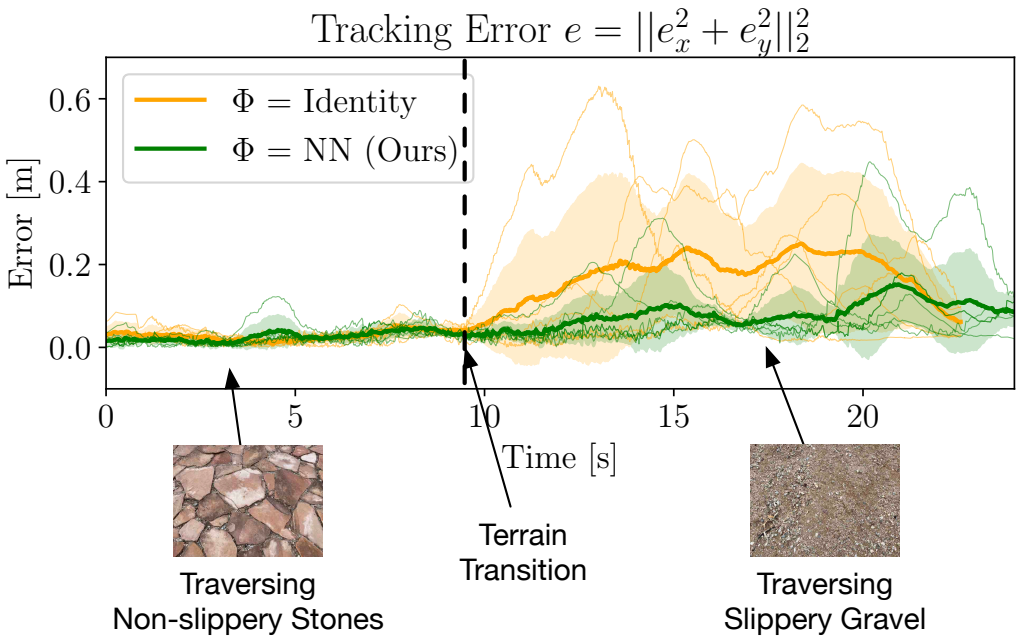


Figure 4.10: Tracking error for the two controllers (constant basis function and terrain-dependent basis function) on the slopes for a tracked vehicle. The error is computed as the Euclidean distance between actual and desired positions in the $\mathcal{I}$ frame. For both colors, the thick line outlines the mean of the 5 experiments, the shaded area represents 1 standard deviation, and the thin and transparent lines denote the 5 experiments.

4.7 Experiments On-board an Ackermann Steering Vehicle

We performed similar experiments to those described in Sec. 4.6 using an Ackermann steering vehicle. Here, the robot traverses two different terrains, as seen in Fig. 4.9, which induce different dynamic behaviors onto the robot (grass is more slippery than concrete). Our experiments on both vehicles showed that, on flat ground, the car experiences more significant slippage and terrain disturbances compared to the tracked vehicle. Therefore, for this experiment, we validated MAGIC$^{\text{VFM}}$ on flat ground.

In Fig. 4.11, we show the product $\mathbf{\Phi}\hat{\boldsymbol{\theta}}$ for the constant basis function of the nonlinear tracking controller in (3.32). As the robot transitions between the two terrains, we see that the robot effectively adapts to each terrain during this transition. This behavior mirrors that observed in the simulation plots (Fig. 4.5). Note that we maintained $n_\theta = 4$, to be consistent with the DNN model of the basis function, even though all four parameters are identical in this instance.

In Fig. 4.12, we emphasize the adaptation coefficients (left) and the DNN basis

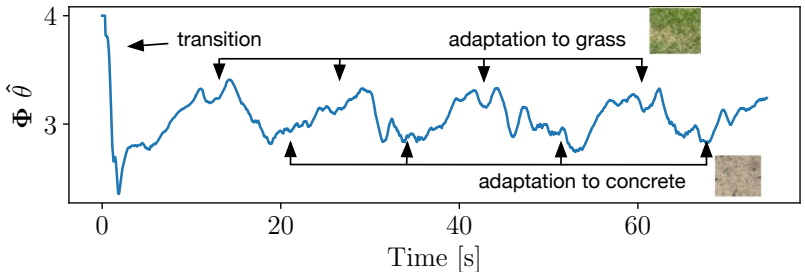


Figure 4.11: $\mathbf{\Phi}\hat{\boldsymbol{\theta}}$ of (3.32) for the constant basis function baseline for the car-like robot. Different terrains induce convergence to different adaptation coefficients.

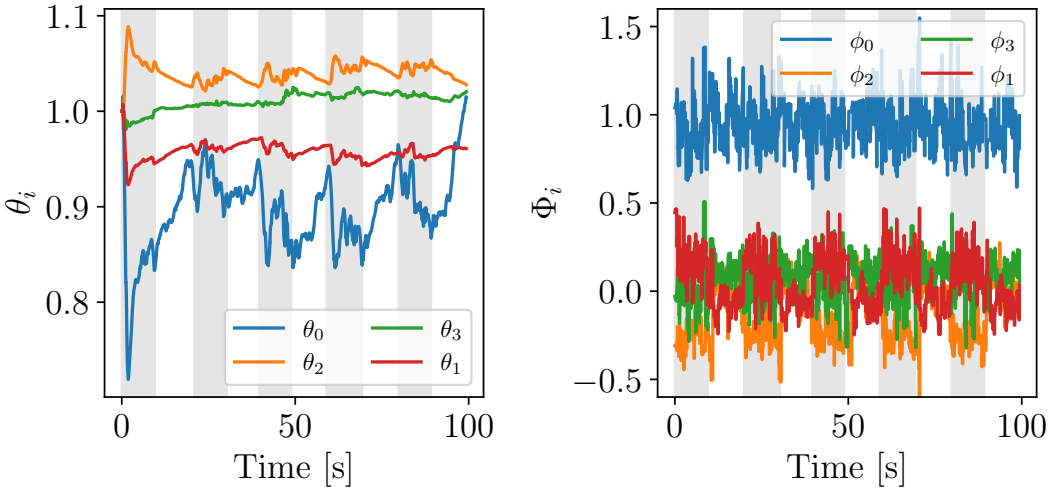


Figure 4.12: Adaptation coefficients $\hat{\boldsymbol{\theta}}$ and terrain-dependent DNN basis function for a car-like robot. Approximate under-vehicle terrain is denoted with the white (concrete) and gray (grass) bars.

function output (right) for the nonlinear tracking controller in (3.32) as the robot transitions between the two terrains (grass and concrete) several times. The DNN basis function switches depending on the type of environment it operates in, while the corresponding adaptation coefficients $\hat{\boldsymbol{\theta}}$ remain mostly constant. This behavior also mirrors that observed in the simulation plots (Fig. 4.5) when a DNN with VFM is employed. Lastly, in Fig. 4.13, we present the lateral position error ($e^{\perp}$) and lateral velocity ($v_y^{\mathcal{B}}$) for the 3 controllers (a) nonlinear PD ((3.32) without the adaptation), (b) MAGIC with constant $\mathbf{\Phi}$ in (3.32) and(3.15), (c) MAGIC$^{\text{VFM}}$ with DNN $\mathbf{\Phi}$ in (3.32) and (3.15). Our method shows superior performance compared to the baseline nonlinear PD controller. The control coefficients for the three controllers are outlined in Table 4.5.

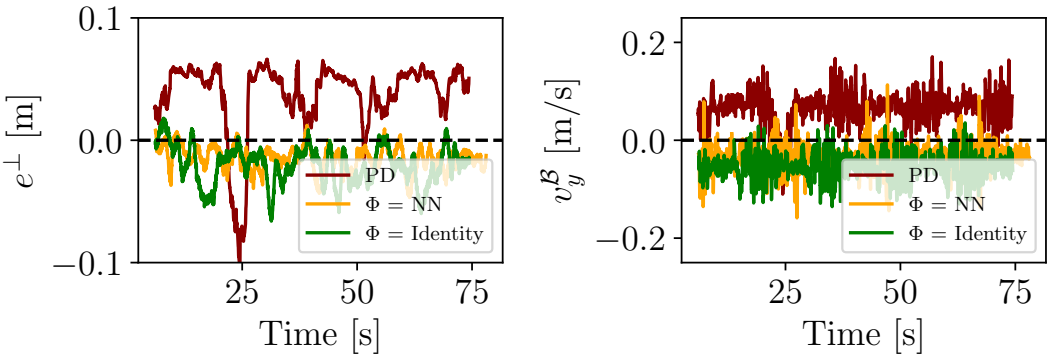


Figure 4.13: Convergence of the lateral error $e^{\perp}$ and lateral velocity $v_y^{\mathcal{B}}$ for a circular trajectory traversing two terrains like the one seen in Fig. 4.9 with $v_x^{\mathcal{B}} = 1.5\ m/s$. The velocity of the desired trajectory is limited by the performance of the VIO at higher speeds.

Table 4.5: Control coefficients for the lateral control of the Ackermann steering vehicle.

Controller	k_p	k_d	$\mathbf{\Gamma}$ diag.	$\mathbf{Q}$ diag.	$\mathbf{R}$ diag.	λ
(a) nonlinear PD	1.0	1.0	-	-	-	-
(b) $\mathbf{\Phi}$ constant	1.0	1.0	1.5	1.0	0.01	0.05
(c) $\mathbf{\Phi}$ DNN	1.0	1.0	1.5	1.0	0.01	0.05

Indoor Track Degradation Experiments

Indoor experiments were conducted at Caltech's Center for Autonomous Systems and Technologies (CAST) (Fig. 4.14). The primary objective of these experiments was to evaluate the robustness and performance of our proposed controller under artificially-induced track degradation. Specifically, the experiments quantify the extent of degradation that our controller can effectively manage and demonstrate its advantage over baseline controllers in similar scenarios. We compare three controllers: (a) nonlinear PD ((3.14) without the adaptation), (b) MAGIC with constant $\mathbf{\Phi}$ in (3.14) and (3.15), and (c) MAGIC$^{\text{VFM}}$ with DNN $\mathbf{\Phi}$ in (3.14) and (3.15). The DNN is not retrained on the new ground type, but the previously trained DNN from Sec. 4.6 is employed.

To simulate track degradation, a scalar factor is applied to one track that reduces its commanded rotation speed downstream of (and opaquely to) the controller. In this case, we apply a 70% reduction in speed to the right track using a step function with a period of 3 seconds, while keeping the left track operating 'nominally.' The GVR-Bot is commanded to follow a figure 8 trajectory, and the results are shown in Fig. 4.14, with the RMSE in position tabulated in Table 4.6. The results show that

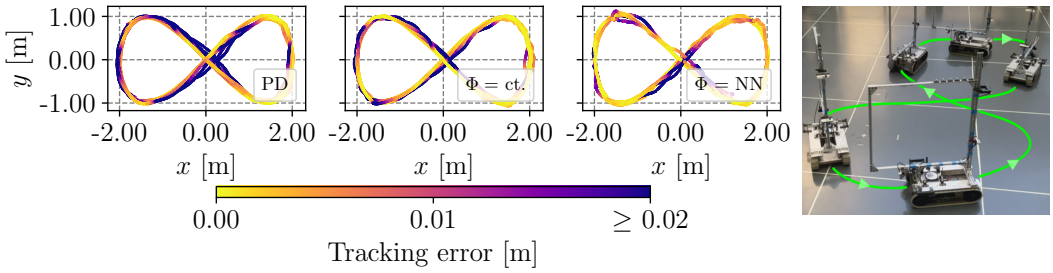


Figure 4.14: Tracking error for the three controllers during track degradation. (left) the performance for the nonlinear PD (the baseline). (middle and right) performance for the constant basis function and the terrain-informed basis function. On the right, we show the figure 8s trajectories for evaluating track degradation performance conducted indoors at CAST. The consistent floor of CAST ensures any slippage is consistent both within the figure 8 and between tests.

Table 4.6: Position tracking error statistics for the tracked vehicle experiencing track degradation.

Controller	Tracking RMSE [m]	Improvement
(a) nonlinear PD	0.102	-
(b) $\mathbf{\Phi}$ = constant	0.079	23%
(c) $\mathbf{\Phi}$ = DNN	0.070	31%

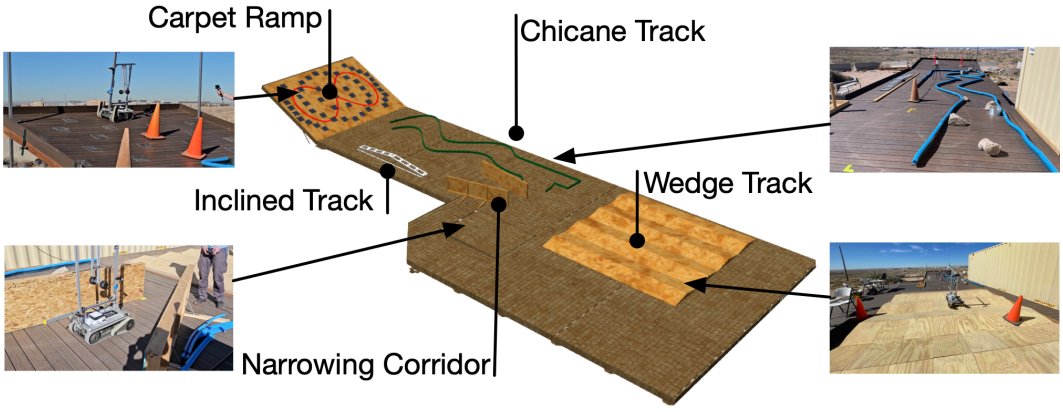


Figure 4.15: Combined Circuit for the DARPA LINC runs showing the full course with break outs of each of the elements. Track credit: Sandia National Laboratories team.

both constant $\mathbf{\Phi}$ and DNN $\mathbf{\Phi}$ controllers outperform the tracking of the baseline PD controller by 23% and 31%, proving the robustness to model mismatch of both DNN and non-DNN controllers.

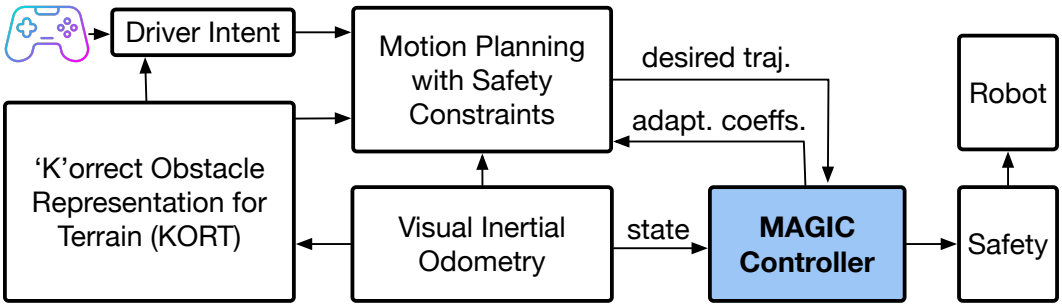


Figure 4.16: Architecture for our DARPA LINC software stack, with the MAGIC controller showcased in blue.

4.8 Performance at DARPA's Learning Introspection Control

The DARPA LINC program [6] develops machine learning-based introspection technologies that enable systems to respond to changes not predicted at design time. LINC took place throughout 2023 at Sandia National Laboratories.

The main exercise, Combined Circuit (Fig. 4.15), evaluated conditions such as track degradation, collisions, tip-over, and reduced cognitive load on the driver across a variety of test elements. Importantly, these exercises were completed with a human driver as the global planner in order to introduce additional challenges such as adversarial driving and driver intent inference.

For this exercise, we implemented the MAGIC controller from (3.14), (3.15) in which the basis function (4.3) was constant. Trajectories (both position and velocity) were generated using a sampling-based motion planner based on MCTS, with the desired goal locations generated using a 'driver intent' module that generated a desired path based on operator joystick inputs. The main modules of the software stack and their interfaces are shown in Fig. 4.16, with our MAGIC controller highlighted in blue.

To evaluate the performance of our MAGIC controller, we compare the estimated state (linear and angular velocities) from the VIO with the reference trajectory $\mathbf{v}_{\text{ref}}$ computed from the desired trajectories generated by the MCTS planner, as explained in Sec. 3.3. For the baseline, we compare the desired command from the joystick with the actual state from the VIO.

The following subsections discuss each of the components of the Combined Circuit and the performance of our controller. In Table 4.7-4.8, we present the performance metrics for the four exercises of the LINC project. Each exercise was traversed 4 times and the RMSE of the linear and angular velocity was computed.

Table 4.7: Performance metrics for the two of the four exercises of the DARPA LINC project.

	Chicane			Carpet Ramp		
	v error [m/s]	ω error [rad/s]	Time [s]	v error [m/s]	ω error [rad/s]	Time [s]
LINC off	0.28	0.45	16.36	0.96	0.59	19.96
LINC on	0.16	0.36	44.36	0.36	0.32	39.22
Improvement	**42%**	**19%**		**62%**	**46%**	

Table 4.8: Performance metrics for the two of the four exercises of the DARPA LINC project.

	Wedges			Narrow Corridor		
	v error [m/s]	ω error [rad/s]	Time [s]	v error [m/s]	ω error [rad/s]	Time [s]
LINC off	0.37	0.58	34.95	0.52	1.452	17.95
LINC on	0.25	0.34	39.72	0.19	0.44	23.73
Improvement	**33%**	**41%**		**64%**	**39%**	

Chicane Track

The Chicane Track highlighted the rejection of artificially induced track degradation, which was applied dynamically and opaquely as the GVR-Bot traversed the course. Due to the narrow track (the width is 0.9 m on average, 0.25 m wider than the GVR-Bot on both sides), track degradation leads to an increase in collisions with the chicane walls if not quickly adapted to. Our MAGIC controller was able to successfully adapt to these challenges, thus making this artificially induced track degradation almost imperceptible to the driver after a very short initial adaptation transient.

The effectiveness of the trajectory tracking on the Chicane Track is shown in Fig. 4.17 for both the baseline and the MAGIC controller. In the first two rows, the tracking of the velocities is emphasized. The third row shows the amount of degradation applied to the system. The bottom plot shows the estimated adaptation parameters $\hat{\boldsymbol{\theta}}$ changing in real time to compensate for the track degradation. Table 4.7-4.8 shows the improved performance of the MAGIC controller on this exercise. Our controller improved linear velocity tracking by 42%, and angular velocity tracking by 19%. Because the track degradation information, $\mathbf{B}_n + \boldsymbol{\Phi}\hat{\boldsymbol{\theta}}$ of (3.14), is estimated by the MAGIC controller in real-time, the MCTS can successfully generate trajectories that use this corrected control matrix, thereby successfully avoiding collisions with the

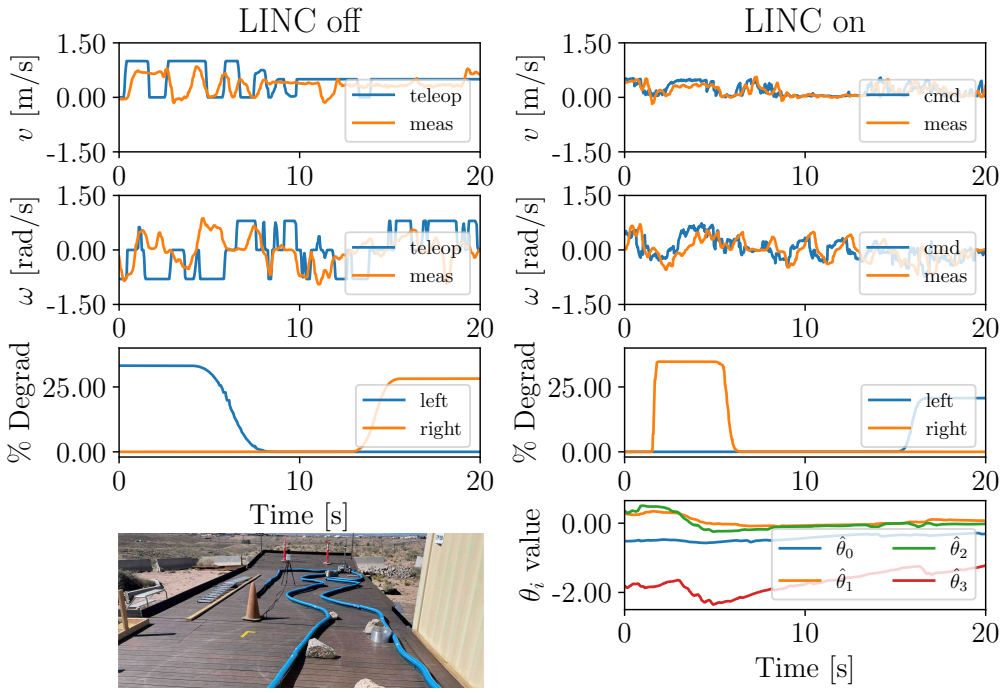


Figure 4.17: (Chicane Track) The left and right columns display tracking performance without and with our MAGIC controller, respectively.

chicane walls. When MAGIC was activated, the robot navigated the chicane track more cautiously, moving approximately 2.5 times slower than with the baseline controller. This reduction in pace was a result of the software stack prioritizing safety.

Carpet Ramp

The goal of the Carpet Ramp exercise is to restore and maintain control under *track degradation* and *variable slippage*, all whilst mitigating the risk of tipping over. The ramp had a slippery wooden surface with several patches of carpet to alter the ground friction coefficient, causing the tracks to slip asymmetrically. Additionally, as the roll angle of the robot increases over the incline, the traction of one of its tracks is reduced as more of the weight falls over one of the tracks due to the high vertical center of gravity. This imbalance in traction causes the dynamics of the GVR-Bot to change significantly, especially affecting the ability to turn. This restricted turning behavior is shown in Fig. 4.18. The plot in the first column, second row shows that although the operator attempts to turn the GVR-Bot, very little control authority in angular velocity is achieved. By comparison, when operated with our MAGIC controller, the robot adapts to the terrain, tracking safer turn commands that reduce

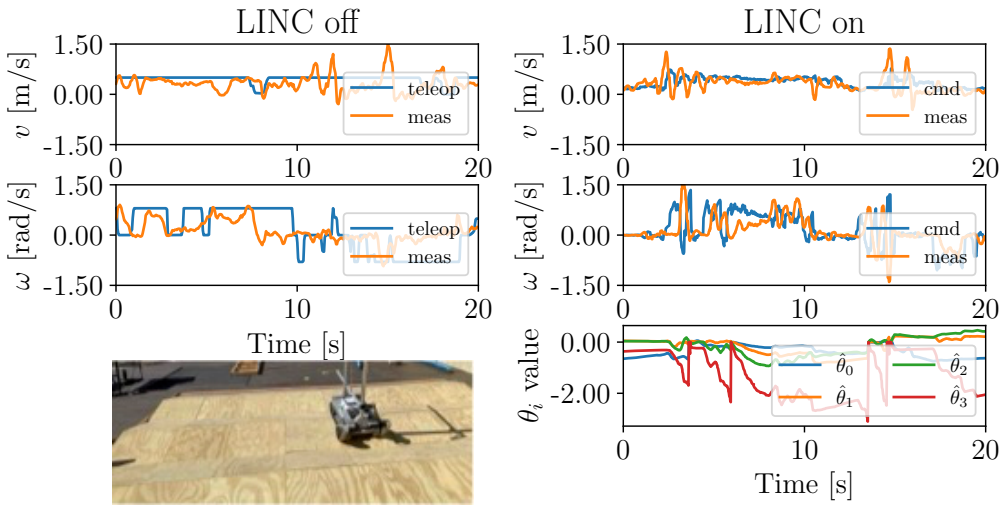


Figure 4.18: (Carpet Ramp) The left and right columns display tracking performance without and with our MAGIC controller.

the risk of tipping (second column, second row). As seen in the bottom row of Fig. 4.18, the adaptation coefficients, especially the one for the angular velocity, greatly increase to compensate for slip. This particular exercise demonstrates the greatest improvement in performance relative to the baseline, as seen in Table 4.7-4.8.

Narrowing Corridor

The aim of the Narrowing Corridor mirrored that of the Chicane Track, assesing the robot's ability to consistently navigate through a tight corridor despite track degradation. For the robot's performance, see Table 4.7-4.8.

MAGIC and Human-in-the-Loop

The LINC program was different from many robotics projects in that the global planner was human-driven rather than autonomous. This presents a challenge as MAGIC must not degrade the user driving experience but instead must augment it without the forward-planning and control input smoothness assumptions of typical robotic projects. The success of MAGIC in augmenting a human driver was twofold: firstly, MAGIC consistently ran fast enough such that there was no perceptible increase between joystick input and robot, and secondly, much of the adaptation to the changing terrain and vehicle were significantly reduced by MAGIC.

4.9 Conclusion

We introduced a novel learning-based composite adaptive controller that incorporates visual foundation models for terrain understanding and adaptation. The basis function of this adaptive controller, which is both state and terrain dependent, is learned offline using our proposed meta-learning algorithm. We prove the exponential convergence to a bounded tracking error ball of our adaptive controller and demonstrate that incorporating a pre-trained VFM into our learned representation enhances our controller's tracking performance compared to an equivalent controller without the learned representation. Our method showed a 53% decrease in position tracking error when deployed on a tracked vehicle traversing two different sloped terrains. We further demonstrated our algorithm on-board a car-like vehicle and showed that the learnt DNN basis function captures the residual dynamics generated by the two different terrains.

To gain insight into the inner workings of our full method, we empirically analyzed the features of the pre-trained VFM in terms of separability and continuity using support vector classifiers. This analysis showed positive empirical evidence that the DINO VFM is suitable for fine-grained discrimination of terrain types in images containing only terrain, and thus suitable for our control method.

We further tested our method under other perturbations, such as artificially induced track degradation. We demonstrated the effectiveness of our algorithm without terrain-aware basis function in human-in-the-loop driving scenarios. Our controller improved tracking of real-time human generated trajectories both in nominal and degraded vehicle states without introducing noticeable system delay as part of the DARPA's LINC project.

Acknowledgement

We thank T. Touma for developing the hardware and the testing environments replicas for the tracked vehicle; L. Gan and P. Proença for the state estimation; E. Sjögren for her early work on DINO features; Sandia National Laboratories team (T. Blada, D. Wood, E. Lu) for organizing the LINC tracks; J. Burdick for the LINC project management; A. Rahmani for the JPL Mars Yard support, LINC project management, and technical advice, and Y. Yue, B. Riviere, and P. Spieler for stimulating discussions. Some hardware experiments were conducted at Caltech's CAST.

BIBLIOGRAPHY

[1] Mathilde Caron, Hugo Touvron, Ishan Misra, Herv'e J'egou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. "Emerging Properties in Self-Supervised Vision Transformers". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (2021), pp. 9630–9640.

[2] M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. "Support vector machines". In: *IEEE Intell. Syst. and their Applications* 13.4 (July 1998), pp. 18–28.

[3] Richard Volpe. *JPL Robotics: The MarsYard III*. Accessed: 2024-01-11.

[4] US Army CCDC GVSC. *GVR-BOT Users Guide*. 2019.

[5] Patrick Geneva, Kevin Eckenhoff, Woosik Lee, Yulin Yang, and Guoquan Huang. "OpenVINS: A Research Platform for Visual-Inertial Estimation". In: *IEEE Int. Conf. Robot. Autom.* (May 2020).

[6] DARPA. *Learning Introspective Control (LINC)*. https://www.darpa.mil/program/learning-introspective-control.

Chapter 5

EXTENSION OF THE MAGIC-VFM METHOD TO MOTION PLANNING

In this section, we present several extensions of the MAGIC-VFM algorithm. The first extension (Sec. 5.1) incorporates visual information into an elevation map, demonstrating the advantages of adding terrain data for planning. The second extension (Sec. 5.2) involves integrating our approach with bi-level motion planning to update the dynamics model utilized in planning We used this strategy for the DARPA Learning Introspective Control competition and show that planning with a online-learned dynamics model can improve the performance of the system [1].

5.1 Terrain-informed Planning with Visual Foundation Models

Introduction

Our method, MAGIC-VFM, detailed in Chapter 2 is designed to dynamically adapt to terrain variations in real time via adaptive control. However, the challenge often extends beyond immediate control responses. Specifically, it is crucial for the robot to preemptively avoid certain areas. This requirement underscores a fundamental need for trajectory planning, rather than solely relying on reactive control strategies.

This subsection aims to address the question: "How can vision be integrated into a terrain-informed planning framework for autonomous robots?" To address this question, we propose to use a dynamics model that embeds terrain information through a Visual Foundation Model (VFM), and present simulation results utilizing a sampling based methodology for trajectory planning. We benchmark our solution when planning happens with a dynamics model that is not informed by the terrain and when the dynamics model is informed by the terrain, and show that the latter leads to safer trajectories, which avoids hazardous terrain.

Methods

We consider a robotic system with state $\mathbf{x} \in \mathbb{R}^n$ and control input $\mathbf{u} \in \mathbb{R}^m$ following the continuous-time dynamics

$$\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t), \tag{5.1}$$

where $\mathbf{f} : \mathbb{R}^{n+m+1} \to \mathbb{R}^n$ is the *nominal* dynamics model. Next, we augment the system in (5.1) with an additive disturbance $\mathbf{d}$ modelled in a similar fashion as in Chapter 2, i.e., $\mathbf{d} = \mathbf{\Phi}(\mathbf{x}, \mathbf{u}, \mathcal{E})\boldsymbol{\theta}(t)$, with $\boldsymbol{\theta}(t)$ denotes the time-varying vector of linear parameters and $\mathbf{\Phi}(\mathbf{x}, \mathbf{u}, \mathcal{E})$ is a basis functions that depend on the state, control input, and the terrain map $\mathcal{E}$. As in Chapter 2, $\mathcal{E}$ is the output of a Visual Foundation Model (VFM) that processes the raw sensor data to provide a terrain map.

Given the dynamics with disturbance we solve the following planning problem: Given an initial position $\mathbf{x}_0$, the goal is to find a sequence of control inputs $\mathbf{u}_{[1:T]}$ that minimizes the cost function

$$\begin{aligned} \mathbf{x}^*_{[1:T]}, \mathbf{u}^*_{[1:T]} = & \underset{\mathbf{x}_{[1:T]},\, \mathbf{u}_{[1:T]}}{\arg\min} \sum_{k=0}^{T} c(\mathbf{x}_k, \mathbf{u}_k) \\ \text{subject to } & \mathbf{x}_{k+1} = \mathbf{x}_k + (\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{\Phi}(\mathbf{x}, \mathbf{u}, \mathcal{E})_k \boldsymbol{\theta}_k)\Delta t, \ \forall k \in [0, T-1], \\ & \mathbf{x}_k \in \mathcal{X}, \ \forall k \in [0, T], \\ & \mathbf{u}_k \in \mathcal{U}, \ \forall k \in [0, T], \end{aligned} \tag{5.2}$$

where we specify the dynamics in discrete form, with Δt being the discretization step, T is the final time, $\mathcal{X}$ is a compact state space, $\mathcal{U}$ is the set of allowable inputs, and $c : \mathbb{R}^{n+m} \to \mathbb{R}$ is a cost function.

Sampling-based Planning

To solve the motion planning problem in (5.2), we use a sampling-based planner, which is well-suited for handling non-convex cost functions and nonlinear dynamics. Specifically, we adopt the Cross Entropy Method (CEM) [2] in Algorithm 3 due to its simplicity. The algorithm is initialized with a Gaussian distribution on the action space, and the parameters of the distribution, μ and σ, are updated in each iteration to improve the quality of the samples (Line 13). The algorithm terminates when the number of iterations exceeds the maximum number of iterations $K_{\max}$ or the cost function is minimized. Solving (5.2) also requires integrating the dynamics forward in time, which assumes access to an environment map that encodes the VFM features.

Simulation results

We simulate a two dimensional double integrator system with damping, where the state is $\mathbf{x} = [x, y, \dot{x}, \dot{y}]$ and the control input is $\mathbf{u} = [u_x, u_y]$. The dynamics are given

Algorithm 3 Cross Entropy Method [2]

Input
    $K_{\max}$ - maximum number of iterations.
    N - number of samples in each iteration.
    T - time horizon
    ρ - elite fraction (i.e., top percentile of samples).
    σ - initial standard deviation, μ - initial mean.
Output: Best actions found.
while not converged or $k < K_{\max}$ **do**
    Generate N samples of action sequences from $\mathcal{N}(\mu, \sigma)$ on time horizon T.
    Evaluate the cost function for each sample sequence.
    Sort the samples based on the cost function.
    Select the top ρ samples (elite samples).
    Update the parameters of the distribution, μ and σ, based on the elite samples.
end while

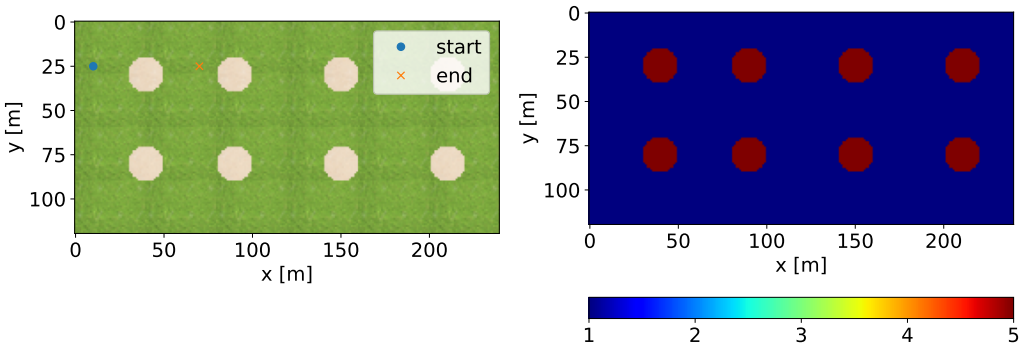


Figure 5.1: Simulation environment (a) Environment with sand and grass with the starting position and the end position. (b) The values of the damping coefficients for grass (b=1.0) and sand (b=5.0).

by

$$\begin{aligned} \ddot{x} &= u_x - b_1 \dot{x}, \\ \ddot{y} &= u_y - b_2 \dot{y}, \end{aligned} \tag{5.3}$$

where b_1 and b_2 are the damping coefficients, dependent on the terrain. The nominal system has no damping, thus $b_1 = b_2 = 0$. In Fig. 5.1, we present the environment with sand and grass, and the values of the damping coefficients for the two environments.

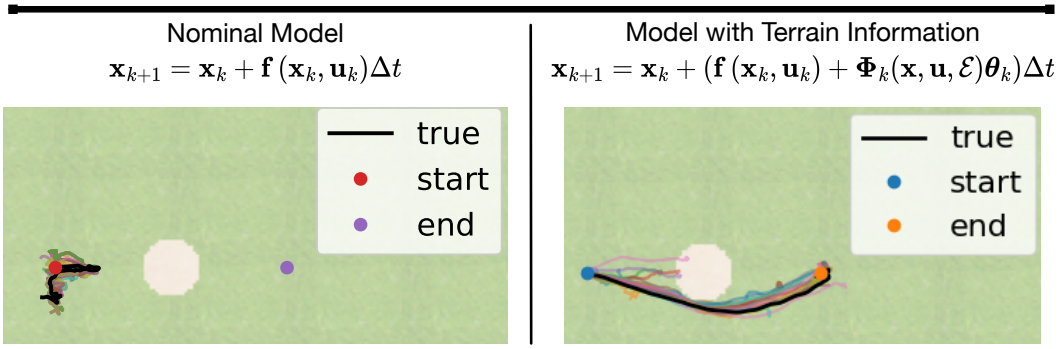


Figure 5.2: Qualitative results of the CEM planner when the dynamics model is informed by the terrain map (left) and when it is not (right). The robot is able to avoid the sand and reach the goal faster when the dynamics model is informed by the terrain map.

Similar to Chapter 2, we learn the basis function $\mathbf{\Phi}$ in (5.2) using a DNN that takes as input the output of a VFM as well as the state. We invite the reader to refer to Chapter 2 for more details on the training of the DNN and the environment setup in Chapter 4. We compare the performance of the CEM planner when the dynamics model contains terrain information and when it does not (i.e., the optimization problem in (5.2) is solved with and without the term $\mathbf{\Phi}(\mathbf{x}, \mathbf{u}, \mathcal{E})\boldsymbol{\theta}_k$). Note that compared to Chapter 2, we keep $\boldsymbol{\theta}$ constant, with their values computed from the training algorithm (Algorithm 1). The qualitative results are presented in Fig. 5.2. We show that when the planner uses the terrain-informed dynamics model, the robot avoids the sand, which adds a slow-down to the robot, and reaches the goal faster than when the planner uses the dynamics model without terrain information.

5.2 Improved Motion Planning with Rapidly Learned Dynamics from Adaptive Control: MCTS-MAGIC

Joint work with J. Lathrop, J. A. Preiss, F. Xie, M. Anderson, and S.-J. Chung

In this work, we combine the adaptive controller from Chapters 2-4 with a bi-level motion planner that contains both search (using Monte-Carlo Tree Search) and optimization components (using Sequential Convex Optimization) [3]. In this manner, the low-level controller learns the disturbances using composite adaptation with both prediction and tracking errors, which are then passed to the motion planner for improved predictions. Note that in the previous subsection (Section 5.1), the adaptation coefficients $\boldsymbol{\theta}$ were kept constant.

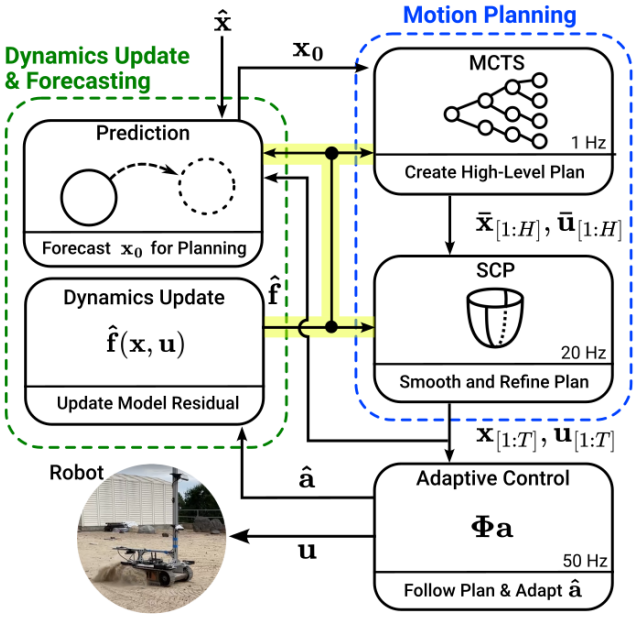


Figure 5.3: Component diagram of our autonomy stack. Notable is the highlighted arrow carrying an online-adapted estimate of the dynamics $\hat{\mathbf{f}}$ to the prediction and planning modules (Figure created by J. Lathrop).

Methods

Consider a robotic system with state $\mathbf{x} \in \mathbb{R}^n$ and input $\mathbf{u} \in \mathbb{R}^m$. We study systems with control-affine dynamics

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{f}_n(\mathbf{x}) + \mathbf{B}_n(\mathbf{x})\mathbf{u} + \mathbf{d}(\mathbf{x}, \mathbf{u}), \tag{5.4}$$

where $\mathbf{f}_n : \mathbb{R}^n \to \mathbb{R}^n$ and $\mathbf{B}_n : \mathbb{R}^n \to \mathbb{R}^{n \times m}$ are nominal dynamics and input matrices, respectively, and $\mathbf{d} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is an unknown, potentially time-varying, modeling error.

Similar to Chapter 2, we model the disturbance entering the system as a control matrix perturbation $\mathbf{d}(\mathbf{x}, \mathbf{u}) \approx \left(\sum_{i=1}^{p} \theta_i \mathbf{\Phi}_i\right) \mathbf{u}$, for a vector of adaptation parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_p]$ and a set of basis matrices $\mathbf{\Phi} = \{\mathbf{\Phi}_1, \mathbf{\Phi}_2, \ldots, \mathbf{\Phi}_p\}$. This model is intended to capture both actuator degradation and terrain slippage. For notational simplicity, we denote the tensor multiplication as $[\mathbf{\Phi}\theta] = \sum_{i=1}^{p} \boldsymbol{\theta}_i \mathbf{\Phi}_i$.

Using our adaptation parameter update laws, we form estimates $\hat{\boldsymbol{\theta}}$ of the unknown true values $\boldsymbol{\theta}$. The adaptation parameter estimates lead to an estimate of the disturbance term in (5.4), $\hat{\mathbf{d}}(\mathbf{x}, \mathbf{u}) = [\mathbf{\Phi}\hat{\boldsymbol{\theta}}]\mathbf{u}$, which is then provided to the motion planning modules. We use the following control:

$$\mathbf{u} = (\mathbf{B}_n + [\mathbf{\Phi}\hat{\boldsymbol{\theta}}])^{\dagger}(\dot{\mathbf{x}}_{\text{ref}} - \mathbf{f}(\mathbf{x}_{\text{ref}}) - \mathbf{K}_d \mathbf{s}) \tag{5.5}$$

where $\mathbf{s} = \mathbf{x} - \mathbf{x}_{\text{ref}}$ is the composite tracking error, with $\mathbf{x}_{\text{ref}} = \mathbf{x}_{\text{des}} + \mathbf{K}_p(\mathbf{x} - \mathbf{x}_{\text{des}})$, $\mathbf{K}_p, \mathbf{K}_d$ positive definite gains, and $\mathbf{B}_n$ is the nominal control matrix. Our corresponding parameter update laws are similar to those in [4]:

$$\dot{\hat{\theta}}_i = \underbrace{-\lambda\hat{\theta}_i}_{\text{exp. forgetting}} \underbrace{-\gamma_i \mathbf{u}^\top \mathbf{\Phi}_i^\top \mathbf{W}^{-1}([\mathbf{\Phi}\hat{\boldsymbol{\theta}}]\mathbf{u} - \mathbf{y})}_{\text{online parameter estimation}} + \underbrace{\gamma_i \mathbf{s}^\top \mathbf{\Phi}_i \mathbf{u}}_{\text{tracking error}}$$
$$\dot{\gamma}_i = -2\lambda\gamma_i + q_i + \gamma_i \mathbf{u}^\top \mathbf{\Phi}_i^\top \mathbf{W}^{-1} \mathbf{\Phi}_i \mathbf{u}\gamma_i, \ \forall i \in 1 \ldots p. \tag{5.6}$$

This composite adaptation can be interpreted in an $\mathcal{H}_\infty$ filtering framework [5, 6] combined with tracking error adaptation, in which the variable $\mathbf{a}$ is the state to be estimated and $\mathbf{\Gamma} = [\gamma_1, ..., \gamma_p]$ is a variance matrix. In the same context, we define the process noise as $\mathbf{q} = [q_1, ..., q_p]$, and the measurement noise as $\mathbf{W} \in \mathbb{R}^n$. Here, λ is used for exponential forgetting to account for time-varying dynamics. The measurement signal is $\mathbf{y} = \dot{\mathbf{x}} - \mathbf{f}_n(\mathbf{x}) - \mathbf{B}_n(\mathbf{x})\mathbf{u}$, the observed error in the nominal dynamics. As shown in Chapter 3, the prediction term adds a gradient-descent correction that drives the parameter estimates toward their true values, while the tracking error term finds the update direction to minimize tracking error.

While a tracking error only-based update can lead to convergence to the desired trajectory, tracking error-based adaptive laws cause adaptation parameters to converge to non-physical values. We use a composite adaptation law that combines tracking error and prediction error terms to guarantee convergence of both tracking error and adaptation parameters (see Theorem 1), thereby permitting simultaneous feedback control and parameter estimation.

Given an initial position $\mathbf{x}_0$, we seek a trajectory that optimizes the sum of a known reward function $\mathbf{R} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}_{\geq 0}$ plus a terminal value function $\mathbf{V} : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$:

$$\mathbf{x}^*_{[1:K]}, \mathbf{u}^*_{[1:K]} = \underset{\mathbf{x}_{[1:K]},\, \mathbf{u}_{[1:K]}}{\arg\max} \sum_{k=0}^{K-1} \mathbf{R}(\mathbf{x}_k, \mathbf{u}_{k+1}) + \mathbf{V}(\mathbf{x}_K)$$
$$\text{subject to } \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{f}(\mathbf{x}_k, \mathbf{u}_{k+1})\Delta t, \ \forall k \in [0, K-1],$$
$$\mathbf{x}_k \in \mathcal{X}\backslash\mathcal{O}, \ \forall k \in [0, K],$$
$$\mathbf{u}_k \in \mathcal{U}, \ \forall k \in [1, K],$$

where $\mathcal{X}$ is a compact state space and $\mathcal{O}$ is a set of obstacles. We denote the set of allowable inputs $\mathcal{U}$. Even with a fully known dynamics model, this problem is in general nonconvex and NP-hard. We, however, assume we only have access to the nominal dynamics model $\mathbf{f}_n(\mathbf{x}) + \mathbf{B}_n(\mathbf{x})\mathbf{u}$ and not the full dynamics $\mathbf{f}(\mathbf{x}, \mathbf{u})$. Our

Algorithm 4 Adaptive Motion Planning

Inputs: state estimate $\hat{\mathbf{x}}$, nominal model $\mathbf{f}_n, \mathbf{B}_n$
Parameters: planning horizon H, replanning horizon T

$\hat{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ ▷ Initialize adaptation parameters
$\mathbf{x}_{[1:T]}, \mathbf{u}_{[1:T]} \leftarrow \{\mathbf{0}\}, \{\mathbf{0}\}$ ▷ Initialize first plan
while not terminated **do**
  ▷ Update dynamics model
  $\hat{\mathbf{f}}(\mathbf{x}, \mathbf{u}) \leftarrow \mathbf{f}_n(\mathbf{x}) + \mathbf{B}_n(\mathbf{x})\mathbf{u} + \hat{\mathbf{d}}(\mathbf{x}, \mathbf{u}, \hat{\boldsymbol{\theta}})$
  ▷ Predict future initial position
  $\mathbf{x}_0 \leftarrow \text{Predict}(\hat{\mathbf{x}}, \mathbf{x}_{[1:T]}, \mathbf{u}_{[1:T]}, \hat{\mathbf{f}})$
  ▷ Make coarse plan
  $\bar{\mathbf{x}}_{[1:H]}, \bar{\mathbf{u}}_{[1:H]} \leftarrow \text{MCTS}(\mathbf{x}_0, \hat{\mathbf{f}})$
  ▷ Generate smooth plan
  $\mathbf{x}_{[1:H]}, \mathbf{u}_{[1:H]} \leftarrow \text{SCP}(\bar{\mathbf{x}}_{[1:H]}, \bar{\mathbf{u}}_{[1:H]}, \bar{\mathbf{x}}_{[1:H]}, \hat{\mathbf{f}})$
  ▷ Follow refined plan with a tracking controller
  $\hat{\boldsymbol{\theta}} \leftarrow \text{Adaptive Controller}(\hat{\mathbf{x}}, \mathbf{x}_{[1:T]}, \mathbf{u}_{[1:T]}, \hat{\boldsymbol{\theta}})$
end while

method forms an estimate $\hat{\mathbf{d}}(\mathbf{x}, \mathbf{u})$ of the disturbance via adaptive control and solves an approximate form of (5.7) with the estimated dynamics model and a shorter horizon $H < K$.

For real-time deployment, our algorithm operates in receding-horizon fashion. When a desired trajectory with horizon H is generated, only the initial section up to a shorter horizon $T < H$ is tracked by the controller, after which a new plan is available. This continues until the full horizon K of (5.7) is reached (see Algorithm 4 and Fig. 5.3).

Results

This framework was the main algorithm running during at the DARPA Learning Introspective Control competition, where it was able to successfully avoid obstacles (Fig. 5.4) and pass through narrow corridors even when actuator degradations were present. The quantitative performance metrics of this algorithm can be seen in Table 4.7-4.8. The algorithm was able to adapt to the changing dynamics of the robot and provide a safe trajectory to the robot.

In Fig. 5.5, we show the behaviour of the robot during the MCTS-MAGIC planning process. First, in (a), we show the expected path of the robot around a rock. If there is no path forward, the robot will backtrack slowly until it can turn around the rock. In (b), the actual path of the robot around a cone is presented. The robot experienced

Figure 5.4: The 5 elements of the Combined Circuit at DARPA Learning Introspective Control. Our MCTS-MAGIC algorithm actively replans trajectories to avoids collisions and accurately tracks desired paths even under heavy track degradation.

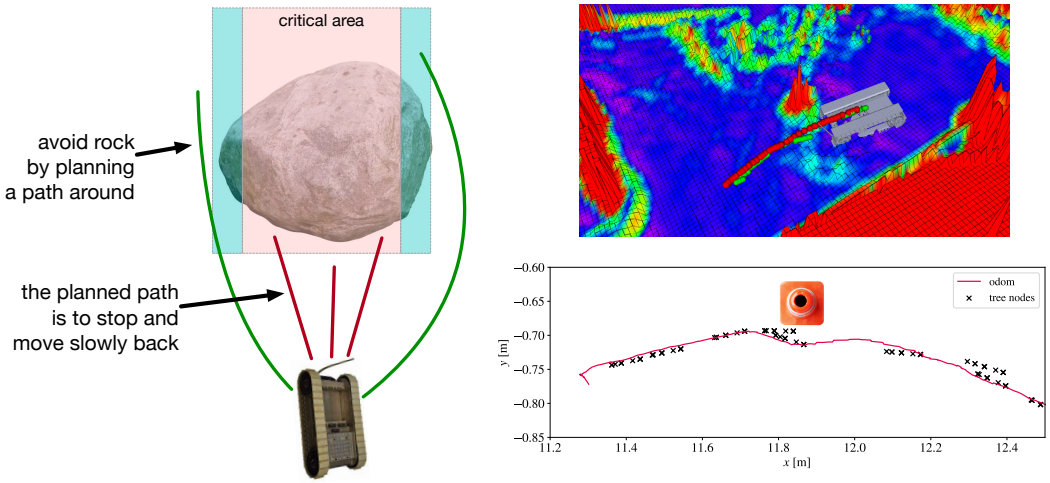


Figure 5.5: Behaviour of the robot during the MCTS-MAGIC planning process. (a) Example of a trajectory behaviour around a rock. (b) Top figure: RVIZ visualization of the robot's path around a cone. Bottom figure: The odometry path of the robot around the cone during the MCTS-MAGIC planning process. The robot experienced track degradation but planning with learned dynamics enabled it to avoid the cone. (Bottom plot credit: J. Lathrop and J. Alindogan)

track degradation but planning with learned dynamics enabled it to avoid the cone. In black, we emphasize the points where the safety controller (not described in this thesis) is active.

BIBLIOGRAPHY

[1] DARPA. *Learning Introspective Control (LINC)*. https://www.darpa.mil/program/learning-introspective-control.

[2] Zdravko I. Botev, Dirk P. Kroese, Reuven Y. Rubinstein, and Pierre L'Ecuyer. "Chapter 3 - The Cross-Entropy Method for Optimization". In: *Handbook of Statistics*. Ed. by C.R. Rao and Venu Govindaraju. Vol. 31. Handbook of Statistics. Elsevier, 2013, pp. 35–59.

[3] Benjamin Riviere, John Lathrop, and Soon-Jo Chung. "Monte Carlo Tree Search with Spectral Expansion for Planning with Dynamical Systems". In: *Science Robotics* 9.97 (2024), eado1010.

[4] Michael O'Connell, Guanya Shi, Xichen Shi, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung. "Neural-Fly Enables Rapid Learning for Agile Flight in Strong Winds". In: *Science Robotics* 7.66 (2022), eabm6597.

[5] Tamer Başar and Pierre Bernhard. *H-infinity Optimal Control and Related Minimax Design Problems: A Dynamic Game Approach*. Springer Science & Business Media, 2008.

[6] Hiroyasu Tsukamoto and Soon-Jo Chung. "Robust Controller Design for Stochastic Nonlinear Systems via Convex Optimization". In: *IEEE Transactions on Automatic Control* 66.10 (2020), pp. 4731–4746.

Chapter 6

DETECTION OF UNCOOPERATIVE TARGETS AND COMPONENTS ON SMALL SATELLITES USING VISUAL FOUNDATION MODELS

This chapter is based on the publications:

Hannah Grauer*, **Elena Sorina Lupu***, Connor Lee, Darren Rowen, Benjamen Bycroft, Phaedrus Leeds, John Brader, and Soon-Jo Chung. "Vision-Based Detection of Uncooperative Targets and Components on Small Satellites". In: *38th Annual Small Satellite Conference* (2024), pp. 5321–5327

Hannah Grauer*, **Elena Sorina Lupu***, Connor Lee, Cailyn Smith, Sulekha Kishore, Darren Rowen, Benjamen Bycroft, Phaedrus Leeds, John Brader, and Soon-Jo Chung. "Vision-Based Detection of Uncooperative Targets and Components on Small Satellites". In: *ICRA Workshop on Thermal Infrared In Robotics* (2025), pp. 5321–5327

6.1 Chapter Overview

As shown in Chapters 2-4, using Visual Foundation Models (VFMs) for robotic control tasks can significantly improve performance for ground vehicles. In this chapter, we extend the use of VFMs to the domain of spacecraft parts segmentation. We propose a method for spacecraft segmentation that leverages the performance of a VFM distilled into a lightweight fast semantic network to increase segmentation accuracy while keeping inference and training time low. Our method is designed to align with Aerospace Corporation's Edge Node Lite mission. We have successfully implemented our algorithm on the Edge Node, which launched on-board a Falcon rocket in March 2025. This milestone allows us to demonstrate more advanced, machine learning-based algorithms directly in space and build flight heritage for future missions.

6.2 Introduction

On-orbit services (OOS) [3] is a growing sector of the space industry that involve various activities aimed at extending the life, enhancing the capabilities, or re-purposing spacecraft and other objects in space. Some examples of OOS are

in-space servicing and inspection [4], rendezvous and docking [5, 6], or debris removal. [7–10] To make these services viable for small satellites (SmallSats), we must develop advanced autonomous capabilities that reduce costs and minimize the need for human intervention. As an example, in-space servicing and inspection enable the repair and upgrade of satellites while in orbit. One of the key benefits of these technologies is cost reduction. By repairing defunct spacecraft in orbit, the high costs associated with launching new spacecraft can be reduced.

The procedure for in-space servicing involves a sequence of steps such as intercepting with the spacecraft, close-proximity operations and eventually rendezvous and docking. Oftentimes, the servicing vehicle deployed in-orbit does not need to be a large spacecraft. Instead, a SmallSat equipped with necessary sensors, compute, and autonomy is more advantageous from a cost and weight standpoint. However, as their size is further decreased (i.e., nanosatellites and cubesats) the sensors and compute tend to be limited [11].

To achieve autonomous OOS, it is important to first detect and continuously track the target spacecraft from afar. As the servicing spacecraft approaches the target, it must be equipped to accurately identify specific components for maintenance or find suitable attachment points to facilitate the de-orbiting of the spacecraft using on-board cameras. On the ground, computer vision models using machine learning have made significant progress in object detection [12] and segmentation [13, 14]. However, these achievements have yet to be fully applied in space. Like field robotic perception development in terrestrial applications [15–17], this shortfall is primarily due to the limited availability and small sizes of publicly-available datasets available for training, resulting in less robust model performance. Additionally, the computational resources required for both training and inference in space pose a significant challenge in terms of on-board compute, on-board storage, and Earth-to-satellite latency.

Contributions

To address these limitations we present a long and short range methodology that is able to track a spacecraft from afar and identify spacecraft components using segmentation, as demonstrated by Figure 6.1. Our contributions are as follows:

- A novel method to generate long-range images for training using thermal cameras, which are more robust to varied lighting conditions in space.

- A long range detection algorithm operating on thermal images of a spacecraft using the YOLOv8 (You Only Look Once) model [18].
- A novel method for spacecraft part segmentation at a shorter range that leverages the performance of a visual foundation model (VFM) distilled into a lightweight fast semantic network to increase segmentation accuracy while keeping inference and training time low.

Our methodology is designed to align with Aerospace Corporation's Edge Node and risk-reduction Edge Node Lite missions for formation flight and rendezvous and proximity operations. These missions use multiple small satellites equipped with miniature sensor and computing hardware. We use the Edge Node spacecraft as the target. We additionally use from the missions the FLIR Boson+ Long-wave infrared (LWIR) camera in our custom dataset creation and the Jetson TX2 NX to test performance. [19]

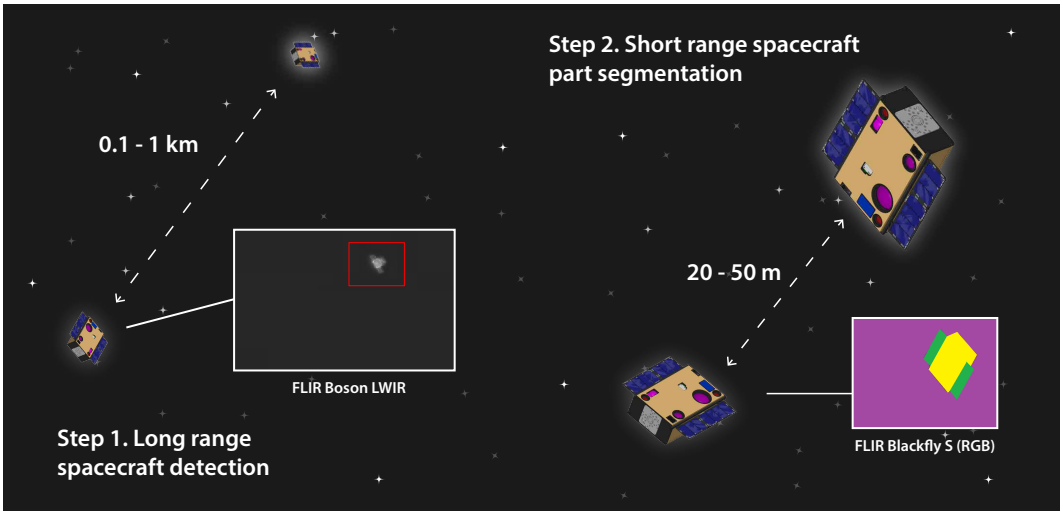


Figure 6.1: Overview of the long-short detection architecture proposed for the Edge Node mission or other OOS tasks. [19] We propose a long distance detection model on thermal images for detection at larger ranges into a segmentation model that identifies spacecraft parts for OOS on RGB images.

6.3 Related Work

Long Range Object Detection

In the domain of spacecraft pose estimation, recent works have adopted various computer vision models, particularly focusing on the use of object detectors based on convolutional neural networks (CNNs). While current state-of-the-art computer vision benchmarks, including object detection, are dominated by transformer-based

architectures [20–22], CNNs exhibit higher computational efficiency on embedded devices and are more suited for real-time, low-compute robotic applications. In field robotic applications where publicly-available curated datasets are rare or limited, CNNs are also more favorable due to their built-in spatial inductive bias (via 2D convolutions) that makes them less prone to overfitting when training with smaller datasets [15–17, 20, 23].

Current CNN-based object detectors fall into one of two categories: two-stage and one-stage detectors. Two-stage detectors include models like Faster-RCNN [24] which perform initial region proposal steps before moving forward to bounding box refinement and classification. On the other hand, single-stage detectors like SSD [25], YOLOv3 [26], RetinaNet [27], and EfficientDet [28] perform bounding box refinement and classification in one shot. Single-stage detectors generally exhibit faster inference times and lower computational overhead compared to larger, two-stage detectors, and are more suited for real-time robotic applications in space [29].

Although the use of object detectors in spaceborne autonomy stacks is not new, current spacecraft-related works only leverage object detectors to detect objects of interest, such as uncooperative spacecraft and debris, at shorter ranges within 75 meters [29, 30]. However, for spacecraft seeking to maneuver closer to an uncooperative spacecraft to perform OOS, the initial ranges to the target can easily exceed such distances and require long-range detection capabilities that current work do not provide. This is exacerbated by the fact that prior works only perform object detection within the visible spectrum. This means that target objects will appear small at long distances and be photometrically imperceptible due to low-light conditions in space.

In our work, we develop long-range spacecraft detection capabilities that leverage thermal imaging which can easily highlight the spacecraft, even at long ranges, due to their extreme thermal signatures from onboard computers juxtaposed against the vacuum of space. We build upon existing work that leverage YOLO object detector variants for thermal object detection from airborne platforms [31], while also exploring new directions for spacecraft detection using zero-shot segmentation models similar to Segment Anything [32].

Short Range Spacecraft Parts Segmentation

Spacecraft parts identification and localization is required for OOS tasks, since they enable precise manipulation of the parts. Current methods for spacecraft parts localization operate at short range and localize parts either through object detection or segmentation. Object detection-based methods typically utilize aforementioned single-stage detectors like YOLO to predict bounding boxes encompassing particular spacecraft parts. [33–35] Segmentation-based methods typically utilize popular CNN-based semantic segmentation models to perform per-pixel classification of incoming imagery. Like in object detection, large transformer-based models dominate current semantic segmentation benchmarks but are generally not suitable for real-time use. In this work, we develop a robust, real-time semantic segmentation model for spacecraft part segmentation by transferring knowledge from a large vision transformer model, Dino [36], to a real-time capable FastSCNN [37] network.

Aside from model selection, one notable challenge in spacecraft part localization, and field robotics in general, is the scarcity of publicly-available datasets for model training and development [15–17, 33, 34, 38]. Some works address this issue by developing RGB spacecraft part datasets using real images, commonly scraped off the web, and synthetic images, typically rendered via Blender. [34, 38] However, such works typically do not release their curated dataset to the public, which is especially common in the spacecraft industry due to the confidential nature of most projects. As such, most methods rely on techniques like transfer learning [33] to finetune pretrained CNNs on small datasets.

In this work, we propose a method for spacecraft part segmentation that is both storage- and computationally-efficient, enabling low-latency scene perception for OOS tasks. In contrast with prior work, we use knowledge distillation to transfer learned features from a large visual-foundation model to a lightweight CNN that can run in real-time on low-compute hardware. Like other works, we develop a custom RGB dataset using our spacecraft simulator hardware [39] for model training and benchmarking. We also validate our method on a publicly-available spacecraft parts dataset.

Knowledge Distillation

Knowledge distillation (KD) [40, 41] seeks to transfer the learned behavior of a larger model (teacher network) to a smaller, lightweight model (student network). KD is a promising technique for spacecraft applications because the deployment

of models in space necessitates model compression that decreases the size of the model in order to increase inference speeds, while maintaining strong predictive performance [42].

In general, KD from a single teacher can be generalized into two categories: knowledge from logits and knowledge from intermediate features. When distilling knowledge from logits, the student network is trained to match the output distribution of the teacher network, usually through the cross entropy loss [40, 43, 44]. In the latter case, knowledge is distilled by driving the features of the student to match the intermediate features of the teacher [44–46], using some distance metric like the L_2 norm along with more sophisticated methods like an adversarial loss [46].

In our work, we utilize a simple KD method that drives the penultimate features of a FastSCNN network to match that of a large pretrained vision transformer [20] using the L_2 distance metric. We use KD to pretrain our FastSCNN segmentation network before finetuning on our small spacecraft part dataset.

6.4 Short Range Spacecraft Parts Segmentation

In this section, we present our method to autonomously detect spacecraft parts at a short range of 20-50 m from an observing spacecraft. We presume that, once identified via long range tracking [1], the spacecraft has the capability to follow and track the uncooperative target.

For this scenario, we assume the spacecraft is equipped with a high resolution RGB camera, and switches use from the thermal camera. This change potentially sacrifices reliability under variable lighting conditions, a factor critical in space environments where lighting can be highly dynamic. However, we gain the RGB cameras' ability to capture detailed visual information in three color channels, enhancing image detail and feature recognition.

Subsequently, the collected frames are downlinked to Earth for annotating the segmentation masks and classifications of different parts of the spacecraft, such as solar panels, body, and antenna. Once annotated, these annotations can be uplinked back to the spacecraft[1], where they are utilized to train onboard machine learning models. Alternatively, training can also be done on the ground with weights uploaded back to the spacecraft. Models trained on the ground, however, are subject to bandwidth uploading constraints.

[1] a requirement of the Edge Node mission is to do training on-board

Additionally, we leverage lightweight models in order to enable future possibilities of onboard training with the goal of one day using unsupervised or self-supervised models for fully autonomous learning onboard. The intention is to enable these models to autonomously segment and classify spacecraft components in real-time using minimal images to limit the reliance on ground operations.

Spacecraft Parts Dataset Creation

Custom RGB Spacecraft Parts Dataset

To create a semantic segmentation dataset for spacecraft parts at a shorter range, we took RGB photos of the Edge Node spacecraft in our spacecraft robotic simulator [39] at different orientations and distances. To streamline the labeling process, we use AnyLabeling, an AI-assisted data labeling program that uses SAM with key point prompts to generate the segmentation mask of an object. [47] We manually review the generated labels and perform minor corrections. Our final dataset consists of 301 annotated images. Examples of such images from the dataset can be seen in the top row of Fig. 6.2. For training, we split the dataset into train, validation, and test sets at a 75-20-5 ratio.

Adelaide Spacecraft Parts Dataset

To evaluate the generalizability of the segmentation models, we utilized a publicly-available dataset [38] consisting of 3117 images of various satellites and space stations. This dataset includes both synthetic and real images and videos, providing a diverse range of spacecraft types. Unlike our custom dataset, which contains multiple images of the same spacecraft captured in a controlled lab environment, the Adelaide dataset features a wide array of spacecraft, often with only one or a few images per spacecraft. Additionally, the backgrounds in the Adelaide dataset are more realistic, depicting space or Earth, as opposed to the controlled lab settings in our images. [38] Examples of such images from the dataset can be seen in the bottom row of Fig. 6.2. As before, we also split this dataset into train, validation, and test sets at a 75-20-5 ratio.

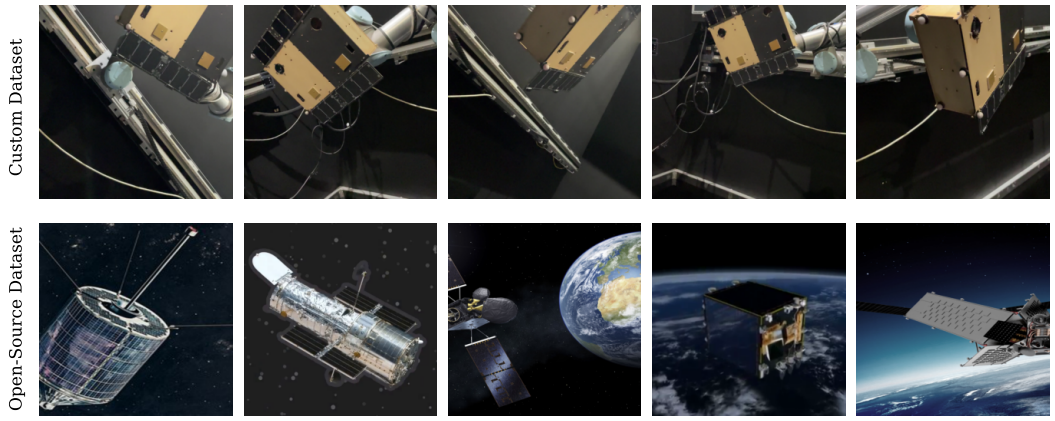


Figure 6.2: Example images from the spacecraft parts dataset generated in the spacecraft robotic simulator (top row) and the Adelaide dataset (bottom row).

Algorithm 5 VFM distillation into a real-time CNN

Input: Networks $\mathcal{F}_{\text{vfm}}$, $\mathcal{F}_{\text{cnn}}$, training dataset $\mathcal{D}$
Number of epochs N, learning rate η
Output: Distilled CNN weights $\mathbf{b} * \theta_{\text{cnn}}$

for $n = 1 : N$ **do**
  Sample image batch $\mathbf{b} * x_n$ from $\mathcal{D}$

  $\mathbf{b} * z_{\text{vfm}} := \mathcal{F}_{\text{vfm}}(\mathbf{b} * x_n;\ \mathbf{b} * \theta_{\text{vfm}})$, ▷ Get VFM features
  $\mathbf{b} * z_{\text{cnn}} := \mathcal{F}_{\text{cnn}}(\mathbf{b} * x_n;\ \mathbf{b} * \theta_{\text{cnn}})$. ▷ Get CNN features

  $\mathcal{L} := \|\mathbf{b} * z_{\text{cnn}} - f_{\text{bicubic}}(\mathbf{b} * z_{\text{vfm}})\|_2^2$.
  $\mathbf{b} * \theta_{\text{cnn}} := \mathbf{b} * \theta_{\text{cnn}} - \eta \nabla_{\mathbf{b} * \theta_{\text{cnn}}} \mathcal{L}$.
end for

Spacecraft Parts Segmentation

We outline our proposed semantic segmentation method for spacecraft parts before describing the baseline methods we compare against in our experiments.

Distilled Fast-SCNN

Our proposed method for spacecraft segmentation utilizes the Fast-SCNN [37] convolutional neural network due to its low compute overhead and fast inference times on edge-compute devices [15]. Because real-world spacecraft datasets are limited in size and quantity, we take an additional step to pretrain the Fast-SCNN in order to prevent overfitting. Specifically, we do this by distilling the features of a VFM (in this case, we choose Dino [36]) into our Fast-SCNN segmentation

network features by performing student-teacher training using the ImageNet-21k dataset [48], which contains approximately 14 million images of different objects, including space-related ones. After the distillation process is complete, we finetune the Fast-SCNN segmentation network on the spacecraft dataset (Sec. 6.4).

The knowledge distillation process (Algorithm 5) is as follows. Given an input image $\mathcal{I} \in \mathbb{R}^{H\times W\times 3}$, we first perform a forward pass using Dino[2], our teacher network, and extract an intermediate feature map $\mathbf{b} * z_{\text{vfm}} \in \mathbb{R}^{384\times\lfloor\frac{HW}{16\cdot 16}\rfloor}$ before reshaping it such that

$$\mathbf{b} * z_{\text{vfm}} \in \mathbb{R}^{\lfloor\frac{H}{16}\rfloor\times\lfloor\frac{W}{16}\rfloor\times 384}, \tag{6.1}$$

where $\lfloor\cdot\rfloor$ denotes the floor operator.

We then extract the intermediate feature map from the student network, Fast-SCNN[3], by performing another forward pass to get

$$\mathbf{b} * z_{\text{cnn}} \in \mathbb{R}^{\lfloor\frac{H}{8}\rfloor\times\lfloor\frac{W}{8}\rfloor\times 384}. \tag{6.2}$$

To train the Fast-SCNN, we regress its feature map to that of Dino using the mean squared error loss function, updating only the weights of the Fast-SCNN via stochastic gradient descent. Note that we only backpropagate with respect to Fast-SCNN's encoder weights, since the segmentation head is not involved in the computation of the loss. During computation of the loss, we also bilinearly upsample the feature maps such that they have the same shape.

Lastly, we train the Dino-distilled Fast-SCNN on the segmentation task. The training parameters (momentum, optimizer, learning rate) are kept consistent with the initial Fast-SCNN architecture [37].

Baselines

YOLOv8 YOLOv8 is a multitask CNN capable of performing both object detection and semantic segmentation. To create a YOLOv8 baseline for spacecraft part segmentation, we finetune the YOLOv8n-seg model variant on our custom spacecraft parts dataset, presented in Sec. 6.4, by starting from publicly-available pretrained weights to facilitate transfer learning to our space domain. We used the

[2]Dino [36] is based on a vision transformer architecture, which operates on a sequence of 16×16 or 8×8 patches from the input image.

[3]For Fast-SCNN, we update the relevant convolution kernel to have an output channel dimension of 384 in order to produce feature maps that match the channel dimension of Dino's corresponding feature map.

Table 6.1: Short-range spacecraft part segmentation model performance (mIoU) on our custom dataset.

Model	All	Body	Solar Panel	Background
YOLOv8	0.967	0.960	0.950	0.990
Fast-SCNN	0.971	0.968	0.958	0.986
Dino + Fast-SCNN (ours)	0.972	0.969	0.961	0.987

default YOLOv8 parameters with an image size of 832×832 and trained with a batch size of 16.

Fast-SCNN Fast Segmentation Convolutional Neural Network (Fast-SCNN) is a real-time semantic segmentation model built on high resolution image data (i.e., 1024 x 2048 px) [37]. This network is designed for low power devices. The main components of the network are a learning-to-downsample module, a coarse global feature extractor, a feature fusion module, and a standard classifier. The global feature extractor module captures the global context for image segmentation. Then, the feature fusion merges these features using addition.

Short Range Results

We report the results of our short-range spacecraft part segmentation algorithms. Specifically, we report the mean Intersection over Union (mIoU) on our test set. The mIoU metric is defined as the overlap between the predicted segmentation and the ground truth mask, normalized by the total area covered by the union of the two. As such, an mIoU score of 1.0 indicates a perfect prediction, while a score of 0.0 indicates a complete mismatch.

We train all models on an NVIDIA Titan RTX GPU. We train for a minimum of 100 epochs and select the best performing epoch based on the mIoU of the validation dataset. We use the Adam optimizer with a learning rate of `1e-3` and a batch size of 32. The results for training the models on the 75%-20%-5% train, test, validation dataset splits are presented in Table 6.1, while in Fig. 6.3, we present the qualitative results from this dataset.

Spacecraft Custom Dataset Results

On our custom spacecraft part segmentation dataset, we see that all models exhibit similar performance and nearly reach 1.0 (Table 6.1). Interestingly, we found no

Figure 6.3: Results of short range spacecraft component detection using RGB imaging.

benefit of model distillation in this dataset setting. This observation as well as the general high level of performance likely results from the smaller domain of our custom dataset.

Adelaide Dataset Results

We evaluate our models on the Adelaide dataset in order to quantify performance on a more difficult dataset (Table 6.2). In this setting, YoloV8 performs the best by a margin of 0.02. We can also see the benefits of model distillation when faced with a more difficult learning task as the distilled FastSCNN model outperforms the model trained from scratch by 0.06 mIoU. In Fig. 6.4, we present the qualitative

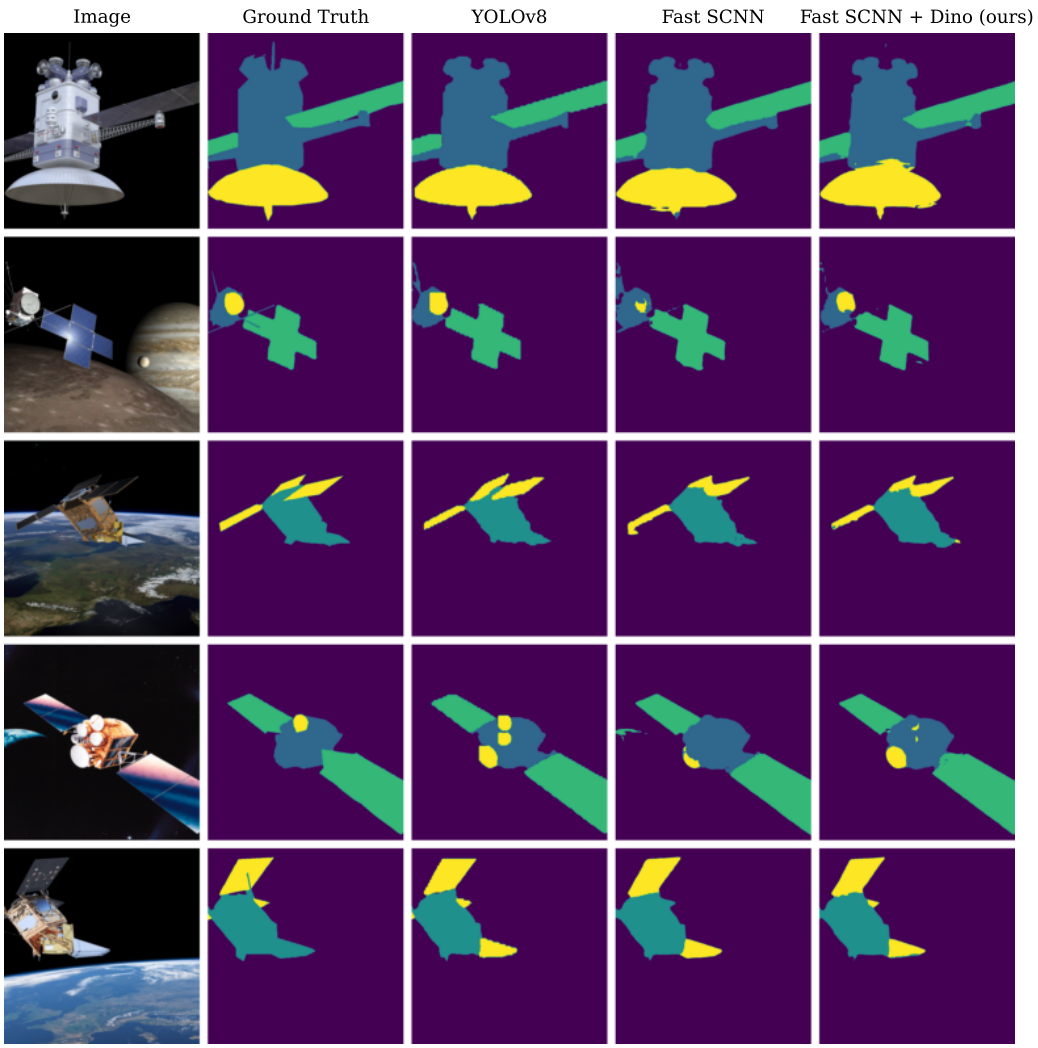


Figure 6.4: Results of short range spacecraft component detection using RGB imaging on the Adelaide dataset.

results from this dataset.

Model Distillation Ablation Study

In order to quantify the effect of our model distillation efforts for Fast-SCNN, we examine how segmentation performance is affected when fewer images are available for model training. Specifically, we examine the results when using only 75%, 50%, and 12.5% of the total training set, with 5% of the total data as validation set and 20% as testing set. We first perform this ablation study on our custom spacecraft parts dataset, as well as on the Adelaide dataset. Table 6.3 shows that the Fast-SCNN and Dino + Fast-SCNN model perform better with larger training datasets while the

Table 6.2: Short-range spacecraft part segmentation performance (mIoU) on the Adelaide dataset.

Model	All	Body	Solar Panel	Antenna	Background
YOLOv8	0.781	0.766	0.792	0.586	0.980
Fast-SCNN	0.683	0.705	0.707	0.347	0.974
Dino + Fast-SCNN (ours)	0.756	0.764	0.768	0.480	0.982

Table 6.3: Effect of training set size (% of overall training set) on short-range spacecraft part segmentation (mIoU) for the custom dataset.

Model	% of Training Data Used		
	75%	50%	12%
YOLOv8	0.967	0.967	0.958
Fast-SCNN	0.971	0.966	0.933
Dino + Fast-SCNN (ours)	0.972	0.969	0.947

Table 6.4: Effect of training set size (% of overall training set) on short-range spacecraft part segmentation (mIoU) for the Adelaide dataset.

Model	% of Training Data Used		
	75%	50%	12%
YOLOv8	0.781	0.709	0.635
Fast-SCNN	0.683	0.598	0.537
Dino + Fast-SCNN (ours)	0.756	0.722	0.636

YOLOv8 model performs consistently well despite the size of the training dataset. We hypothesize that the consistent performance of YOLOv8 is due to its extensive use of data augmentation compared to our Fast-SCNN variants.

In addition to our own dataset, we perform the same study on a more difficult dataset (Sec. 6.4) with added class complexity. [38] We show these results in Table 6.4 and Fig. 6.5. Because of the higher complexity of the dataset, all models performance tends to decrease with the decrease of the training dataset. In addition, our results (top plot of Fig. 6.5) on this dataset show that distillation noticeably increases training convergence speed and overall model performance, which could be a useful behavior for compute-limited onboard training. As such, the more complicated a dataset becomes, the more we can see benefits of distillation versus training from

scratch.

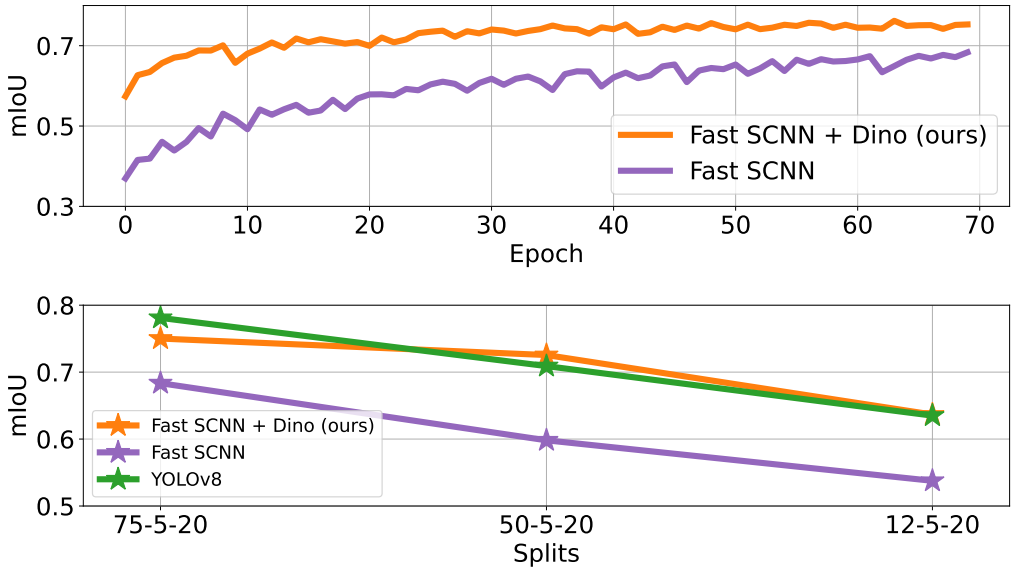


Figure 6.5: (Top) mIoU comparison between Fast-SCNN and our method (Fast-SCNN + Dino) for the 75-20-5 splits. We show that pretraining with Dino features improves performance. (Bottom) Best mIoU performance for different splits of the training dataset.

In Fig. 6.6, we investigate whether KD with Dino features increases the performance, compared to only using pre-trained weights (i.e., Cityscapes weights [49]). The results show that employing KD with Dino slightly increases the overall mIoU performance.

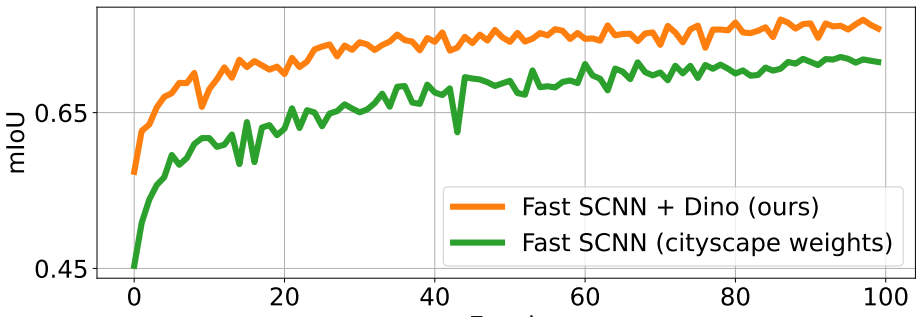


Figure 6.6: Comparison between our method and pre-training Fast-SCNN with other available weights.

Table 6.5: Compute Results. † YOLOv8 outputs binary masks per detected instance and requires an additional postprocessing step to create semantic segmentation masks.

Model	Task	Inference Time (ms)		Params (M)
		TX2	Titan RTX	
YOLOv8	Instance Segm.	78.70	6.41	3.40
YOLOv8	Semantic Segm.†	363.02	31.29	3.40
Fast-SCNN	Semantic Segm.	49.39	3.54	1.13
Dino + Fast-SCNN (ours)	Semantic Segm.	92.56	4.29	1.55

Computational Benchmarks

We benchmark the inference time of the short range detection models in Table 6.5. We report results on the Nvidia Jetson TX2 (256 CUDA cores) since its size and power usage is ideal for low-compute robotic operations. We also provide benchmarks on an NVIDIA Titan RTX (4608 CUDA cores) for comparison to a workstation GPU.

Overall, all models, besides Yolov8 (semantic segmentation), can provide predictions at a 10 Hz minimum (Table 6.5). We find that these results are suitable for real-time use onboard the Edge Node Lite mission. Fast-SCNN variants notably require less than half the storage requirements compared to YoloV8. While storage is less of a concern in terrestrial field robotics, it is vital to keep in check for when developing perception algorithms for spacecraft due to the limited bandwidth if performing model updates from Earth to space.

6.5 Flight Demonstration

The algorithm presented in Algorithm 5 was launched onboard the Edge Node Lite spacecraft. This section begins by situating our flight demonstration in the context of previous missions, then outlines the modifications required to adapt the algorithms for flight.

Context

While common in robotic ground applications, the use of deep learning on-board satellites is still in its infancy. Integrating advanced learning-based techniques directly into spacecraft offers significant advantages. For example, the ability to detect and track objects in real-time can enable autonomous collision avoidance. Similarly, on-board image segmentation can pinpoint regions of interest, reducing the volume of data transmitted back to Earth. These learning-based methods also extend to guidance, navigation, and control tasks.

However, deploying these technologies in space presents unique challenges. Satellites typically have limited computational capabilities and lack radiation-hardened components. Moreover, the space industry's cautious approach towards machine learning methods, adds another layer of complexity. In collaboration with the Aerospace Corporation, our project aims to overcome these obstacles and build a reliable track record for deep learning algorithms in space. An initial milestone involves deploying our algorithms capable of object segmentation using thermal imagery directly on-board a spacecraft.

As part of this broader strategy, the Aerospace Corporation is developing the Edge Node project, a multi-satellite, free-flying testbed designed to explore formation flight, rendezvous and proximity operations (RPO) [19]. The spacecraft is scheduled to launch in 2026-2027. Edge Node Lite is a precursor mission to Edge Node to demonstrate the key technologies that will be used in Edge Node. Edge Node Lite leverages advance compute and sensing such as an NVIDIA Jetson TX2 NX and a thermal camera, among others.

Related Work: Launched Missions with Deep Learning Demonstrations

One of the first experiments running machine learning methods on-board a spacecraft was NASA's Earth Observing-1 mission [50]. For this, a semantic texton forest (STF) which uses a random decision forest (RDF), and a Bayesian thresholding were used to classify pixels as "clear" or "cloudy". The on-board operations of the RDF and BT occurred from November 2016 through March 2017 [50].

Another mission was Intelligent Payload Experiment (IPEX), a 1U Cubesat aiming to validate technologies for onboard instrument processing and autonomous operations [52]. IPEX demonstrates the use of more complex image processing techniques such as support vector machine learning techniques, spectral unmixing,

Name	Type	Launch	Hardware for AI
NASA's Earth Observing-1 [50]	SmallSat	2000	Mongoose M5
CNES DEMETER [51]	SmallSat	2004	ADSP 21020 DSP
Intelligent Payload Experiment [52]	1U Cubesat	2013	Gumstix Earth Storm
OPS-SAT [53]	3U Cubesat	2019	Altera Cyclone V SX SoC (FPGA)
Phi-Sat-1 [54]	6U Cubesat	2020	Intel Movidius board with a Myriad II chip (VPU)
Spiral Blue's SE-Z [55]	0.25U Cubesat	2021	NVIDIA Jetson Nano
D-Orbit's ION Satellite Carrier [56]	SmallSat	2023	Intel Myriad X processor
Spiral Blue's SE-1 [57]	0.25U Cubesat	2023	NVIDIA Xavier NX
ESA's MANTIS [58]	12U Cubesat	2023	Intel Movidius Myriad 2 VPU
Ubotica's CogniSAT-6 [59]	Cubesat	2024	Intel Myriad X processor
Planet Labs Pelican-2 [60]	SmallSat	2025	NVIDIA Jetson

Table 6.6: Flown missions with deep learning demonstrations on-board the spacecraft (includes only publicly available data)

and random decision forest classification techniques. Notably, the IPEX random forest classified was able to achieve impressive results, despite being trained prior to launch using just four hand-labelled images from a high altitude balloon test.

Launched in 2004, CNES DEMETER investigated the ionospheric disturbances due to seismic and volcanic activities [51]. It used Time Delay Neural Network (TDNN) [61] for the automatic and systematic detection and characterization of all whistle-like signals. The inference was done on the ADSP 21020 digital signal processor (DSP) hardware.

OPS-SAT is the first mission to train machine learning (ML) models on-board a flying spacecraft. The mission tests 3 ML applications: (1) image classification with Convolutional Neural Network (CNN) model inferences using TensorFlow Lite, (2) image clustering with unsupervised learning using k-means, and (3) supervised learning to train a Fault Detection, Isolation, and Recovery (FDIR) [53].

Two other missions, Phi-sat-1 and D-Orbit's ION Satellite Carrier, are also using ML methods for science applications. Phi-sat-1 enabled the mission to automatically discard cloudy images and send only useful data down to Earth [54], while D-Orbit's ION Satellite Carrier is using DNN models for detection of flooding [56].

In 2024, Ubotica's CogniSAT-6 launched a spacecraft to test several deep learning applications, including cloud screening, surface water extent, algal bloom/ocean color, and land use. The spacecraft is equipped with an Intel Myriad X processor for AI applications [59].

Just one month before our flight, Planet Labs launched their Pelican-2 satellite, equipped with a NVIDIA Jetson platform [60]. Planet Labs plans to use the on-board computing to run AI algorithms, including for land cover classification and object detection. The company has not publicly stated whether algorithm training, as well as inference, will be performed on orbit. Additionally, it is currently unclear whether Planet Labs will be detecting and classifying other space objects or only ground objects.

Methods

Extension to Thermal Data

The spacecraft parts segmentation algorithm presented in Sec. 6.3 was designed for RGB images, not thermal imagery. However, the Edge Node Lite spacecraft is equipped with a thermal camera, which captures images in the long-wave infrared spectrum (LWIR). The LWIR range is particularly useful because it captures thermal emissions day and night, unlike visible or near-infrared sensors that require sunlight. Thus, the changes performed to the algorithm and architecture as are follows: (1) The feature-based knowledge distillation from the DINO VFM into the FastSCNN network is kept the same as in Algorithm 1. (2) The input image is now a single-channel image, as opposed to the three-channel RGB image. The FastSCNN network architecture is modified to accommodate the single-channel input by adding a CNN layer before the initial first layer, as seen in Fig. 6.7. (3) The FastSCNN network is retrained on a dataset of thermal images to adapt to the new domain.

Operations for On-orbit Demonstration

After a couple of weeks of commissioning, the spacecraft will be ready for the demonstration of the detection algorithms. The demonstration will be conducted in two phases: (1) **Phase 1a (Offline Data from Ground)**: To ensure our algorithms

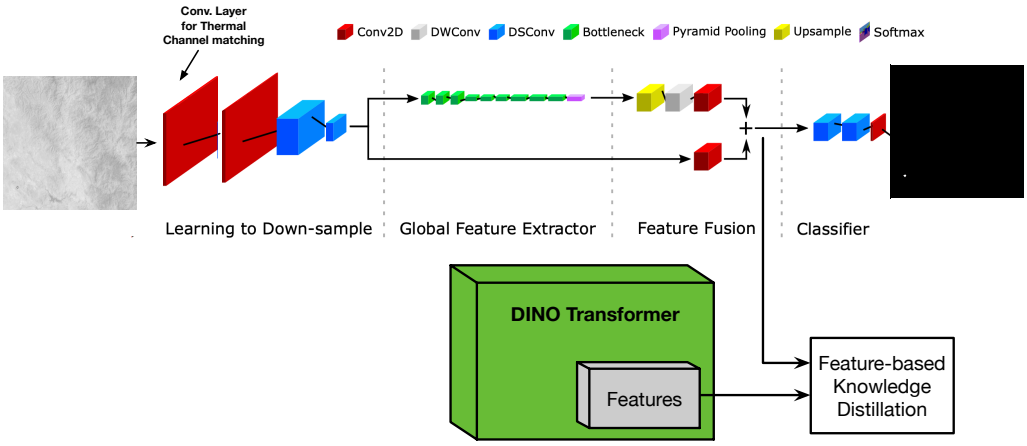


Figure 6.7: Fast SCNN architecture for training with thermal data.

are working nominally, we will run inference and training on-board the spacecraft using data already uploaded to the spacecraft's memory before launch. We will test both Dino+FastSCNN and YoloV8. The results will be stored on the spacecraft's memory and downloaded to the ground station for analysis. (2) **Phase 1b (Offline Data from Space)**: The spacecraft will capture thermal images of other spacecrafts and objects in its field of view. The images will be processed on-board using the both Dino+FastSCNN and YoloV8 networks to segment the spacecraft. The results will be stored on the spacecraft's memory and downloaded to the ground station for analysis. (3) **Phase 2 (Online Data)**: The spacecraft will capture thermal images. The images will be processed on-board using the both Dino+FastSCNN and YoloV8 networks for segmentation. The results will be transmitted to the ground station for analysis. (4) **Phase 3 (Online Data with Model Update)**: The captured images will be sent to the ground station where the models will be retrained. The new models will be uploaded to the spacecraft and retested in the online mode.

Results

Flight Hardware

The Edge Node lite computing contains a single NVIDIA Jetson TX2 NX, "which is built on a foundation of three prior qualification missions in low Earth orbit in addition to proton 50 MeV radiation testing of the target hardware" [19]. The spacecraft is also equipped with a rad-hard Flash storage and a single LWIR camera, Teledyne FLIR BOSON+ model 22640A012-6IAAX with a 12deg Horizontal Field of View (HFOV)[19].

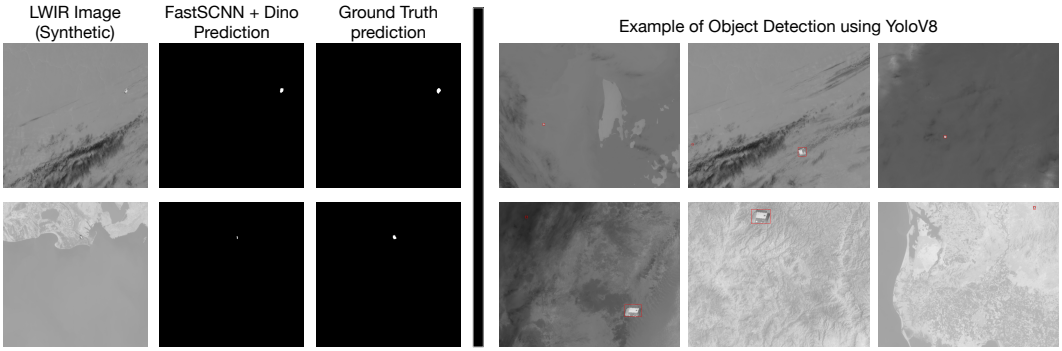


Figure 6.8: A. Segmentation of spacecraft using Dino+FastSCNN algorithm on synthetic dataset. B. Detection of spacecraft using YoloV8 algorithm on synthetic dataset.

Implementation on-board Flight Hardware

Our software is integrated into the Edge Node Lite spacecraft's NVIDIA Jetson TX2 computer. Robot Operating System (ROS2) Foxy is used as the middleware between the satellites as well as for the intra-satellite communication. We developed a ROS2 wrapper around our code, which is responsible for the following tasks: (1) Depending on the modes of operation (Sec. 6.5), the ROS2 code either reads the images from the memory or accesses the thermal camera, which publishes images. (2) The ROS2 code then processes the images using the Dino+FastSCNN and YoloV8 networks. (3) ROS2 queries the NVIDIA Jetson TX2 computer's health and status and saves it in the ROS2 bags. The inference and/or training data will be stored in ROS2 bags and transmitted to the ground station.

Preliminary Results for Hardware in the Loop (HIL) Testing

The inference and training algorithms were extensively tested on-board the Engineering Model of the Edge Node Lite spacecraft and resource usage was monitored constantly (see Appendix B). In Fig. 6.8, we show the qualitative results of the segmentation output using FastSCNN+Dino and detection output using YoloV8 on the synthetic dataset. The results show that the algorithms are able to detect and segment the spacecraft in the thermal images. Similar images as the ones shown in Fig. 6.8 were uploaded to the spacecraft's memory before launch for the Phase 1a testing (Section 6.5).

Table 6.7: Parameter sizes of the YoloV8 and DINO+FastSCNN networks that need to be uplinked and downlinked.

Model Name	Size	Uplink Time	Downlink Time
YoloV8	6.5 MB	520 s	0.26 s
FastSCNN	6.2 MB	496 s	0.25 s

Discussions and Lessons Learned

In this section, we discuss the challenges experienced during the integration of the algorithms into the Edge Node Lite spacecraft along with key lessons learned.

Model Size. The network can be trained either on the spacecraft or on the ground. If the training occurs on the ground, it is necessary to upload the updated weights and biases to the spacecraft for inference. To accommodate this requirement, we employ two lightweight networks, DINO+FastSCNN and YoloV8, which have a low number of parameters. In Table B.1, we show the size of the YoloV8 weights and biases and the FastSCNN weights and biases, and the duration it takes to uplink and downlink the data between the spacecraft and the ground station.

Domain shift. One challenge in training models for space applications is managing the domain shift between synthetic and real datasets. Moreover, accurately replicating space conditions on Earth to generate representative datasets poses difficulties. To address this, we employ knowledge distillation from the DINO VFM to the FastSCNN network, which helps mitigate the domain shift to some extent. Additionally, during the commissioning phase, we will gather data directly from the spacecraft to fine-tune the models, further enhancing their performance in actual space conditions.

On-board Data Recording. Data is recorded on-board a dual terabyte NVMe drives running a ZFS filesystem on top of radiation screened industrial NAND Flash storage drives [19]. However, this storage is shared with other experiments on-board the spacecraft. To optimize resource usage, we have implemented several measures: (1) We reduced the camera's publishing rate from 60 Hz to 1 Hz, and we only record down-sampled images. (2) We save the segmentation and detection images on a ROS2 bag, as well as the health and status of the NVIDIA Jetson TX2 computer. (3) We do not save the raw segmented masks on-board the spacecraft.

Operations. We have encapsulated the experiment within a Docker Compose container, allowing the flight computer to send commands to the Jetson TX2 to

start and stop the container. We have also significantly streamlined the onboard codebase for Dino + FastSCNN and ROS2, minimizing the volume of code deployed to the spacecraft, which allows for better testing. Among the YoloV8 and Dino+FastSCNN algorithms, the open-source YoloV8 has the most extensive code base (see Appendix B). The ROS2 code base is also minimal, as it is responsible for interfacing with the NVIDIA Jetson TX2 computer and the spacecraft's hardware.

Radiation. The Aerospace Corporation has performed radiation testing on the NVIDIA Jetson TX2 computer to ensure its reliability in space. In addition, the computer was flown on three prior qualification missions in Low Earth Orbit [19].

6.6 Chapter Summary

We present a long range detection methodology of uncooperative targets in space utilizing zero-shot detection and domain specific, single-shot detection with YOLOv8. Additionally, we present a short range detection methodology to segment the components of the uncooperative spacecraft potentially utilizing onboard training. We propose using long range detection to identify uncooperative targets, such as space debris and inactive satellites, and segmentation at shorter ranges to learn more detailed features of the targets to increase space situational awareness.

For the short range, we proposed a method for spacecraft part segmentation that leverages visual foundation models distilled into a lightweight fast semantic segmentation network (Fast-SCNN) to increase segmentation performance while keeping inference time and onboard storage requirements low to meet potential onboard training requirements. An additional benefit to distilling the Fast-SCNN from a visual foundation model such as Dino is less storage overhead, making it easier to deploy and update on an edge device in space compared to the larger YOLOv8. We are currently in the process of testing the long range detection and onboard training and segmentation on the Aerospace Corporation Edge Node Lite mission launched in 2025. [19]

BIBLIOGRAPHY

[1] Hannah Grauer*, **Elena Sorina Lupu***, Connor Lee, Darren Rowen, Benjamen Bycroft, Phaedrus Leeds, John Brader, and Soon-Jo Chung. "Vision-Based Detection of Uncooperative Targets and Components on Small Satellites". In: *38th Annual Small Satellite Conference* (2024), pp. 5321–5327.

[2] Hannah Grauer*, **Elena Sorina Lupu***, Connor Lee, Cailyn Smith, Sulekha Kishore, Darren Rowen, Benjamen Bycroft, Phaedrus Leeds, John Brader, and Soon-Jo Chung. "Vision-Based Detection of Uncooperative Targets and Components on Small Satellites". In: *ICRA Workshop on Thermal Infrared In Robotics* (2025), pp. 5321–5327.

[3] Joshua P. Davis et al. *On-Orbit Servicing: Inspection, Repair, Refuel, Upgrade, and Assembly of Satellites In Space*. 2019.

[4] Benjamin B. Reed, Robert C. Smith, Bo J. Naasz, Joseph F. Pellegrino, and Charles E. Bacon. "The Restore-L Servicing Mission". In: *AIAA SPACE 2016*.

[5] R. C. Foust, **E. S. Lupu**, Y. K. Nakka, S.-J. Chung, and F. Y. Hadaegh. "Autonomous In-orbit Satellite Assembly from a Modular Heterogeneous Swarm". In: *Acta Astronautica* 169 (2020), pp. 191–205. DOI: https://doi.org/10.1016/j.actaastro.2020.01.006.

[6] Thaweerath Phisannupawong, Patcharin Kamsing, Peerapong Tortceka, and Soemsak Yooyen. "Vision-Based Attitude Estimation for Spacecraft Docking Operation Through Deep Learning Algorithm". In: *2020 22nd International Conference on Advanced Communication Technology (ICACT)*. 2020, pp. 280–284.

[7] C. Priyant Mark and Surekha Kamath. "Review of Active Space Debris Removal Methods". In: *Space Policy* 47 (2019), pp. 194–206.

[8] *ClearSpace One*. EPFL Space Engineering Center eSpace. 2018.

[9] Wigbert Fehse. *Automated Rendezvous and Docking of Spacecraft*. Cambridge Aerospace Series. Cambridge University Press, 2003.

[10] Shin-Ichiro Nishida, Satomi Kawamoto, Yasushi Okawa, Fuyuto Terui, and Shoji Kitamura. "Space Debris Removal System Using a Small Satellite". In: *Acta Astronautica* 65.1 (2009), pp. 95–102.

[11] Saptarshi Bandyopadhyay et al. "Review of Formation Flying and Constellation Missions Using Nanosatellites". In: *Journal of Spacecraft and Rockets* 0 (2016), pp. 567–578.

[12] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. "Object Detection With Deep Learning: A Review". In: *IEEE Transactions on Neural Networks and Learning Systems* 30.11 (2019), pp. 3212–3232.

[13] Alberto Garcia-Garcia, Sergio Orts, Sergiu Oprea, Víctor Villena Martinez, and José Rodríguez. "A Review on Deep Learning Techniques Applied to Semantic Segmentation". In: (Apr. 2017).

[14] Muhammad Awais et al. "Foundational Models Defining a New Era in Vision: A Survey and Outlook". In: *arXiv preprint arXiv:2307.13721* (2023).

[15] Connor Lee et al. "Caltech Aerial RGB-Thermal Dataset in the Wild". In: *European Conference on Computer Vision*. Springer. 2024, pp. 236–256.

[16] Lu Gan et al. "Unsupervised RGB-to-Thermal Domain Adaptation Via Multi-Domain Attention Network". In: *2023 IEEE International Conference on Robot. and Autom.* 2023, pp. 6014–6020.

[17] Connor Lee et al. "Online Self-Supervised Thermal Water Segmentation for Aerial Vehicles". In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2023, pp. 7734–7741.

[18] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *Ultralytics YOLO*. Version 8.0.0. Jan. 10, 2023.

[19] Darren Rowen et al. "Edge Node: A Multi-User Rendezvous and Proximity Operations On-orbit Testbed". In: *Small Satellite Conf.* (Aug. 2024).

[20] Alexey Dosovitskiy et al. "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *arXiv:2010.11929* (2020).

[21] Ze Liu et al. "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10012–10022.

[22] Nicolas Carion et al. "End-to-End Object Detection with Transformers". In: *European Conference on Computer Vision*. Springer. 2020, pp. 213–229.

[23] Zhuang Liu et al. "A ConvNet for the 2020s". In: *Proceedings of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. 2022, pp. 11976–11986.

[24] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection With Region Proposal Networks". In: *Advances in Neural Information Processing Systems* 28 (2015).

[25] Wei Liu et al. "SSD: Single Shot Multibox Detector". In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 2016, Proceedings*. Springer. Oct. 2016, pp. 21–37.

[26] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *arXiv preprint arXiv:1804.02767* (2018).

[27] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal Loss for Dense Object Detection". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2980–2988.

[28] Mingxing Tan, Ruoming Pang, and Quoc V Le. “EfficientDet: Scalable and Efficient Object Detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10781–10790.

[29] Leo Pauly et al. “A Survey on Deep Learning-Based Monocular Spacecraft Pose Estimation: Current State, Limitations and Prospects”. In: *Acta Astronautica* 212 (Nov. 2023). arXiv:2305.07348, pp. 339–360.

[30] Nouar AlDahoul, Hezerul Abdul Karim, Angelo De Castro, and Myles Joshua Toledo Tan. “Localization and Classification of Space Objects Using EfficientDet Detector for Space Situational Awareness”. In: *Scientific Reports* 12.1 (Dec. 2022). Publisher: Nature Publishing Group, p. 21896.

[31] Chenchen Jiang et al. “Object Detection from UAV Thermal Infrared Images and Videos Using YOLO Models”. In: *Inter. J. of Applied Earth Observation and Geoinformation* 112 (Aug. 2022), p. 102912.

[32] Alexander Kirillov et al. “Segment Anything”. In: *arXiv:2304.02643* (2023).

[33] Yuepeng Liu et al. “Lightweight CNN-Based Method for Spacecraft Component Detection”. In: *Aerospace* 9.12 (Dec. 2022), p. 761.

[34] Qiang Tang, Xiangwei Li, Meilin Xie, and Jialiang Zhen. “Intelligent Space Object Detection Driven by Data From Space Objects”. In: *Applied Sciences* 14.1 (Jan. 2024). Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, p. 333.

[35] Man Yuan, Guhong Zhang, Zhuoqun Yu, Yufan Wu, and Zhonghe Jin. “Spacecraft Components Detection Based on a Lightweight YOLOv3 Model”. In: *2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*. Vol. 10. 2022, pp. 1968–1973.

[36] Mathilde Caron, Hugo Touvron, Ishan Misra, Herv’e J’egou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. “Emerging Properties in Self-Supervised Vision Transformers”. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (2021), pp. 9630–9640.

[37] Rudra PK Poudel, Stephan Liwicki, and Roberto Cipolla. “Fast-SCNN: Fast Semantic Segmentation Network”. In: *arXiv preprint arXiv:1902.04502* (2019).

[38] Dung Anh Hoang, Bo Chen, and Tat-Jun Chin. *A Spacecraft Dataset for Detection, Segmentation and Parts Recognition*. arXiv:2106.08186. June 2021.

[39] Yashwanth Kumar Nakka et al. “Six Degree-of-Freedom Spacecraft Dynamics Simulator for Formation Control Research”. In: *2018 AAS/AIAA Astrodynamics Specialist Conference*. AIAA, 2018.

[40] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the Knowledge in a Neural Network”. In: *arXiv preprint arXiv:1503.02531* (2015).

[41] Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. “Model Compression”. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’06. Philadelphia, PA, USA: Association for Computing Machinery, 2006, pp. 535–541.

[42] Huan Zhang et al. “Review of Machine-Learning Approaches for Object and Component Detection in Space Electro-optical Satellites”. In: *International Journal of Aeronautical and Space Sciences* 25 (Aug. 2023).

[43] Jiafeng Xie, Bing Shuai, Jian-Fang Hu, Jingyang Lin, and Wei-Shi Zheng. *Improving Fast Segmentation With Teacher-Student Learning*. 2018. arXiv: 1810.08476 [cs.CV].

[44] L. Wang and K. Yoon. “Knowledge Distillation and Student-Teacher Learning for Visual Intelligence: A Review and New Outlooks”. In: *IEEE Transactions on Pattern Analysis; Machine Intelligence* 44.06 (June 2022), pp. 3048–3068.

[45] Guo-Hua Wang, Yifan Ge, and Jianxin Wu. “Distilling Knowledge by Mimicking Features”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.11 (2021), pp. 8183–8195.

[46] Inseop Chung, SeongUk Park, Jangho Kim, and Nojun Kwak. “Feature-Map-Level Online Adversarial Knowledge Distillation”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 2006–2015.

[47] Viet Anh Nguyen. *AnyLabeling - Effortless data labeling with AI support*.

[48] Jia Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255.

[49] Marius Cordts et al. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[50] R. P. Schaffer, David R. Thompson, and Daniel Tran. “Cloud Filtering and Novelty Detection Using Onboard Machine Learning for the EO-1 Spacecraft”. In: 2017.

[51] Franck ELIE, Masashi Hayakawa, Michel PARROT, Jean-Louis PINCON, and F. Lefeuvre. “Neural Network System for the Analysis of Transient Phenomena on Board the DEMETER MicroSatellite”. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* (Oct. 1999).

[52] Steve Chien, Joshua Doubleday, David Thompson, Kiri Wagstaff, John Bellardo, Craig Francis, Eric Baumgarten, Austin Williams, Edmund Yee, Eric Stanton, and Jordi Piug-Suari. “Onboard Autonomy on the Intelligent Payload EXperiment CubeSat Mission”. In: *Journal of Aerospace Information Systems* 14 (Apr. 2016), pp. 1–9.

[53] Georges Labrèche, David Evans, Dominik Marszk, Tom Mladenov, Vasundhara Shiradhonkar, Tanguy Soto, and Vladimir Zelenevskiy. "OPS-SAT Spacecraft Autonomy With TensorFlow Lite, Unsupervised Learning, and Online Machine Learning". In: Mar. 2022, pp. 1–17.

[54] Gianluca Giuffrida, Luca Fanucci, Gabriele Meoni, Matej Batic, Léonie Buckley, Aubrey Dunne, Chris Dijk, Marco Esposito, John Hefele, Nathan Vercruyssen, Gianluca Furano, Massimiliano Pastena, and Josef Aschbacher. "The Phi-Sat-1 Mission: The First On-Board Deep Neural Network Demonstrator for Satellite Earth Observation". In: *IEEE Transactions on Geoscience and Remote Sensing* 60 (Jan. 2021), pp. 1–1.

[55] Tereza Pultarova. *Start-Up Spiral Blue Hopes Computers in Space Could Revolutionize Access to Earth-Observation Data*. July 2021.

[56] Gonzalo Mateo-Garcia, Josh Veitch-Michaelis, Cormac Purcell, Nicolas Longépé, Simon Reid, Alice Anlind, Fredrik Bruhn, James Parr, and Pierre Mathieu. "In-Orbit Demonstration of a Re-Trainable Machine Learning Payload for Processing Optical Imagery". In: *Scientific Reports* 13 (June 2023).

[57] Mei He. *Spiral Blue Receives First Operational Data From Space Edge Computer Mission*. Jan. 2024.

[58] Marina Mercè Cañete Muñoz, Florian Deconinck, Jordi Barrera Ars, Nicola Melega, Camille Pirat, Borja Jimeno Soto, Aubrey Dunne, Léonie Buckley, Juan Romero Cañas, Gionata Benelli, et al. "MANTIS, A 12U Smallsat Mission Taking Advantage of Super-Resolution and Artificial Intelligence for High-Resolution Imagery". In: (2024).

[59] David Rijlaarsdam, Tom Hendrix, Pablo González, Alberto Velasco-Mata, Léonie Buckley, Juan Miquel, Oriol Casaled, and Aubrey Dunne. "The Next Era for Earth Observation Spacecraft: An Overview of CogniSAT-6". In: Aug. 2024.

[60] Planet Labs PBC. *Pelican-2 36 SuperDoves Arrived in Vandenberg, California for Launch*. Dec. 2024.

[61] Alex Waibel. "Consonant Recognition by Modular Construction of Large Phonemic Time-Delay Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 1. Morgan-Kaufmann, 1988.

Chapter 7

LEONARDO - A WALKING, FLYING ROBOT. MODELING, CONTROL, AND MOTION PLANNING

This chapter is based on the publication:

> Kyunam Kim*, Patrick Spieler*, **Elena-Sorina Lupu**, Alireza Ramezani, and Soon-Jo Chung. "A Bipedal Walking Robot that can Fly, Slackline, and Skateboard". In: *Science Robotics* 6.59 (2021), eabf8136

Acknowledgments: I would like to acknowledge the co-authors of the paper for their contributions to this work, as follows: Patrick Spieler and Kyunam Kim for the development of the hardware, the hybrid walking-flying controller, manuscript writing, and the simulation results, and Prof. Soon-Jo Chung for the concept of the robot, guidance with the proofs and mathematical formulation, as well as manuscript writing.

7.1 Chapter Overview

Abstract

Numerous mobile robots in various forms specialize in either ground or aerial locomotion, while very few robots can perform complex locomotion tasks beyond simple walking and flying. We present the design and control of a multi-modal locomotion robotic platform called LEONARDO, which bridges the gap between two different locomotion regimes of flying and walking using synchronized control of distributed electric thrusters and a pair of multi-joint legs. By combining two distinct locomotion mechanisms, LEONARDO achieves complex maneuvers that require delicate balancing, such as walking on a slackline and skateboarding, which are challenging for existing biped robots. LEONARDO also demonstrates agile walking motions, interlaced with flying maneuvers to overcome obstacles using synchronized control of propellers and leg joints. The mechanical design and synchronized control strategy achieves a unique multi-modal locomotion capability that could potentially enable robotic missions and operations that would be difficult for single-modal locomotion robots.

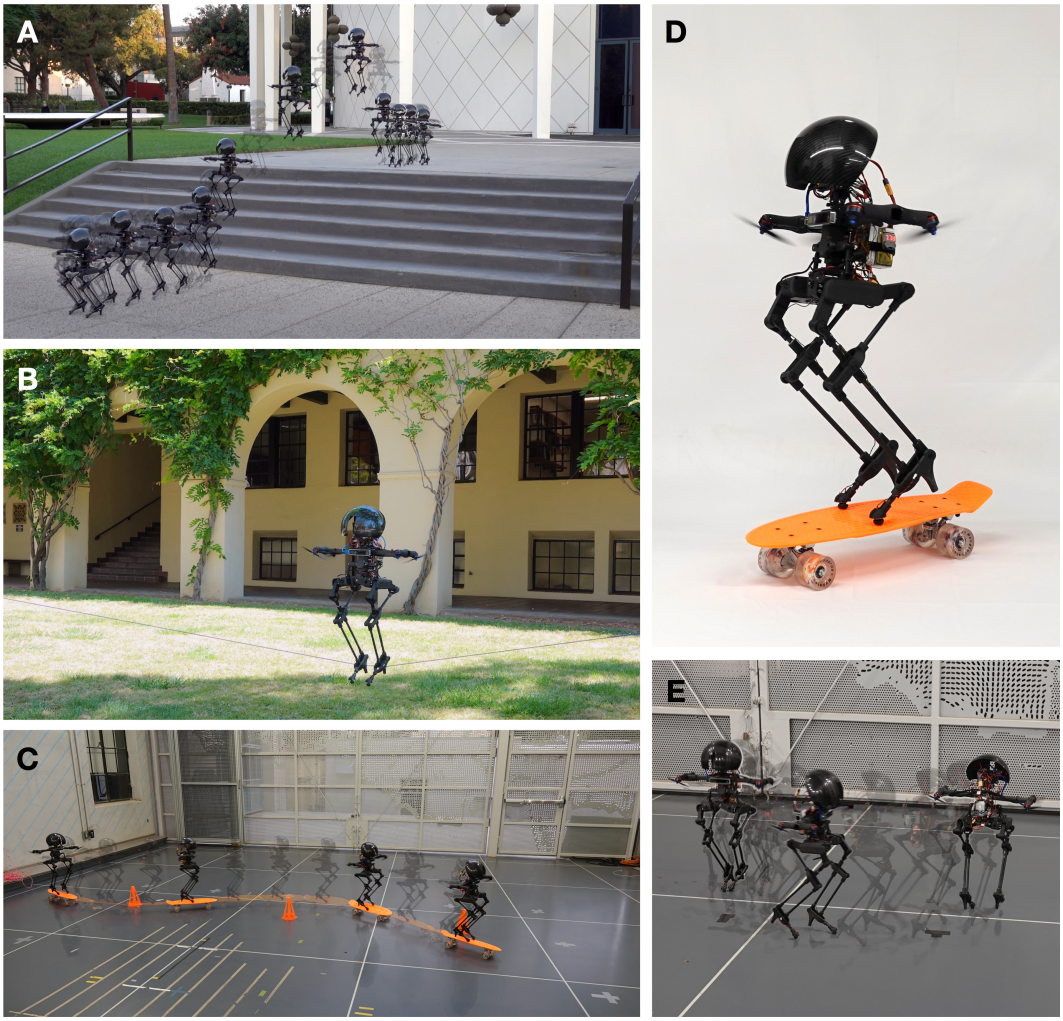


Figure 7.1: LEO robot performing a variety of locomotion maneuvers. (A) Synchronized walking and flying maneuver to traverse stairs. (B) Balancing and walking on a slack rope stretched between two trees. (C) Remote-controlled riding on a skateboard around a series of obstacles. (D) Balancing on a skateboard. (E) Walking in a semi-circle to show yawing movement capabilities.

7.2 Introduction

Many of existing mobile robots use either ground locomotion or aerial locomotion, but not many of them are capable of both. Furthermore, only a few of these robots can perform complex locomotion tasks beyond simple walking and flying. In an effort to develop such a hybrid locomotion robot, this paper presents a multi-modal locomotion robot called LEONARDO, or LEO for short, which is an acronym of LEgs ONboARD drOne (Fig. 7.1). As the name suggests, the robot has two different locomotion mechanisms: multi-joint legs and propeller-based thrusters, thereby achieving both terrestrial and aerial locomotion as well as the transition

between walking and flying. The goal of LEO is two-fold: 1) to enable robotic locomotion capabilities by leveraging its multi-modality of flying and walking, and 2) to study the underlying robot design, dynamics, and control challenges of such a hybrid robotic platform, especially at the interface between walking, take-off, and landing by using synchronous control of propellers and articulated joints.

Advantages and limitations of terrestrial robots

Numerous ground robots in various forms have been studied and developed over the past several decades, which may be categorized by their main locomotion methods: legged [1–4], wheeled [5–7], rolling [8–10], or crawling [11–13]. Bipedal robots in particular have attracted great attention not only because of their human-like body shapes, but also because they can perform versatile and robust walking, running, and jumping actions on uneven terrains [14, 15]. Some advanced humanoids can even execute high-level tasks like object manipulation, ladder climbing, or driving a vehicle [16–19]. However, the ultimate goal of bipedal robotics is to achieve human-like stability and robustness in its walking and running maneuvers over a challenging terrain or in a complex indoor environment. A great deal of research on bipedal robots exists in the literature, and interested readers are referred to a survey work on this topic such as [3].

The safe mobility of ground robots is hindered by unfavorable ground conditions and the existence of obstacles of various sizes. This limits their practical applications to be within well-structured environments such as indoor service robots [20, 21] or vacuum cleaners [22]. Such limitations tend to necessitate that they are a highly-engineered system with high development costs, as it can be seen from some examples of space exploration rovers [23] or advanced humanoid robots [24]. In addition, the applications of ground robots are restricted to movements that occur on or near the ground, as it can be challenging for them to gain access to an elevated location.

Advantages and limitations of aerial robots

The aforementioned difficulties can be overcome with aerial robots that fly over obstacles of any sizes or grounds of any conditions, and such robots have been another mainstream area of research in the robotics community. Both fixed-wing [25–27] and rotary-wing aircraft [28–30] as well as their hybrid types [31, 32] have been studied extensively and are proposed for real-world applications such as remote sensing, delivery, search and rescue, surveillance, and real-time monitoring [33]. How-

ever, these robots come with their drawbacks, including large energy consumption, short flight and operation time, and limited on-board resources and payload weight. Furthermore, aerial robots have more difficulty than ground robots in physically interacting with their environment or other robots because they need to stabilize themselves in mid-air. Therefore, aerial robots have been mainly envisioned for applications that do not or minimally involve physical interactions such as visual inspections, although aerial manipulation has recently become a popular research topic [34–37].

Multi-modal locomotion and hybrid terrestrial and aerial robots

Robots with a multi-modal locomotion ability possess advantages over robots that have only a single mode of locomotion, such as moving through challenging environments by appropriately switching between available locomotion modes or having flexibility with the execution of their missions. However, the realization of such multi-modal locomotion robots in practice has been a challenge. Further studies are needed to resolve the wide range of research issues, such as mechanical design, modeling and analysis, control system design, manufacturing, and experimental validation.

While some previous works have presented terrestrial and aquatic locomotion abilities [38, 39], others have attempted to develop hybrid ground and aerial locomotion robots. Some of these robots adopt a fixed-wing to take advantage of its endurance and efficiency combined with wheel-legs [40] or whegs [41] to enable ground locomotion. Arguably, a challenge for these robots is to successfully transition from ground locomotion to flight by accelerating their forward speed to make their wing-borne lift force large enough for take-off. Several approaches have been proposed to address this issue, including rooftop take-off [42], gliding after jumping with spring-loaded legs [43], or adding propellers to enable vertical take-off [44], at the expense of increased mechanical complexity.

To overcome the difficulty of ground-to-air transitions, hybrid locomotion robots using rotary wings for aerial locomotion have been developed. Although there has been an effort to invent a transformable robot that folds/unfolds a tail mechanism and rotor blades to fly similar to a helicopter [45], many prior works in this direction adopt a multi-rotor actuation mechanism. For example, the rolling cage [46] and the dynamic flying-walker [47] achieve ground locomotion with propeller thrust. However, the proposed designs exhibit limited versatility and adaptability on rough

terrains. Others utilize actuated wheels [48, 49], as they are mechanically simple, but they too have limited adaptability on complex terrains.

Some hybrid robot designs enable impressive locomotion maneuvers. For example, a quadrotor equipped with a climbing mechanism achieves the ability to perch on a wall from flying, climb of the wall, and take off from the wall to return to flying [50]. In [51], a four-wheeled ground robot with two tiltable propellers demonstrates ground locomotion and wall-climbing as well as the transitioning between the two. An interesting concept of a lighter-than-air flying robot with legs was proposed with the goal of improving bipedal walking stability using a helium filled balloon, but its buoyancy was not sufficient for flying, and the robot was susceptible to wind [52].

Biological motivation for bi-modal locomotion

While bio-inspired aerial robots [53–55] have been focused only on controlled flying maneuvers, terrestrial and aerial bi-modal locomotion is a form of locomotion that is commonly found in animals such as insects, bats, and birds. Birds, in particular, fly to move a long distance to forage, to flee from predators, or to migrate for a better climate. Still, their multi-purpose legs are what drove their evolutionary success as they enable multiple movement modalities on the ground that are essential for terrestrial foraging like walking, running, jumping, resting, and perching [56].

A complex yet intriguing behavior happens at the transitional interface of the birds' two locomotion modes. Several robot designs have attempted to adopt the ideas and mechanisms from nature for their air-surface transitions [57]. Furthermore, it is recognized in the literature that some birds use legs to provide a substantial amount of thrust for take-off [58], which motivated and were applied to the development of the airplane [59]. As such, one direction of studying bi-modal locomotion robots is to take bio-inspired approaches to facilitate their development based on scientific observations of birds. Moreover, bi-modal locomotion robots have also been used to investigate the potential evolutionary process of bird locomotion [57]. For instance, a hexapedal robot was appended with flapping wings not only to improve its running speed on the ground but also to provide an insight into the origin of bird flight [60].

Filling the gap between ground and aerial robots

LEO aims to bridge the gap between the two disparate domains of aerial and bipedal locomotion that are not typically intertwined in existing robotic systems. LEO targets versatile tasks that need to be accomplished in places that are difficult to reach by ground robots. Compared to a ground robot, LEO can overcome any

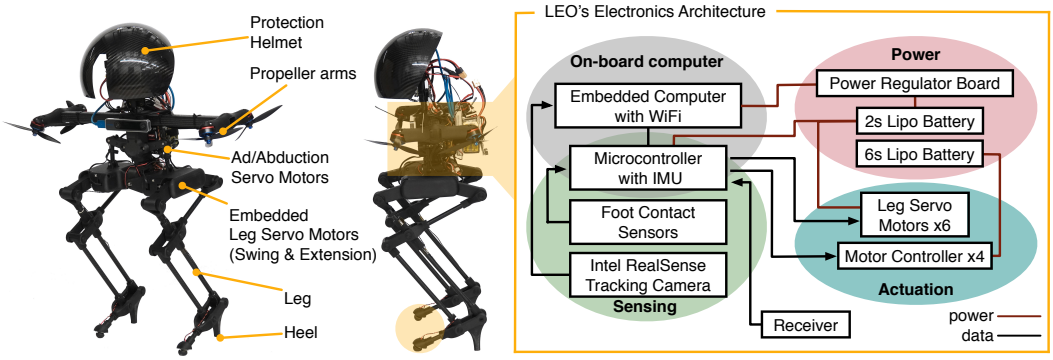


Figure 7.2: Main electronics and mechanical components of LEO.

obstacle using the transition between its ground and flying modes or easily reach an elevated location using propellers. In addition, while an aerial robot can hover over targets, LEO can use its ground locomotion to approach them for closer inspection. When LEO is in ground contact or walking, it can also reject large disturbances and prevent falling even on extremely slippery surfaces by using synchronized control of its distributed propellers and leg joints, thereby allowing for more robust and precise walking motion.

By leveraging its unique design and control concepts, LEO successfully demonstrates walking and standing on a loosely-tensioned rope (Fig. 7.1B), a feature which is often seen in nature, as exhibited by birds walking on high voltage or telephone lines. Furthermore, we demonstrate that LEO is capable of riding a passive wheeled skateboard, which is a nontrivial maneuver requiring a mastery of balancing even for humans, not to mention biped robots (Fig. 7.1C).

7.3 Results

Robot system overview

LEO has a weight of 2.58 kg and an overall height of 75 cm when walking. LEO is comprised of three main subsystems, namely, a torso, a propeller propulsion system, and two legs with point feet, as depicted in Fig. 7.2. LEO can operate completely autonomously with its on-board computers and sensor suite. The nominal walking speed of LEO is 20 cm/s, while its overall ground speed can substantially increase by using intermittent flying while close to the ground.

Legs and feet

The LEO legs are designed and built to be lightweight with a low moment of inertia, enabling agile walking and reducing the required propeller thrust to carry the legs

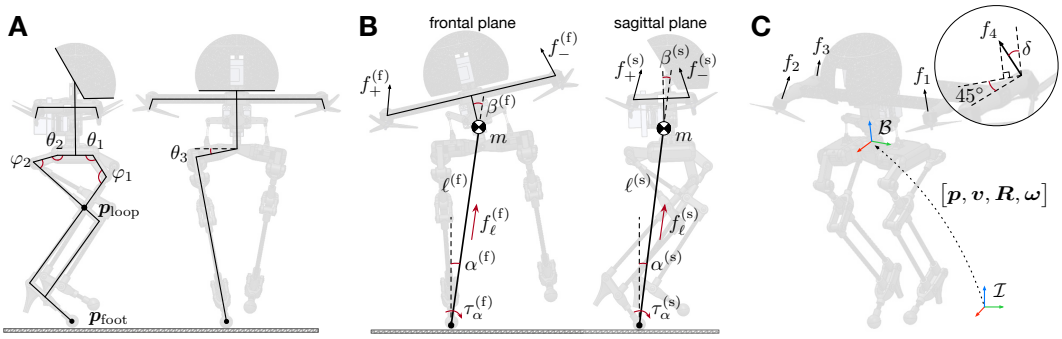


Figure 7.3: The dynamical model of LEO. (A) Leg geometry with controlled joint angles θ_1, θ_2, and θ_3, and passive joint angles φ_1 and φ_2. (B) Planar Inverted Pendulum model with propellers. Here, α^i with $i = \{(\text{f}), (\text{s})\}$ is the angle between the virtual leg and the vertical axis, β^i is the torso angle with respect to the virtual leg, and ℓ^i is the length of the virtual leg. f_ℓ^i is the leg extension force and τ_α^i is the moment about the pivot point generated by the lumped propeller thrusts f_+^i and f_-^i. (C) The body frame $\mathcal{B}$ is fixed to the torso with the origin at the nominal CoM. Its location and orientation with respect to the inertial frame $\mathcal{I}$ are given by $\boldsymbol{p}$ and $\boldsymbol{R}$, respectively. Their time derivative in the inertial frame are the linear and angular velocities $\boldsymbol{v}$ and $\boldsymbol{\omega}$. The four controllable thrust forces $f_1, ..., f_4$ are at a fixed tilt angle $\delta = 25°$ with respect to the torso vertical axis. This tilt of the four thrust axes is directed inwards at a 45° angle in the horizontal plane with respect to the forward direction. A vector description of the thrust axes is given in the Supplementary Text.

as a payload during flight. To achieve this, the leg structures are constructed with carbon fiber tubes and 3D printed carbon fiber reinforced nylon joints holding ball bearings. The leg geometry is designed to have two closed kinematic loops. This parallel mechanism allows the leg actuators to be placed close to the torso, resulting in a compact form with reduced leg inertia.

Both legs are symmetric, and each leg has three servo motors for actuation. The first one is located at the pelvis and moves the leg structure within the frontal plane of LEO. The other two servo actuators are located at the front and back of the hip and drive the parallel leg mechanism (see Fig. 7.3A). In combination, these two actuators drive the leg length and swing in a forward direction. The servo actuators driving the legs of LEO are integrated brushless DC (BLDC) motors with high reduction gearboxes and embedded position control. The use of these integrated servo actuators minimizes the system size and weight, which are critical aspects for a flying robot. In addition, the embedded position controller is precise and fast enough for LEO's leg control due to the low inertia of the legs and the high bandwidth of the propeller controller that stabilizes the walking gait.

Regarding the design of the feet, there is a half-sphere of polyurethane rubber at the end of each leg, serving as a point foot with a high friction coefficient to prevent slipping during standing or walking. The half-sphere rubber foot sits on top of an on/off switch, which allows LEO to determine when ground contact is made during normal walking or when LEO switches its locomotion mode between walking and flying. Lastly, heels are installed such that LEO can rest on them without requiring active balancing, thereby allowing LEO to power down the propellers. When initiating normal walking, LEO raises itself from the lowered resting position on the heels. This motion lifts the heels to allow them to clear the ground during normal walking on a flat terrain.

Distributed propellers and motors

LEO has four symmetrically placed propellers at its shoulders, which are used for stabilizing and controlling both the walking and flying maneuvers. The propeller axes of rotation are selected to enable the generation of roll, pitch, and yaw moments in both positive and negative directions about the CoM for flying and about the current stance foot location for walking. Specifically, the propeller axes are tilted by the angle of $\delta = 25^\circ$ inwards from a vertical axis (see Fig. 7.3C). Tilting the propeller thrust forces inwards increases the moment the propellers can produce about the ground contact point, which improves the controllability for balancing and walking motions. However, there is a trade-off to be made, since higher tilt angles also reduce the net vertical thrust and flight efficiency. A further analysis of this trade-off can be found in the Supplementary Text. It is also possible to vary the tilt angles of the propellers with additional actuators while LEO is walking or flying in order to improve its performance. However, this is not done for LEO, as this feature will add extra weight and complexity. Each 6-inch diameter unidirectional 3-blade propeller has a fixed pitch, and its thrust is controlled via its rotation speed. The propellers are directly driven by small BLDC motors, which are mounted on the arms fixed to the torso.

Torso design and electronics components

LEO's torso not only connects the arms and legs into one assembly but also houses on-board computers, sensors, and two lithium polymer (LiPo) batteries for powering the system (Fig. 7.2). LEO is equipped with two on-board computers. The first computer, a NanoPC-T4, has higher computational power and runs Robot Operating System 2 to interface with sensors, to receive commands, to estimate states based

on sensor readings, and to compute the desired trajectories. The second computer, a microcontroller with a Real-Time Operating System, runs our nonlinear walking controller and the attitude flight controller at an update rate of 200 Hz. It interfaces directly with the leg actuators and the propeller motor controller as well as an inertial measurement unit (IMU) for low latency attitude information. The on/off switches embedded into the feet are used to detect foot-ground contact and are wired to the microcontroller.

Mass integration metric

The level of integration of a system having multiple modes of operation is quantitatively measured as the *mass integration metric*, denoted as I_{mass}, which is defined as the ratio of the summation of masses required to realize the modes without integration to the total integrated robot mass [61]. It is defined for LEO as

$$I_{\text{mass}} \coloneqq \frac{m_{\text{fly}} + m_{\text{walk}}}{m},$$

where m is the mass of LEO, and m_{fly} and m_{walk} are the masses of components of LEO associated with flying and walking, respectively. The metric $I_{\text{mass}} \in [1, 2]$, where 1 means no integration and 2 means complete integration between the two modes. For LEO, the metric is computed as $I_{\text{mass}} = 1.39$, which shows that a moderate amount of LEO's components are shared for the two modes of locomotion.

Synchronized leg and propeller control

LEO's locomotion modes can be grouped into two main categories, ground and aerial locomotion, and each mode runs a dedicated feedback controller. The ground and aerial locomotion controllers are switched depending on the measured contact state of the two feet. LEO runs a state machine that ensures that these deliberate transitions occur reliably without switching back to the previous state even when the contact sensing is noisy (e.g., during landing). The walking controller further distinguishes between single and double stance phases, again based on the foot contact sensors.

All ground locomotion types (i.e., walking on the ground, skateboarding, walking on a slack rope, and balancing) make use of the same walking controller but with different input trajectories. In all these cases, the legs carry the majority of LEO's weight and control LEO's position by commanding the leg servo motors to follow joint angle trajectories derived from a walking gait trajectory. The propellers stabilize the walking gait by taking into account the foot positions as well as the foot

contact states. This creates a synchronized control action between the legs and the propellers. Since the servo motors do not provide any feedback, the foot position estimates are taken from the reference trajectory.

For the flight mode, LEO is controlled as a standard quadrotor using only the four tilted propellers. An overview of the proposed controller architecture for LEO is shown in Fig. 7.9. In the Materials and Methods section, each component of this control architecture is elucidated in detail.

7.4 Experimental Validation

We performed extensive testing of LEO in various scenarios to best show its versatile locomotion capabilities (see Fig. 7.1). Because of its on-board stereo camera for localization, the robot can be tested both outdoors and indoors. These scenarios included walking on a flat terrain, synchronized flying and walking, and walking and balancing on a slack rope. In addition, we performed several experiments to showcase its high degree of maneuverability. For example, LEO was able to ride a skateboard around traffic cones by using the thrust of its propellers.

Thruster-aided ground locomotion

In this experiment, we demonstrated LEO's ground locomotion capability. The robot started from a resting position on its heels, with the propellers in an idle mode (Fig. 7.4A, i). In this mode, the propellers could also be turned off. LEO then stood up using its leg servo motors and, at the same time, used its propellers to balance (Fig. 7.4A, ii). Afterward, the robot started walking (Fig. 7.4A, ii→ iv) and entered a periodic walking gait over the flat ground (Fig. 7.4A, v→ viii).

The same experiment was repeated multiple times within a motion capture area to analyze the robot's motion. Note that the motion capture system was not used for feedback control; rather, it was used only to record the torso poses and the positions and orientations of both feet. The tracking data of 11 runs over a sequence of multiple steps is presented in Fig. 7.5A,B. In Fig. 7.5C, the trajectory of one of these runs is visualized in 3D.

The six plots in Fig. 7.5A show the left and right feet locations with respect to the Center of Mass (CoM) in the body axes. The dashed line marks the desired foot location. In all axes, we notice some delay between the commanded and actual leg positions caused primarily by the limited tracking performance of the lightweight servo motors. We also observe that, particularly for the x-axis, the delay increased

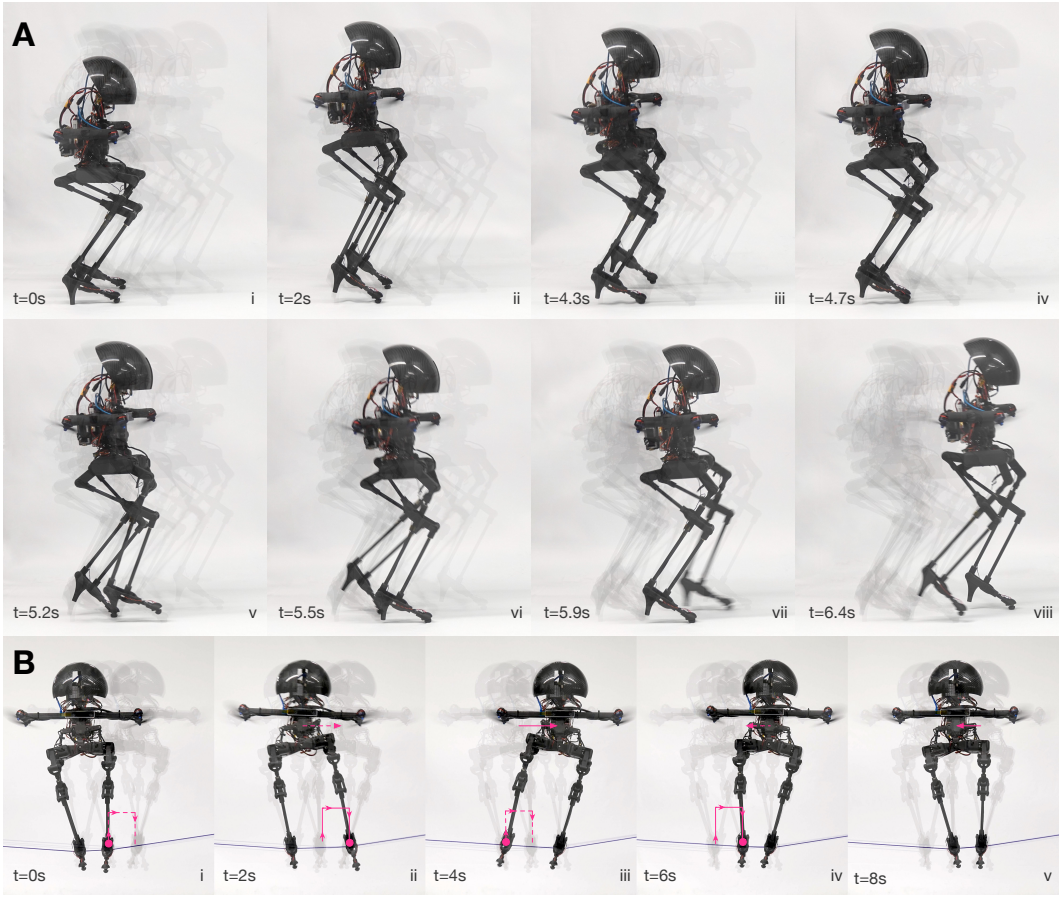


Figure 7.4: Walking sequences. (A) LEO standing up from its resting position on the heels and starting a periodic walking gait. (B) LEO traversing a flexible rope. Dashed arrows indicate the movement from the current to the next snapshot in sequence and solid arrows show the executed movement since the previous snapshot.

slightly as soon as the leg touched the ground because the servo motors experienced more load.

Another aspect captured across the data in all axes is the leg deformation of the robot. In the y-axis, the error increases whenever the robot is standing on the leg, especially for the left one. This error stems from the bending deformation of the leg structure in the frontal plane. In the z-axis, there is an oscillation visible whenever the leg impacted the ground due to the overall elasticity of the leg structure. To reduce such leg deformation during walking, we commanded the propellers to generate a minimum thrust of 40% of the body weight to lessen the loads on the legs.

Figure 7.5B shows the walking trajectory tracking errors of the nonlinear controller in the sagittal and frontal planes. The details of the controller are presented in the subsection below titled "Nonlinear tracking controller for walking using propellers."

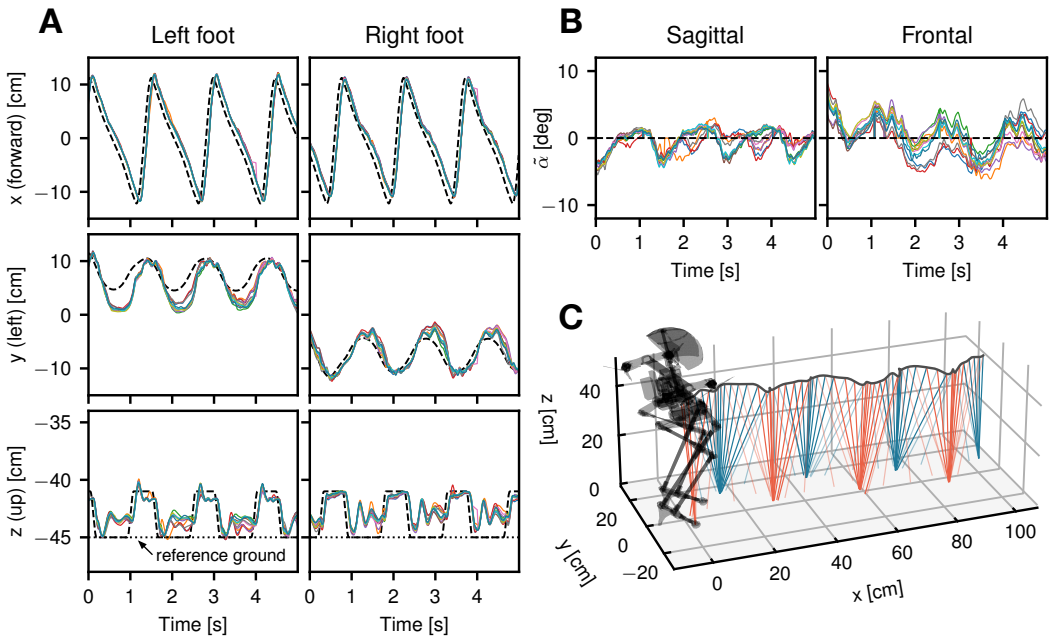


Figure 7.5: LEO walking at 0.2 m/s over flat ground. (A) Left and right feet positions tracked by a motion capture system for 11 runs. The coordinates are in a Forward-Left-Up body frame with respect to the CoM. The dashed lines are the reference foot trajectories. In the z-axis plot, the ground level of the reference trajectory is shown as a dotted line. The deviations of the foot trajectories from the reference ground level during stance phases are caused by compression of the leg, which manifests as an offset since the foot position is plotted relative to the CoM. (B) Tracking errors of our nonlinear controller in sagittal and frontal planes, $\tilde{\alpha}^{(\mathrm{s})}$ and $\tilde{\alpha}^{(\mathrm{f})}$, for the same runs (see Fig. 7.3 for a free-body diagram of LEO). (C) 3D visualization of a single run. The line segments connect foot positions to CoM position at regular time intervals. The transparent line segments indicate that the foot sensor detects no contact. Figure generated by Patrick Spieler and Kyunam Kim.

The frontal tracking has a root mean square (RMS) error of 2.6° median across all the runs. In contrast, the sagittal tracking error is lower, with a median RMS error of 1.6°. This difference occurs because the frontal plane controller has lower feedback gains than the sagittal one, since the legs were more flexible in that direction, which induced unwanted oscillations for higher gains.

Another experiment was performed to demonstrate the yawing movement capability of LEO as depicted in Fig. 7.1E and Video 1 in [62], where LEO was tracking a semi-circle trajectory and the yaw control was achieved using propellers.

Robustness to external disturbances on the ground

To further demonstrate LEO's strong robustness to external disturbances, additional experiments were performed. In the first experiment, LEO used the nonlinear tracking controller with a constant attitude setpoint standing upright on one leg, and it was pushed with a stick multiple times in the sagittal and frontal planes, as seen in Fig. 7.6B. The robot successfully rejected the disturbances, as presented in the top plot of Fig. 7.6A by applying the thrusts shown in the bottom plot. Video 2 in [62] shows this experiment as well as the same experiment repeated on a slippery surface (a whiteboard with oil), where LEO starts sliding when disturbances are applied, but it does not lose balance.

The ground locomotion experiments were mainly focused on flat terrain conditions. The main limitation with walking on a highly rough terrain comes from the limited range of motion of the legs, not the stability of the controller. The current leg kinematics design of LEO allows for about 4 cm of ground clearance, thus traversing rough terrain cannot be achieved safely with this leg geometry. An improved leg design that allows a more extensive range of leg motions and more ground clearance should allow LEO to walk over a rough terrain by leveraging the stabilization capability of propellers. With the current design, however, LEO can still overcome the rough terrain by flying over it.

To compare LEO's disturbance rejection capability against a flying vehicle, we placed both LEO balancing on one leg on a rigid table and a standard commercial drone in front of a wind tunnel. We then switched on the wind tunnel to a speed of 3.8 m/s. While the drone was immediately blown back at this wind speed, LEO could withstand this disturbance thanks to the frictional ground contact (see Video 3 in [62]), which demonstrates that LEO is more resistant to wind when it is in contact with a rigid surface.

Synchronized walking and flying

In this experiment, we showcase LEO's capability to combine walking and flying to overcome obstacles. We performed the experiment in an indoor laboratory environment with the robot tethered for safety precautions (Fig. 7.6D and Video 4 in [62]). Two tables were placed about 1 m apart, and LEO traversed them by flying over the gap. LEO followed a pre-defined trajectory since the stereo camera was not used to detect the gap or the tables, but only provided navigation information to the robot. We present the actual and desired positions and velocities of LEO together

with the propeller thrusts in Fig. 7.6C. LEO transitioned from walking to flying by increasing its collective propeller thrust until both feet lost ground contact (around $t = 8s$). At that point, an upward velocity was initiated and LEO climbed away from the ground. Then, the robot followed a smooth flying trajectory up to the landing point. The forward landing velocity was matched to the pre-determined walking speed, and the walking phase was again triggered when one foot touched the ground. After the touchdown, the robot continued to walk on the second table. In addition, a similar experiment was performed without the safety tether, flying down from a single table, as shown in Video 5 in [62].

We also validated the synchronized walking and flying control by performing outdoor experiments where LEO flew down a set of stairs, as seen in Video 6 in [62]. A sequence of images overlaid to form a single image showing this motion is illustrated in Fig. 7.1A. This experiment demonstrates that LEO can execute controlled maneuvers that combine walking, take-off, flying, and landing with smooth transitions between flying and walking gaits.

Thruster-aided balancing and walking on a slackline

Inspired by the fact that birds can walk and balance on a loosely-tensioned rope, we designed several experiments to show LEO's balancing and walking capabilities on an elastic rope stretched between two trees (Fig. 7.1B). The rope was placed along the arch of LEO's feet. LEO was able to balance and walk sideways successfully using the same nonlinear balancing controller employed for walking on the ground. However, in this configuration, the feet contact sensors cannot detect when the robot was standing on the rope. Therefore, we overrode the feet contact sensors and feet position information in the controller with a fixed virtual foot contact centered just underneath the robot. The controller was robust enough to perform well even though the actual foot location and contact states are different from the controller estimates.

A sequence of images showing LEO walking sideways on the rope is displayed in Fig. 7.4B. The walking trajectory is a periodic sequence, which alternates between shifting the CoM from one leg to another (Fig. 7.4B, ii→ iii, iv→ v) and advancing a leg (Fig. 7.4B, i→ ii, iii→ iv). In this trajectory, the CoM is always kept above the stance foot or line connecting the two feet in a double stance phase. The position of the robot, tracking error, and the generated thrust from the propellers are plotted in Fig. 7.7 in the Supplementary Text. A movie of the experiment can be found in Video 6 in [62].

Figure 7.7 presents the experimental data for walking on a loosely-tensioned rope. The top plot shows the CoM x position (horizontally away from the rope) and z position (upward), each plotted against the y position (along the rope). The rope is deflected downward under the weight of the robot, which makes the mean z position increase as the robot moves from the center of the rope toward the higher attachment point. A back-and-forth movement is seen in the y direction as the robot takes each step, shifting its CoM from one foot to the other and back, as described in the trajectory of Fig. 7.4. The middle plot shows the evolution in time of the tracking error in the sagittal and frontal planes. We notice periodic disturbances in the frontal plane caused by the foot advancing to a point on the pulled-down rope, which is at a higher position relative to where the robot stands. The sagittal plane has a small tracking error compared to the frontal plane because there is no disturbance, since the steps are made in the frontal plane. The bottom plot shows the commanded propeller thrust signals with respect to time.

Riding a passive skateboard

In this experiment, LEO was riding on a passive skateboard. The skateboard controller used the same nonlinear walking controller as in the previously mentioned experiments, since this controller can track arbitrary body and leg orientations using control of distributed propellers. Riding the skateboard was decoupled into two control problems: controlling the steering angle and controlling the skateboard's forward acceleration/deceleration. On a skateboard, steering is achieved by tilting the board around its longitudinal axis. This, in turn, rotates the skateboard's wheel-axis, which is mounted on an angled, spring-loaded joint. We placed LEO on the skateboard with its feet spaced apart in the lateral direction. An extension of its legs tilted the skateboard and allowed steering.

The forward acceleration was achieved by moving the CoM of the robot backward while simultaneously pitching the body forward. When in a steady state, the controller applied a moment that canceled the gravitational moment of the off-centered CoM to maintain balance in this state. This moment was realized with the rear propellers generating more thrust, which resulted in the robot and the skateboard being pushed forward. Deceleration is achieved in an opposite fashion by leaning forward and pitching its body backward. This method ignores the transients of moving the body into the leaning positions during which reverse forces would be produced, thereby limiting the control bandwidth. However, this controller was adequate for direct manual control.

One result of experimentation is shown in Fig. 7.1C and in Video 8 in [62], where LEO rode on the skateboard and slalomed around a series of traffic cones. LEO was driven manually via a remote control by directly controlling the desired acceleration and steering angle. In this composite image (Fig. 7.1C), a snapshot is taken at every 1.5 seconds. The figure shows the acceleration phase in the beginning, deceleration in the end, and the tilted skateboard for steering.

Control input signals during walking and flying

As a hybrid walking-flying robot, it is important for LEO to have a sufficient amount of control authority both on the ground and in the air. Specifically, LEO's propellers must be able to provide sufficient moments in all three axes for balancing when walking and for attitude control when flying, while guaranteeing that a desired range of thrust is attainable as the moment and thrust capabilities are coupled together. If the desired thrust and moments computed from the controller require the propellers to generate thrusts exceeding their limit, saturation will occur, which could lead to control system failure. In addition, if friction or normal force constraints during ground locomotion are not met, the robot could start sliding or take off. In order to verify that our choice of LEO's design parameters provides sufficient control capability, we investigate control input signals from our experiments and compare them with LEO's control capability. This analysis can be seen in [62].

7.5 Discussion

LEO's position as a multi-modal locomotion robot

Recently, robots using both propellers and legs, similar to LEO, have been proposed in the literature. For instance, the monopedal robot in [63] was initially built as an agile vertical jumping robot, but later, two thrusters were added [64, 65] to control its attitude in mid-air to enable multiple high precision jumps in series. Another robot with four propellers and two legs has been introduced with the goal of demonstrating a walking appearance, which was referred to as *pseudo-locomotion*, for entertainment purposes [66]. While the configuration of the proposed robot is similar to that of LEO, this robot mainly uses propellers, not legs, for its movement, and its multi-modal locomotion ability has not been demonstrated. In [67], a pair of coaxial propellers are added to a biped robot to assist its stable walking, and hence, the work is not focused on investigating multi-modal locomotion aspects, especially for take-off and landing maneuvers.

It might appear as if the designs of the aforementioned robots shared a robot concept

similar to LEO; however, we note that the design philosophies that these robots are based on are fundamentally different from each other due to their clearly different objectives. Specifically, the robot in [66] is not designed for dynamic walking, and its design and control do not consider the robot's interaction with the ground. In addition, the robot in [65] focuses only on precision robotic hopping whereas the robot in [67] is designed only for biped walking, and hence, their propellers are not intended for flight maneuvers. On the other hand, LEO's design enables dynamic biped walking with complex ground interaction, while preserving the flight performance of a multi-rotor vehicle. Therefore, we believe LEO truly integrates bipedal locomotion with flying. LEO demonstrated that synchronized propeller control greatly improves the stability of inherently unstable bipedal walking even within extreme scenarios in which other bipedal robots usually fail, such as walking on a rope. Furthermore, successfully riding a wheeled skateboard can be regarded as LEO's achieving of an additional type of locomotion modality.

Energy efficiency and Cost of Transportation of LEO

The extreme balancing ability of LEO comes at the cost of continuously running propellers, which leads to higher energy consumption than legs-only ground robots. However, this stabilization with propellers allowed the use of low-power leg servo motors and lightweight legs with flexibility, which was a design choice to minimize the overall weight of LEO to improve its flying performance.

We note that the optimization of LEO's energy consumption was not a priority in this work. However, we investigated its Cost of Transportation (CoT), a dimensionless quantity characterizing the energy efficiency of locomotion, defined as $\text{CoT} \coloneqq \frac{P}{mgv}$, where P is the power consumption of the robot whose mass is m while moving with a constant velocity v measured at the standard gravitational acceleration g. For LEO, the measured CoT was 108 when walking at a speed of 20 cm/s. When flying at 1 m/s, the CoT was 48, and it decreased to 15.5 at the flight speed of 3 m/s. Further information on the energy consumption and a comparison with other biological systems and robots are provided in the Supplementary Text.

There are possible ways to improve the energy efficiency by making different design trade-offs. For instance, LEO with more rigid legs would require less thrust for preventing excessive leg deformation. Furthermore, LEO could walk with the reduced support from the propellers by adopting finite feet for better stability or higher power motors with torque control for joint actuation that would allow for fast

and accurate enough foot position tracking to stabilize the walking gait. In such a case, propellers may need to turn on only when the legs fail to maintain stability on the ground without having to run continuously. These solutions would cause a weight increase and lead to a higher energy consumption during flight maneuvers, but they would lower energy consumption during walking. In the case of LEO, we aimed to achieve balanced aerial and ground locomotion capabilities, and we opted for lightweight legs. Achieving efficient walking with lightweight legs similar to LEO's is still an open challenge in the field of bipedal robots, and it remains to be investigated in future work.

Although energy efficiency is not the focus of the LEO design, the Cost of Transportation (CoT) was analyzed to present the current limitations of such a hybrid locomotion system and to inform future researchers interested in this direction about potential challenges that have to be addressed.

While hovering, LEO consumes an average of 992 W, out of which 933 W are powering the propellers and 59 W are powering the on-board electronics and leg actuators. This power consumption is almost cut in half when LEO is walking on the ground, drawing an average of 544 W, which is split between 445 W for propellers and 99 W for electronics and legs. These power measurements were made by measuring the energy required to recharge LEO's battery after performing a walking or flying maneuver. Therefore, they include the overall power consumption as well as the battery charge/discharge losses. With the relatively small batteries used on LEO, the resulting flight endurance is about 100 seconds and the walking endurance is about 3.5 minutes. The limiting factor is the 29 Wh capacity of the battery powering the propellers.

In Fig. 7.8, the CoT for different animals, insects, and robotic systems as well as LEO during its two main locomotion modes are plotted. When walking at a speed of 20 cm/s, the measured CoT for LEO was 108. When flying at 1 m/s, the CoT was 48, and it decreased to 15.5 at the flight speed of 3 m/s. The robots used for comparison have CoT values that are lower than LEO's, but they are lacking LEO's multi-modal capabilities. The data used for the plot is summarized in Table 7.2 and in [68].

Component	Reference	Specifications
Leg motors (swing & extension)	MKS HBL599	73.1 g, 4.1 Nm, 2.1 RPS (0.08 s/60°)
Leg motors (ad/abduction)	Hitec HSB-9381TH	78 g, 3.3 Nm, 1.2 RPS (0.14 s/60°)
Propeller motors	Garila X2508	2200kv
Propellers	HQ prop 6x4.5x3	
Propeller motor controller	Holybro Tekko32	35A 4-in-1 3-6S ESC
Microcontroller	STM32F722RE	216 MHz ARM Cortex-M7, 256kB SRAM, 512KB Flash
IMU	ICM-20602	range: ±2000°/s, ±16 g, noise: $0.004°/s/\sqrt{\text{Hz}}$, 100 $\mu g/\sqrt{\text{Hz}}$
VIO camera	Intel Realsense T265	55 g, 163° FoV, on-board VIO processing
Embedded computer	NanoPC-T4	2x2GHz + 4x1.5GHz, 4GB RAM, PCIe SSD, USB 3.0, 802.11ac WiFi, 63 g
Battery for propellers	Tattu R-line 6S	1300 mAh (29 Wh), 155 g, 95C discharge
Battery for legs & electronics	HRB 2S	2700 mAh (20 Wh), 110 g, 10C discharge
Remote control receiver	Graupner GR-12L	2.4 GHz digital RC receiver

Table 7.1: LEO's components specifications.

Robots	Mass [kg]	Speed [m/s]	CoT	Flyer	Mass [g]	Speed [m/s]	CoT
1. Stanford Dogoo	4.8	0.9	3.20	Fruit Fly	0.002	2.06	5.88
2. Minitaur	5	1.5	2.30	Black-fly	0.0025	1.19	6.32
3. Salto-1P	0.1	3.6	6.60	Mosquito	0.0032	2.28	7.89
4. Jerboa	2.5	1.52	2.50	Honey Bee	0.1	8.33	6.27
5. MIT Cheetah 2	33	6.0	0.51	Hummingbird	3	13.61	1.68
6. MIT Cheetah 3	45	N/A	0.45	Budgerigar	35	9.72	1.21
7. StarIETH	23	0.7	2.57	Laughing Gull	310	8.61	0.60
8. ANYMAL	30	0.8	1.23	Pigeon	384	16.11	0.39
9. Cheetah Cub	1	1.4	9.80				
10. XRL	23	1.54	2.57				
11. DURUS	79.5	0.6	1.02				

Table 7.2: Mass, speed, and Cost of Transportation for different walking/jumping robots and flyers adapted from [68–70].

Walker/Runner	Mass [kg]	Speed [m/s]	CoT	Walker/Runner	Mass [kg]	Speed [m/s]	CoT
	0.018	0.15	39.28	Lizard	0.900	0.10	2.60
	0.018	0.10	30.64	Rabbit	2.540	0.58	1.99
Mouse Species	0.023	0.30	14.64	Dog	10.70	1.63	0.57
	0.024	0.16	20.47	Sheep	30	0.80	0.42
	0.027	0.19	17.86	Human	70	1.56	0.31
Lemming	0.061	0.43	17.12	Pony	270	0.10	2.62
Kangaroo Rat	0.100	0.58	3.85	Colt	477	0.98	0.29
Ground Squirrel	0.198	0.78	2.27	Cattle	656	0.98	0.34
Rat Species	0.397	0.58	3.59	Horse	707	1.39	0.22
	0.250	0.30	6.28				

Table 7.3: Cost of Transportation for different walkers/runners adapted from [68].

Potential applications

By leveraging LEO's hybrid locomotion capability, it is anticipated that LEO will enable a wide range of robotic missions that are hard to accomplish by the sole use of ground or aerial robots. Perhaps the most well-suited applications for LEO would be the ones that involve physical interactions with structures at a high altitude, which are usually dangerous for human workers and call for a substitution by robotic workers. In such applications, conventional biped robots have difficulties with reaching the site, and standard multi-rotor drones have an issue with stabilization in high disturbance environments. LEO uses the ground contact to its advantage and, compared to a standard multi-rotor, is more resistant to external disturbances such as wind (Video 3 in [62]), which would improve the safety of the robot operation in an outdoor environment where LEO can maintain contact with a rigid surface.

Specifically, multi-point inspection, repair, or replacement tasks at locations that are difficult to reach by humans could be one primary application domain for LEO. For example, high voltage line inspection is currently done by highly skilled professionals who not only inspect the lines from a distance but also walk on the lines to inspect and repair them. If the lines are in hard-to-reach places, helicopters bring workers close to the lines. Instead of sending humans, LEO could fly up to the high voltage lines and walk on them for close-by inspection and repair, which could potentially reduce fatalities and cost of the task. Another good application for LEO would be painting of tall bridges. Currently, workers walk on slippery and sometimes round trusses for painting, and their risk of slipping is high. We

believe LEO is a good substitution of human workers for this task because LEO can maintain balance on slippery terrain (Video 2 in [62]) and because it can choose to fly to a safe location in case it falls. Similarly, LEO can be envisioned for other tasks such as inspection of a building roof or pipes of an oil refinery.

On the other hand, the technology developed for LEO's transition between ground and aerial locomotion modes could foster the development of adaptive landing gear systems comprised of controlled leg joints for aerial robots and other types of flying vehicles [71, 72]. Such legged landing gear systems would be especially useful for maintaining body balance of aerial robots while landing on sloped or uneven terrains, thereby reducing the risk of failure under harsh landing conditions.

7.6 Material and Methods

In this section, the individual components of the control architecture of LEO from Fig. 7.9 are detailed. We emphasize the kinematics and dynamics models used for generating walking trajectories and discuss our control strategy for walking and flying as well as the transition between the two. An in-depth analysis of the control authority of LEO's actuation mechanism is also provided.

Inverted Pendulum model for the walking phase of LEO

A controller based on an Inverted Pendulum (IP) model is chosen for LEO's walking because it captures the robot's fundamental dynamics (see Fig. 7.3B). This model assumes the following: all the masses are concentrated at a single point mass inside the torso, the legs are massless, and the stance foot acts as a pivot with the ground. These are reasonable assumptions since LEO's legs are lightweight, with all the actuators and electronics located in the torso. The model is further simplified by considering the frontal and sagittal planes separately, hence the dynamics prescribing LEO's ground locomotion are represented in two 2D planes instead of one 3D space. In addition to a conventional IP model, we add propellers that are used to stabilize the dynamics. These thrust forces serve as the only control input in this model.

Walking trajectory generation with Linear Inverted Pendulum model

If the height of the CoM is kept constant by controlling the leg extension force, the dynamic model of IP becomes linear in a Cartesian coordinate parameterization. This model is called Linear Inverted Pendulum (LIP) [73] and was used for LEO's walking trajectory generation. Because we constrain the CoM to be at a constant height, this reference trajectory has no discontinuity in the vertical velocity from

the impulsive ground reaction force due to impacts from leg exchange, which also leads to no discontinuities in the horizontal velocity. This model and the reference trajectory simplify the controller design, as explained further in the below section "Nonlinear tracking controller for walking using propellers."

Our approach of designing the walking trajectory closely follows [73], which exploits the linearity of the model to find closed-form solutions for foot placements and CoM trajectories. While simple in formulation, versatile walking patterns are possible with this method by modifying walking characteristics through different choices of a step period, a step length, or a desired CoM height. Because of the simplicity of this approach, the trajectories can be easily computed online.

We modify the LIP model with the addition of a constant uniform thrust from all four propellers to reduce the weight on the legs. By setting the torso attitude level ($\beta^i = \alpha^i$), with $i \in \{(\mathrm{s}), (\mathrm{f})\}$ corresponding to the sagittal and frontal planes, respectively, we find that the constant propeller thrust f_z enters the dynamics as a reduced gravitational acceleration:

$$\ddot{x} = \frac{g - f_z/m}{z} x. \tag{7.1}$$

Here, the x and z coordinates are the forward and vertical positions of the CoM with respect to the stance foot location, and g is the gravitational acceleration. This planar model is also valid in the frontal plane with y instead of x. The resulting trajectory looks natural, with the CoM swinging side-to-side as the robot makes steps. These trajectories require only the chosen constant thrust to be followed, minimizing the control action needed. The swing leg trajectory is parameterized such that it lifts and sets the foot down vertically and moves in between stance locations in a straight line.

Leg joint trajectory generation

Once the feet and CoM trajectories are designed, the relative foot trajectories are transformed into the associated joint angle trajectories to be tracked by a set of servo motors. This requires solving the inverse kinematics problem of the leg chain. Within the workspace of the legs, there is a unique feasible joint configuration for any foot position which makes the inverse kinematics a well-posed problem.

We solve the inverse kinematics by first finding θ_3 (Fig. 7.3) in the frontal plane

analytically and then numerically solving for θ_1 and θ_2 by finding the solution of:

$$\begin{bmatrix} \mathbf{p}^d_{\text{foot}} - \mathbf{p}_{\text{foot}}(\theta_1, \theta_2, \varphi_1, \varphi_2) \\ \mathbf{p}_{\text{loop}_1}(\theta_1, \varphi_1) - \mathbf{p}_{\text{loop}_2}(\theta_2, \varphi_2) \end{bmatrix} = \mathbf{0},$$

where $\mathbf{p}^d_{\text{foot}}$ is the desired foot position in the plane of the parallel leg mechanism. We over-parameterize the kinematics problem with the additional angles φ_1 and φ_2, and we add a constraint that the point $\mathbf{p}_{\text{loop}}$ is the same from both sides of the kinematic chain, i.e., $\mathbf{p}_{\text{loop}_1} = \mathbf{p}_{\text{loop}_2}$. This results in much simpler expressions compared to an analytic solution of $\mathbf{p}_{\text{loop}}$ based on θ_1 and θ_2 only, which speeds up the computation. Note that the second kinematic loop is a parallelogram, which makes the computation of $\mathbf{p}_{\text{foot}}$ as a function of $\mathbf{p}_{\text{loop}}$ and the angles θ_1, θ_2, φ_1, and φ_2 straightforward. Assuming feasibility, the angles are found using the Levenberg-Marquardt (LM) algorithm. We use the LM implementation in the nonlinear optimization module of Eigen library [74], which runs fast enough to be used online. Even though the LM is a local method, it converges to the correct solution using a neutral leg configuration as an initial guess.

Modeling of Inverted Pendulum for propeller control during stance phase

A diagram of the considered model is shown in Fig. 7.3B. Conceptually, as previously mentioned, this model can be seen as a projection of a full 3D model onto the frontal or sagittal plane. In this model, LEO's configuration is parameterized by the link length ℓ and the angle α between the virtual leg (the line segment connecting the CoM to the stance foot) and the inertial vertical axis on each plane. Thus, we define the vector of generalized coordinates as $\mathbf{q}^i \coloneqq [\ell^i, \alpha^i]^\top$ (i.e., $\mathbf{q}^i = \mathbf{q}^{(\text{s})}$ or $\mathbf{q}^i = \mathbf{q}^{(\text{f})}$). Using the Lagrangian method, the equations of motion for this model can be put in the following Euler-Lagrange form:

$$\mathbf{M}^i(\mathbf{q}^i)\ddot{\mathbf{q}}^i + \mathbf{C}^i(\mathbf{q}^i, \dot{\mathbf{q}}^i)\dot{\mathbf{q}}^i + \mathbf{G}^i(\mathbf{q}^i) = \begin{bmatrix} f^i_\ell \\ \tau^i_\alpha \end{bmatrix}, \tag{7.2}$$

where

$$\mathbf{M}^i = \begin{bmatrix} m & 0 \\ 0 & m\left(\ell^i\right)^2 \end{bmatrix}, \ \mathbf{C}^i = \begin{bmatrix} 0 & -m\ell^i\dot{\alpha}^i \\ m\ell^i\dot{\alpha}^i & m\ell^i\dot{\ell}^i \end{bmatrix}, \ \mathbf{G}^i = \begin{bmatrix} mg\cos\left(\alpha^i\right) \\ -mg\ell^i\sin\left(\alpha^i\right) \end{bmatrix},$$

and f^i_ℓ is the leg extension force along the virtual leg and τ^i_α is the moment generated by the propeller thrusts f^i_+ and f^i_- about the pivot point. Note that β^i does not contribute to the model, although in reality, this angle is controlled by the leg servo motors, which have a limited bandwidth. Therefore, we do not assume control of β^i for stabilization.

Nonlinear tracking controller for walking using propellers

We present a nonlinear integral controller for walking using propellers, which guarantees exponential convergence of the angle α^i to a desired trajectory α^i_d obtained from Eq. 7.1. We use the IP model presented in the previous section restricted to one coordinate (α^i) because we assume that the servo motors embedded in the legs are tracking the leg angle (β^i) and length (ℓ^i). Unlike in the ideal case of the planned LIP trajectory, impacts can occur in our IP model when the foot exchange did not happen at the planned time because of tracking errors or uneven terrains. However, these impacts can be regarded as a disturbance in the controller's nonlinear stability analysis. LEO's nonlinear tracking controller with a sufficiently high gain achieves exponential convergence of tracking errors to the bounded balls around the impactless desired trajectory before the next leg exchanges. Note that since this controller stabilizes the planar model, two instantiations are run to control the sagittal and frontal plane individually.

Several factors can potentially yield modeling errors such as the CoM offset, inaccurate thruster models, structural deformation, or gearbox backlash. We noticed that the largest error arises from the CoM offset, which we model as a bounded external torque τ_{ext}. To compensate for this torque, we implement a nonlinear integral tracking controller with feedforward cancellation [75, 76] for each angle $\alpha^i = \alpha^{(\text{f})}$ or $\alpha^i = \alpha^{(\text{s})}$ as:

$$\tau^i_\alpha = m\left(\ell^i\right)^2 \ddot{\alpha}^i_r + 2m\ell^i\dot{\ell}^i\dot{\alpha}^i - mg\ell^i\sin(\alpha^i) - k^i(\dot{\alpha}^i - \dot{\alpha}^i_r) - k^i_I\int_{t_0}^{t}\left(\dot{\alpha}^i(\xi) - \dot{\alpha}^i_r(\xi)\right)\,d\xi, \tag{7.3}$$

where $\dot{\alpha}^i_r$ is the reference angular velocity defined as $\dot{\alpha}^i_r = \dot{\alpha}^i_d - k^i_\alpha\tilde{\alpha}^i$, with an error angle $\tilde{\alpha}^i = \alpha^i - \alpha^i_d$. The parameters k^i_α, k^i, and k^i_I are positive scalar gains, and t_0 and t are the initial and current time steps, respectively. The adaptive term (i.e., integral term in Eq. 7.3) ensures exponential convergence to the desired trajectory with the following error ball in the presence of an external disturbance term with the bounded time derivative $\dot{\tau}^i_{\text{ext}}(t)$:

$$\lim_{t\to\infty}\int_0^{\tilde{\alpha}^i}\|\delta\tilde{\alpha}^i\|_2 \;\le\; \eta(m, \ell^i(t), k^i_I, k^i, k^i_\alpha)\sup_t\|\dot{\tau}^i_{\text{ext}}(t)\|_2, \tag{7.4}$$

which indicates that the size of the error ball only depends on $\sup_t\|\dot{\tau}^i_{\text{ext}}\|_2$, not $\|\tau^i_{\text{ext}}\|_2$. Also note that this error bound is valid only in the regime between impacts since they are not explicitly modeled.

Proof 5 *The construction of the proof follows [77]. The closed-loop dynamics is given by*

$$m\ell^2\dot{\omega}_e + k\omega_e + k_I \int_{t_0}^{t} \omega_e \, dt' = \tau_{\text{ext}}, \tag{7.5}$$

where $\omega_e = \dot{\alpha} - \dot{\alpha}_r$ *is the composite error term that includes both the angular position and rate errors and* $k_I, k > 0$ *are positive constants. By introducing the term* $\dot{y} = k_I\omega_e$*, we can write (7.5) without external torque as a linear system with a state-dependent coefficient matrix* $\boldsymbol{A}(t)$

$$\begin{bmatrix} \dot{\omega}_e \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -km^{-1}\ell(t)^{-2} & -m^{-1}\ell(t)^{-2} \\ k_I & 0 \end{bmatrix} \begin{bmatrix} \omega_e \\ y \end{bmatrix} =: \mathbf{A}(t) \begin{bmatrix} \omega_e \\ y \end{bmatrix}, \tag{7.6}$$

where the leg extension $\ell(t)$ *is regarded as an external time-varying term for (7.5).*

We show that the system in (7.6) is contracting (i.e., exponentially converging to a single trajectory globally from any initial condition [78]) by constructing a positive definite matrix $\mathbf{P} = \begin{bmatrix} m & b \\ b & 1 \end{bmatrix}$*, with a parameter* $b \in \left(0, \sqrt{m}\right)$*. The symmetric matrix* $(\mathbf{PA})_{\text{sym}} = \frac{1}{2}\left((\mathbf{PA}) + (\mathbf{PA})^\top\right)$ *is given by:*

$$(\mathbf{PA})_{\text{sym}} = \begin{bmatrix} k_I b - k\ell^{-2} & \frac{1}{2}k_I - \frac{1}{2}kbm^{-1}\ell^{-2} - \frac{1}{2}\ell^{-2} \\ \frac{1}{2}k_I - \frac{1}{2}kbm^{-1}\ell^{-2} - \frac{1}{2}\ell^{-2} & -bm^{-1}\ell^{-2} \end{bmatrix}.$$

Note that $(\mathbf{PA})_{\text{sym}}$ *is negative definite uniformly in time if the following conditions hold:*

$$b < kk_I^{-1}\ell^{-2} \quad \textit{and} \quad b \in (b_1, b_2), \tag{7.7}$$

with b_1 *and* b_2 *being the roots of* $\det\left((\mathbf{PA})_{\text{sym}}\right) = 0$*. Combining these conditions with the condition for positive definiteness of matrix* $\mathbf{P}$*,* b *needs to be chosen in the interval:*

$$b \in \left(max(0, b_1), min\left(b_2, \frac{k}{k_I\ell^2}, \sqrt{m}\right)\right).$$

We define the generalized virtual displacement vector as $\delta\mathbf{z} = [\delta\omega_e, \delta y]^\top$*, where* $\delta\omega_e$ *and* δy *are infinitesimal displacements at fixed time. We compute the rate of change as follows:*

$$\begin{aligned} \frac{d}{dt}\left(\delta\mathbf{z}^\top\mathbf{P}\delta\mathbf{z}\right) &= \delta\dot{\mathbf{z}}^\top\mathbf{P}\delta\mathbf{z} + \delta\mathbf{z}^\top\mathbf{P}\delta\dot{\mathbf{z}} \\ &= \delta\mathbf{z}^\top\left((\mathbf{PA}) + (\mathbf{PA})^\top\right)\delta\mathbf{z} \\ &\leq 2\lambda_{\max}\left((\mathbf{PA})_{\text{sym}}\right)\|\delta\mathbf{z}\|_2^2 \\ &\leq \frac{2\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}\left(\delta\mathbf{z}^\top\mathbf{P}\delta\mathbf{z}\right), \end{aligned} \tag{7.8}$$

where $\lambda_{\max}$ is the maximum eigenvalue of the argument matrix. Note that in (7.8), we used the eigenvalue bounds for quadratic forms property $\lambda_{\min}(\mathbf{P})\|\delta\mathbf{z}\|_2^2 \leq \delta\mathbf{z}^\top\mathbf{P}\delta\mathbf{z} \leq \lambda_{\max}(\mathbf{P})\|\delta\mathbf{z}\|_2^2$.

Considering the bounded external torque τ_{ext}, we construct the following system from (7.5):

$$\begin{bmatrix}\dot{\omega}_e \\ \dot{y}\end{bmatrix} = \begin{bmatrix}-km^{-1}\ell(t)^{-2} & -m^{-1}\ell(t)^{-2} \\ k_I & 0\end{bmatrix}\begin{bmatrix}\omega_e \\ y\end{bmatrix} - \boldsymbol{\tau} = \boldsymbol{A}(t)\begin{bmatrix}\omega_e \\ y\end{bmatrix} - \boldsymbol{\tau}, \tag{7.9}$$

where $\boldsymbol{\tau} = \begin{bmatrix}0 & \dot{\tau}_{\text{ext}}\end{bmatrix}^\top$. Using the virtual displacement $\delta\mathbf{z}$, we obtain:

$$\delta\dot{\mathbf{z}} = \boldsymbol{A}(t)\delta\mathbf{z} - \boldsymbol{\tau},$$

with its transpose $\delta\dot{\mathbf{z}}^\top = \delta\mathbf{z}^\top\boldsymbol{A}(t)^\top - \boldsymbol{\tau}^\top$. Computing the rate of change of $\delta\mathbf{z}^\top\mathbf{P}\delta\mathbf{z}$ for the case with external torque, we obtain:

$$\begin{aligned}\frac{d}{dt}\left(\delta\mathbf{z}^\top\mathbf{P}\delta\mathbf{z}\right) &= \left(\delta\mathbf{z}^\top\boldsymbol{A}(t)^\top - \boldsymbol{\tau}^\top\right)\mathbf{P}\delta\mathbf{z} + \delta\mathbf{z}^\top\mathbf{P}\left(\boldsymbol{A}(t)\delta\mathbf{z} - \boldsymbol{\tau}\right) \\ &= \delta\mathbf{z}^\top\left((\mathbf{PA}) + (\mathbf{PA})^\top\right)\delta\mathbf{z} - \boldsymbol{\tau}^\top\mathbf{P}\delta\mathbf{z} - \delta\mathbf{z}^\top\mathbf{P}\boldsymbol{\tau} \\ &\leq 2\lambda_{\max}\left((\mathbf{PA})_{\text{sym}}\right)\|\delta\mathbf{z}\|_2^2 - \boldsymbol{\tau}^\top\mathbf{P}\delta\mathbf{z} - \delta\mathbf{z}^\top\mathbf{P}\boldsymbol{\tau} \\ &\leq 2\lambda_{\max}\left((\mathbf{PA})_{\text{sym}}\right)\|\delta\mathbf{z}\|_2^2 - 2\delta\mathbf{z}^\top\begin{bmatrix}b \\ 1\end{bmatrix}\dot{\tau}_{\text{ext}} \\ &\leq 2\lambda_{\max}\left((\mathbf{PA})_{\text{sym}}\right)\|\delta\mathbf{z}\|_2^2 + 2\delta\mathbf{z}^\top\begin{bmatrix}b \\ 1\end{bmatrix}\|\dot{\tau}_{\text{ext}}\|_2,\end{aligned} \tag{7.10}$$

where the last term in (7.10) is derived by substituting $\mathbf{P} = \begin{bmatrix}m & b \\ b & 1\end{bmatrix}$. Next, by combining (7.8) and (7.10), we get:

$$\frac{d}{dt}\left(\delta\mathbf{z}^\top\mathbf{P}\delta\mathbf{z}\right) \leq \frac{2\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}\left(\delta\mathbf{z}^\top\mathbf{P}\delta\mathbf{z}\right) + 2\delta\mathbf{z}^\top\begin{bmatrix}b \\ 1\end{bmatrix}\|\dot{\tau}_{\text{ext}}\|_2. \tag{7.11}$$

Similar to the proof techniques in Contraction theory [78][79], we simplify (7.11) by finding a change of coordinates such that $\mathbf{P} = \boldsymbol{\Theta}^\top\boldsymbol{\Theta}$. Letting $\delta\boldsymbol{\xi} = \boldsymbol{\Theta}\delta\mathbf{z}$, and thus $\delta\mathbf{z} = \boldsymbol{\Theta}^{-1}\delta\boldsymbol{\xi}$, (7.11) becomes:

$$\frac{d}{dt}\left(\delta\boldsymbol{\xi}^\top\delta\boldsymbol{\xi}\right) \leq \frac{2\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}\delta\boldsymbol{\xi}^\top\delta\boldsymbol{\xi} + 2\delta\boldsymbol{\xi}^\top\boldsymbol{\Theta}^{-1}\begin{bmatrix}b \\ 1\end{bmatrix}\|\dot{\tau}_{\text{ext}}\|_2. \tag{7.12}$$

Further, we obtain:

$$\begin{aligned} 2\|\delta\boldsymbol{\xi}\|_2\frac{d}{dt}\|\delta\boldsymbol{\xi}\|_2 &\leq \frac{2\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}\|\delta\boldsymbol{\xi}\|_2^2 + 2\delta\boldsymbol{\xi}^\top\boldsymbol{\Theta}^{-1}\begin{bmatrix} b \\ 1 \end{bmatrix}\|\dot{\tau}_{\text{ext}}\|_2 \\ &\leq \frac{2\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}\|\delta\boldsymbol{\xi}\|_2^2 + \|2\delta\boldsymbol{\xi}^\top\boldsymbol{\Theta}^{-1}\begin{bmatrix} b \\ 1 \end{bmatrix}\|\dot{\tau}_{\text{ext}}\|_2\|_2 \\ &\leq \frac{2\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}\|\delta\boldsymbol{\xi}\|_2^2 + 2\sqrt{b^2+1}\|\delta\boldsymbol{\xi}^\top\|_2\|\boldsymbol{\Theta}^{-1}\|_2\|\dot{\tau}_{\text{ext}}\|_2. \end{aligned} \tag{7.13}$$

As $\|\delta\boldsymbol{\xi}\|_2$ is positive, we further simplify (7.13) to obtain:

$$\frac{d}{dt}\|\delta\boldsymbol{\xi}\|_2 \leq \frac{\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}\|\delta\boldsymbol{\xi}\|_2 + \sqrt{b^2+1}\|\boldsymbol{\Theta}^{-1}\|_2\|\dot{\tau}_{\text{ext}}\|_2. \tag{7.14}$$

Recall that the matrix norm induced by a vector norm for a matrix $\mathbf{A}$ is $\|\mathbf{A}\|_2 = \sup_{x\neq 0}\frac{\|\mathbf{Ax}\|_2}{\|\mathbf{x}\|_2} = \lambda_{\max}(\mathbf{A})$. Applying this property to $\boldsymbol{\Theta}^{-1}$ and considering the Cholesky Decomposition theorem which states that for a positive definite matrix (real or complex), there exists one positive definite matrix $\mathbf{B}$ such that $\mathbf{A} = \mathbf{B}^*\mathbf{B}$, where $\mathbf{B}^*$ is the conjugate transpose, we obtain:

$$\frac{d}{dt}\|\delta\boldsymbol{\xi}\|_2 \leq \frac{\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}\|\delta\boldsymbol{\xi}\|_2 + \frac{1}{\sqrt{\lambda_{\min}(\mathbf{P})}}\sqrt{b^2+1}\|\dot{\tau}_{\text{ext}}\|_2. \tag{7.15}$$

We further apply the path integral $\int_{\dot{\alpha}_r}^{\dot{\alpha}}\|\delta\boldsymbol{\xi}\|_2 = R(t)$ to (7.15) and obtain:

$$\dot{R}(t) \leq \frac{\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}R(t) + \frac{1}{\sqrt{\lambda_{\min}(\mathbf{P})}}\sqrt{b^2+1}\|\dot{\tau}_{\text{ext}}\|_2. \tag{7.16}$$

Using the Comparison Lemma [80], we get:

$$R(t) \leq e^{\frac{\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}}R(0) + \frac{1}{\sqrt{\lambda_{\min}(\mathbf{P})}}\sqrt{b^2+1}\int_0^t e^{\frac{\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}(t-\tau)}\|\dot{\tau}_{\text{ext}}\|_2 d\tau. \tag{7.17}$$

Recall the following:

$$\begin{aligned} R(0) &= \int_{\dot{\alpha}_r}^{\dot{\alpha}}\|\delta\boldsymbol{\xi}(0)\|_2 = \int_{\dot{\alpha}_r}^{\dot{\alpha}}\|\boldsymbol{\Theta}\delta\mathbf{z}(0)\|_2 = \sqrt{\lambda_{\max}(\mathbf{P})}\int_{\dot{\alpha}_r}^{\dot{\alpha}}\|\delta\mathbf{z}(0)\|_2. \\ R(t) &= \int_{\dot{\alpha}_r}^{\dot{\alpha}}\|\delta\boldsymbol{\xi}(t)\|_2 = \int_{\dot{\alpha}_r}^{\dot{\alpha}}\|\boldsymbol{\Theta}\delta\mathbf{z}(t)\|_2 \geq \sqrt{\lambda_{\min}(\mathbf{P})}\int_{\dot{\alpha}_r}^{\dot{\alpha}}\|\delta\mathbf{z}(t)\|_2. \end{aligned} \tag{7.18}$$

Using the properties in (7.18), we obtain:

$$\begin{aligned} \int_{\dot{\alpha}_r}^{\dot{\alpha}}\|\delta\mathbf{z}(t)\|_2 \leq{}& \frac{\sqrt{\lambda_{\max}(\mathbf{P})}}{\sqrt{\lambda_{\min}(\mathbf{P})}}e^{\frac{\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}t}\int_{\dot{\alpha}_r}^{\dot{\alpha}}\|\delta\mathbf{z}(0)\|_2 \\ &+ \frac{1}{\lambda_{\min}(\mathbf{P})}\sqrt{b^2+1}\int_0^t e^{\frac{\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}(t-\tau)}\|\dot{\tau}_{\text{ext}}\|_2 d\tau. \end{aligned} \tag{7.19}$$

Recall the definition of condition number

$$\kappa(\mathbf{A}) = \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})},$$

where $\sigma_{\max}(\mathbf{A})$ *and* $\sigma_{\min}(\mathbf{A})$ *are the maximal and minimal singular values of* $\mathbf{A}$ *respectively. Using this definition, (7.19) becomes:*

$$\begin{aligned}\int_{\dot{\alpha}_r}^{\dot{\alpha}} \|\delta\mathbf{z}\|_2 \leq \sqrt{\kappa(\mathbf{P})} e^{\frac{\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}t} \int_{\dot{\alpha}_r}^{\dot{\alpha}} \|\delta\mathbf{z}(0)\|_2 \\ - \frac{\sqrt{b^2+1}}{\lambda_{\max}((\mathbf{PA})_{\text{sym}})}\kappa(\mathbf{P})\left(1 - e^{\frac{\lambda_{\max}((\mathbf{PA})_{\text{sym}})}{\lambda_{\max}(\mathbf{P})}t}\right) \sup_t \|\dot{\tau}_{\text{ext}}\|_2.\end{aligned} \tag{7.20}$$

In limit as $t \to \infty$*:*

$$\lim_{t\to\infty} \int_{\dot{\alpha}_r}^{\dot{\alpha}} \|\delta\mathbf{z}\|_2 \leq \sqrt{b^2+1} \frac{\kappa(\mathbf{P})}{-\lambda_{\max}((\mathbf{PA})_{\text{sym}})} \sup_t \|\dot{\tau}_{\text{ext}}\|_2. \tag{7.21}$$

We further use the fact that $\|\delta\omega_e\|_2 \leq \|\delta\mathbf{z}\|_2$ *to obtain the final inequality:*

$$\lim_{t\to\infty} \int_{\dot{\alpha}_r}^{\dot{\alpha}} \|\delta\omega_e\|_2 \leq \sqrt{b^2+1} \frac{\kappa(\mathbf{P})}{-\lambda_{\max}((\mathbf{PA})_{\text{sym}})} \sup_t \|\dot{\tau}_{\text{ext}}\|_2. \tag{7.22}$$

Note that this is the worst-case bound for the error in the angular rate. By hierarchical combination of the dynamics of ω_e *and* $\tilde{\alpha} = \alpha - \alpha_r$*, we can also prove the convergence of* $\tilde{\alpha}$*, which is the angle error of the robot.*

Heading controller for walking

LEO's heading is controlled using propellers. Since LEO's feet form point contacts with the ground, the robot is free to rotate in all three axes, including yaw, when only one foot is in ground contact. In this case, the propellers can be used to control the heading of the robot using a Proportional-Integral-Derivative (PID) controller. The controller outputs a yaw moment that is mixed with the desired pitch and roll moments from the walking tracking controllers.

Optimized control allocation for walking

Once the desired control moments are computed, they are mixed to get propeller motor signals. Let $\tau_d = [\tau_\alpha^{(\text{f})}, \tau_\alpha^{(\text{s})}, \tau_{\text{yaw}}]^\top$, where the first two elements of the torque vector are computed from Eq. 7.3 and the last element from the previously mentioned heading controller, and let $\mathbf{f}_i$ be the thrust force from the i-th propeller and denote its magnitude as f_i (i.e., $f_i = \|\mathbf{f}_i\|_2$). The control effectiveness matrix $\mathbf{B}^{\text{walk}} \in \mathbb{R}^{3\times4}$ maps individual propeller thrusts to the moments about the stance

foot. To compute the propeller thrusts that yield the desired control moments, we solve the following optimization problem for $\mathbf{u} = [f_1, \ldots, f_4]^\top$:

$$\begin{aligned} \arg\min_{\mathbf{u}} \quad & \|\mathbf{u}\|_1 \\ \text{subject to} \quad & \mathbf{B}^{\text{walk}}\mathbf{u} = \boldsymbol{\tau}_d \\ & 0 \leq f_{\min} \leq f_i, \;\; i = 1, \ldots, 4 \\ & f_z^d / \cos(\delta) \leq \|\mathbf{u}\|_1. \end{aligned} \tag{7.23}$$

Each propeller thrust f_i is constrained to be greater than the idle thrust $f_{\min}$, which is based on the minimum rotation speed at which the sensorless BLDC controllers can run the motors. The last inequality enforces a minimum desired collective upward thrust f_z^d, which is used to reduce weight on the legs. Note that Eq. 7.23 always has a solution since the rows of $\mathbf{B}^{\text{walk}}$ are linearly independent and there exists a $\mathbf{u}^+$ strictly positive element-wise such that $\mathbf{B}^{\text{walk}}\mathbf{u}^+ = \mathbf{0}$, which is given by the geometry of our tilted propeller axes. The optimization problem is solved using a closed-form solution given in the Supplementary Text. In case the allocation results in thrust forces f_i that exceed the maximum thrust, they are truncated. However, such a situation was not observed in our experiments. This allocation scheme does not consider friction or normal force constraints. However, our experiments and analysis show that these constraints are usually met in practice.

Flight controller

In a flight mode, LEO uses a position and attitude controller similar to [31]. Specifically, the desired angular rate is computed from the attitude error between the current and desired body orientation. Then, the desired control moments are computed in roll, pitch, and yaw directions from the angular rate error. Finally, the control moments and the desired thrust are mixed to compute propeller motor signals, where the desired thrust is computed from position and velocity errors. Note that in this case the propeller moments are computed with respect to the CoM, not about the stance foot as it no longer serves as a pivot point.

Transition between walking and flying

As explained earlier, the selection of the active controller (flying or walking) depends on whether the foot contact sensors detect the ground or not (see Fig. 7.9). Take-off and landing are achieved by generating suitable walking and flying trajectories and monitoring the foot contact state to switch between them. During take-off, the collective propeller thrust f_z^d from Eq. 7.23 is increased while LEO is walking until

the ground contact is lost. Then, a flight trajectory with an initial upward velocity is initiated, which allows LEO to quickly lift away from the ground. LEO executes its desired flight trajectory thereafter.

For landing, LEO enters a constant velocity descent until one of the foot contact sensors detects the ground. During this descent, the legs are in a configuration that allows the robot to switch to walking immediately. Once the ground contact is detected, the flying trajectory is aborted, and the flight attitude controller is given a zero thrust and a level attitude setpoint. Further, the walking gait with the same speed as the forward velocity of the descent is initiated. This allows for a seamless transition from flying to walking.

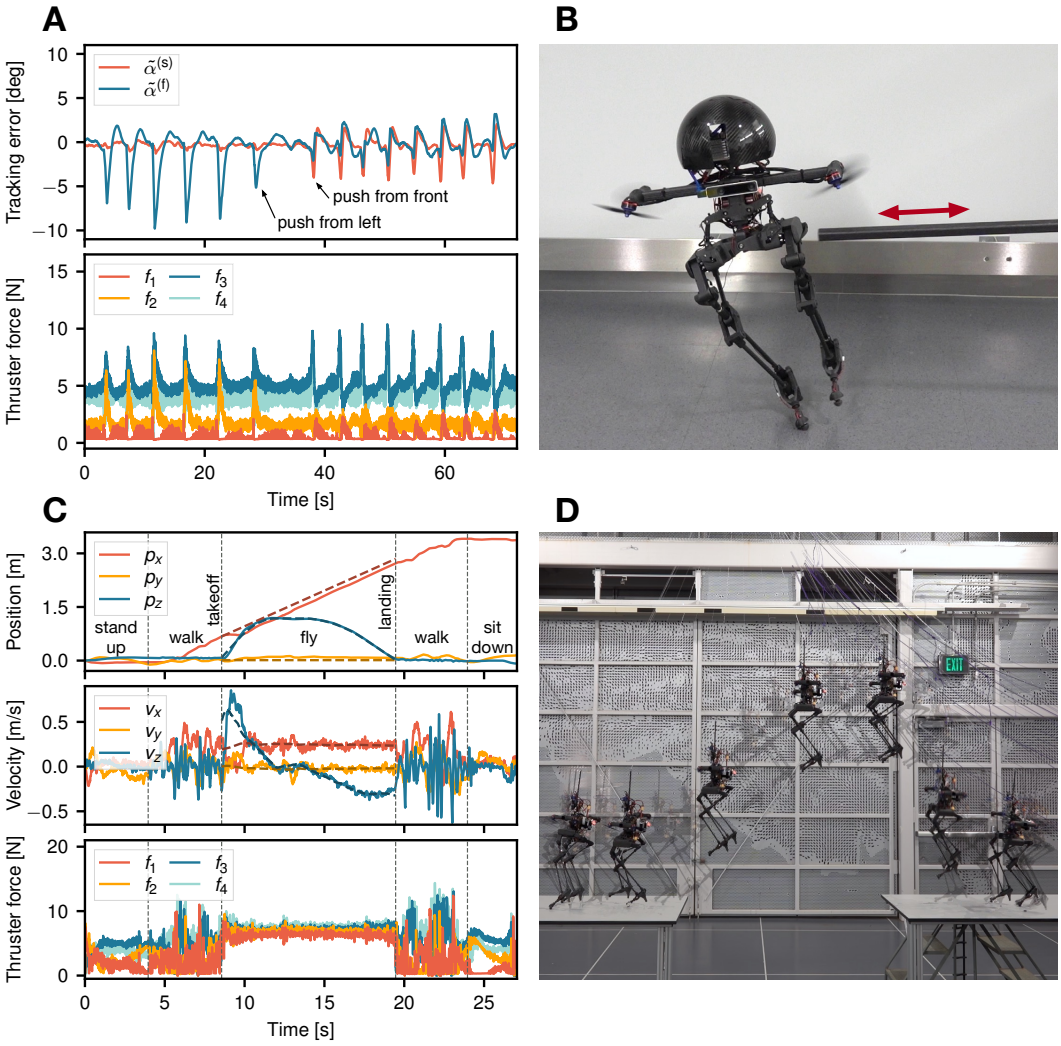


Figure 7.6: Disturbance rejection and synchronized walking and flying experiments for LEO. (A) Perturbation rejection experiment. The top plot shows the tracking error in both the sagittal and frontal planes during a series of 14 applied perturbations. The bottom plot presents the commanded thrust force of each of the four propellers. (B) Picture of LEO being pushed externally with a stick. (C) Trajectory plots of LEO performing a synchronized walking and flying maneuver to traverse two tables. The first two plots show the robot's actual position and velocity, together with the desired position and velocity (dashed line) for the flight phase. The third plot shows the commanded propeller thrust. (D) Overlaid snapshots of LEO performing the synchronized walking and flying experiment, tethered for safety.

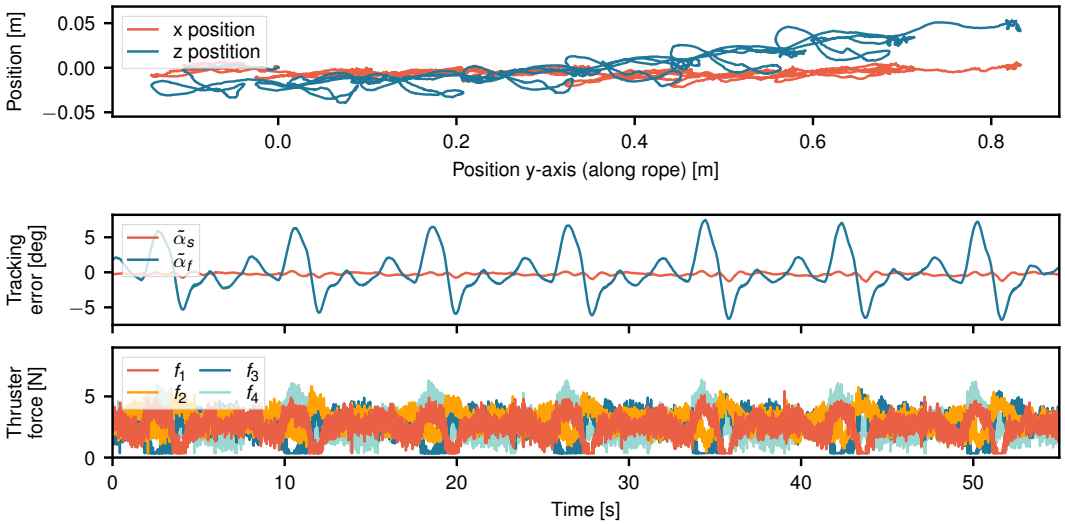


Figure 7.7: Experimental data for walking on a loosely-tensioned rope.

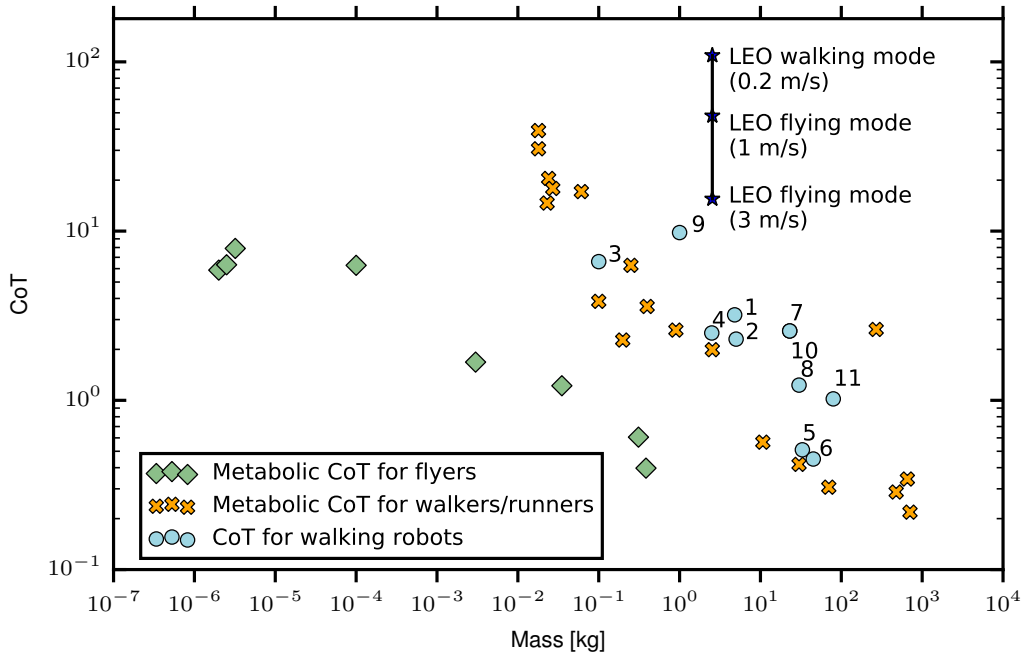


Figure 7.8: Mass vs. Cost of Transportation for different walkers, flyers, and robots.The data are provided in Table 7.2 and in [68]. The solid line between the data points of LEO represents a range of CoT achievable by combining the two locomotion modes of LEO.

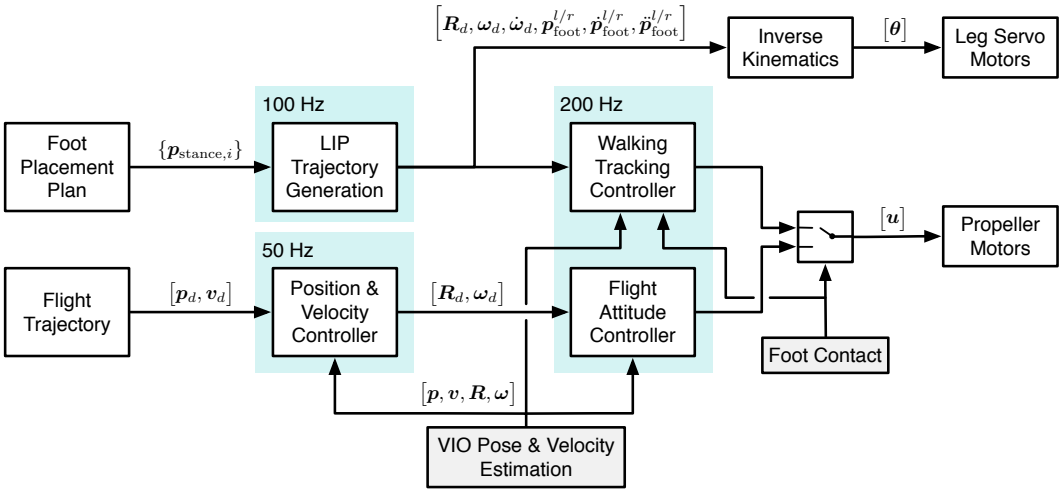


Figure 7.9: Control architecture of LEO. The upper signal chain controls the walking mode and the lower chain controls the flying mode. The propeller control is switched between these controllers based on the foot contact sensor measurements. $\boldsymbol{u} = [f_1, \ldots, f_4]^\top$ is the propeller thrust vector, $\boldsymbol{\theta}$ is the vector of servo joint angles for both legs, $\boldsymbol{p}_d, \boldsymbol{v}_d, \boldsymbol{R}_d$, and $\boldsymbol{\omega}_d$ are the desired position, velocity, attitude, and angular rate to be tracked. The vector $\boldsymbol{p}^{l/r}_{\text{foot}}$ contains the left and right foot positions. The list $\{\boldsymbol{p}_{\text{stance},i}\}$ contains a sequence of stance foot positions.

BIBLIOGRAPHY

[1] Marc H. Raibert. *Legged Robots that Balance*. MIT Press, 1986.

[2] Umar Asif and Javaid Iqbal. "On the Improvement of Multi-Legged Locomotion over Difficult Terrains using a Balance Stabilization Method". In: *International Journal of Advanced Robotic Systems* 9.1 (2012), p. 1.

[3] Jessy W. Grizzle, Christine Chevallereau, Ryan W. Sinnet, and Aaron D. Ames. "Models, Feedback Control, and Open Problems of 3D Bipedal Robotic Walking". In: *Automatica* 50.8 (2014), pp. 1955–1988.

[4] Dong Jin Hyun, Sangok Seok, Jongwoo Lee, and Sangbae Kim. "High Speed Trot-Running: Implementation of a Hierarchical Controller Using Proprioceptive Impedance Control on the MIT Cheetah". In: *The International Journal of Robotics Research* 33.11 (2014), pp. 1417–1445.

[5] Jong-Min Yang and Jong-Hwan Kim. "Sliding Mode Control for Trajectory Tracking of Nonholonomic Wheeled Mobile Robots". In: *IEEE Transactions on Robotics and Automation* 15.3 (1999), pp. 578–587.

[6] Fabien Tâche, Wolfgang Fischer, Gilles Caprari, Roland Siegwart, Roland Moser, and Francesco Mondada. "Magnebike: A Magnetic Wheeled Robot with High Mobility for Inspecting Complex-Shaped Structures". In: *Journal of Field Robotics* 26.5 (2009), pp. 453–476.

[7] Ronald Ping Man Chan, Karl A. Stol, and C. Roger Halkyard. "Review of Modelling and Control of Two-Wheeled Robots". In: *Annual Reviews in Control* 37.1 (2013), pp. 89–103.

[8] Shourov Bhattacharya and Sunil K. Agrawal. "Spherical Rolling Robot: A Design and Motion Planning Studies". In: *IEEE Transactions on Robotics and Automation* 16.6 (2000), pp. 835–839.

[9] Vrunda A. Joshi, Ravi N. Banavar, and Rohit Hippalgaonkar. "Design and Analysis of a Spherical Mobile Robot". In: *Mechanism and Machine Theory* 45.2 (2010), pp. 130–136.

[10] Kyunam Kim, Adrian K. Agogino, and Alice M. Agogino. "Rolling Locomotion of Cable-Driven Soft Spherical Tensegrity Robots". In: *Soft Robotics* 7.3 (2020), pp. 346–361.

[11] Alessandro Crespi, André Badertscher, André Guignard, and Auke Jan Ijspeert. "AmphiBot I: An Amphibious Snake-Like Robot". In: *Robotics and Autonomous Systems* 50.4 (2005), pp. 163–175.

[12] Aaron M. Hoover, Erik Steltz, and Ronald S. Fearing. "RoACH: An Autonomous 2.4g Crawling Hexapod Robot". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008, pp. 26–33.

[13] Robert F. Shepherd et al. “Multigait Soft Robot”. In: *Proceedings of the National Academy of Sciences of the United States of America* 108.51 (2011), pp. 20400–20403.

[14] Eric R. Westervelt, Jessy W Grizzle, Christine Chevallereau, Jun Ho Choi, and Benjamin Morris. *Feedback Control of Dynamic Bipedal Robot Locomotion*. CRC press, 2018.

[15] Christian Hubicki et al. “Walking and Running with Passive Compliance: Lessons from Engineering: A Live Demonstration of the ATRIAS Biped”. In: *IEEE Robotics & Automation Magazine* 25.3 (2018), pp. 23–39.

[16] Matthew Johnson et al. “Team IHMC’s Lessons Learned from the DARPA Robotics Challenge Trials”. In: *Journal of Field Robotics* 32.2 (2015), pp. 192–208.

[17] Siyuan Feng, Eric Whitman, X. Xinjilefu, and Christopher G. Atkeson. “Optimization-Based Full Body Control for the DARPA Robotics Challenge”. In: *Journal of Field Robotics* 32.2 (2015), pp. 293–312.

[18] Holly A. Yanco, Adam Norton, Willard Ober, David Shane, Anna Skinner, and Jack Vice. “Analysis of Human-Robot Interaction at the DARPA Robotics Challenge Trials”. In: *Journal of Field Robotics* 32.3 (2015), pp. 420–444.

[19] Jeongsoo Lim et al. “Robot System of DRC-HUBO+ and Control Strategy of Team KAIST in DARPA Robotics Challenge Finals”. In: *Journal of Field Robotics* 34.4 (2017), pp. 802–829.

[20] Rudolph Triebel et al. “SPENCER: A Socially Aware Service Robot for Passenger Guidance and Help in Busy Airports”. In: *Proceedings of the 10th Conference on Field and Service Robotics*. Vol. 113. 2015, pp. 607–622.

[21] Manuela Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. “CoBots: Robust Symbiotic Autonomous Mobile Service Robots”. In: *Proceedings of the 24th International Conference on Artificial Intelligence*. 2015, pp. 4423–4429.

[22] Joseph Geringer, Richard Watson, and Jason Cooper. *Robot Vacuum Cleaner*. 2010.

[23] John P. Grotzinger et al. “Mars Science Laboratory Mission and Science Investigation”. In: *Space Science Reviews* 170.1-4 (2012), pp. 5–56.

[24] Gabe Nelson, Aaron Saunders, and Robert Playter. “The PETMAN and Atlas Robots at Boston Dynamics”. In: *Humanoid Robotics: A Reference*. 2019, pp. 169–186.

[25] Jean-Christophe Zufferey, Adam Klaptocz, Antoine Beyeler, Jean-Daniel Nicoud, and Dario Floreano. “A 10-Gram Vision-Based Flying Robot”. In: *Advanced Robotics* 21.14 (2007), pp. 1671–1684.

[26] H. Jin Kim, Mingu Kim, Hyon Lim, Chulwoo Park, Seungho Yoon, Daewon Lee, Hyunjin Choi, Gyeongtaek Oh, Jongho Park, and Youdan Kim. "Fully Autonomous Vision-Based Net-Recovery Landing System for a Fixed-Wing UAV". In: *IEEE/ASME Transactions On Mechatronics* 18.4 (2013), pp. 1320–1333.

[27] Xichen Shi, Patrick Spieler, Ellande Tang, Elena-Sorina Lupu, Phillip Tokumaru, and Soon-Jo Chung. "Adaptive Nonlinear Control of Fixed-Wing VTOL with Airflow Vector Sensing". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2020, pp. 5321–5327.

[28] Bill Crowther, Alexander Lanzon, Martin Maya-Gonzalez, and David Langkamp. "Kinematic Analysis and Control Design for a Nonplanar Multirotor Vehicle". In: *Journal of Guidance, Control, and Dynamics* 34.4 (2011), pp. 1157–1171.

[29] Robert Mahony, Vijay Kumar, and Peter Corke. "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor". In: *IEEE Robotics and Automation magazine* 19.3 (2012), pp. 20–32.

[30] Dario Brescianini and Raffaello D'Andrea. "An Omni-Directional Multirotor Vehicle". In: *Mechatronics* 55 (2018), pp. 76–93.

[31] Xichen Shi, Kyunam Kim, Salar Rahili, and Soon-Jo Chung. "Nonlinear Control of Autonomous Flying Cars with Wings and Distributed Electric Propulsion". In: *2018 IEEE Conf. on Decision and Control (CDC)*. IEEE. 2018, pp. 5326–5333.

[32] Christophe De Wagter, Rick Ruijsink, Ewoud J.J. Smeur, Kevin G. van Hecke, Freek van Tienen, Erik van der Horst, and Bart DW Remes. "Design, Control, and Visual Navigation of the DelftaCopter VTOL Tail-Sitter UAV". In: *Journal of Field Robotics* 35.6 (2018), pp. 937–960.

[33] Hazim Shakhatreh et al. "Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges". In: *IEEE Access* 7 (2019), pp. 48572–48634.

[34] Guillermo Heredia, A.E. Jimenez-Cano, I. Sanchez, Domingo Llorente, V. Vega, J Braga, J.A. Acosta, and Aníbal Ollero. "Control of a Multirotor Outdoor Aerial Manipulator". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 3417–3422.

[35] F. Ruggiero, M. A. Trujillo, R. Cano, H. Ascorbe, A. Viguria, C. Perez, V. Lippiello, A. Ollero, and B. Siciliano. "A Multilayer Control for Multirotor UAVs Equipped with a Servo Robot Arm". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2015, pp. 4014–4020.

[36] Suseong Kim, Hoseong Seo, and H. Jin Kim. "Operating an Unknown Drawer Using an Aerial Manipulator". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2015, pp. 5503–5508.

[37] Hai-Nguyen Nguyen, Sangyul Park, Junyoung Park, and Dongjun Lee. “A Novel Robotic Platform for Aerial Manipulation Using Quadrotors as Rotating Thrust Generators”. In: *IEEE Transactions on Robotics* 34.2 (2018), pp. 353–369.

[38] Gregory Dudek et al. “Aqua: An Amphibious Autonomous Robot”. In: *Computer* 40.1 (2007), pp. 46–53.

[39] Auke Jan Ijspeert, Alessandro Crespi, Dimitri Ryczko, and Jean-Marie Cabelguen. “From Swimming to Walking with a Salamander Robot Driven by a Spinal Cord Model”. In: *Science* 315.5817 (2007), pp. 1416–1420.

[40] Richard J. Bachmann, Ravi Vaidyanathan, and Roger D. Quinn. “Drive Train Design Enabling Locomotion Transition of a Small Hybrid Air-Land Vehicle”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009, pp. 5647–5652.

[41] Ludovic Daler, Julien Lecoeur, Patrizia Bernadette Hählen, and Dario Floreano. “A Flying Robot with Adaptive Morphology for Multi-Modal Locomotion”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 1361–1366.

[42] Frank J. Boria, Richard J. Bachmann, Peter G. Ifju, Roger D. Quinn, Ravi Vaidyanathan, Chris Perry, and Jeffrey Wagener. “A Sensor Platform Capable of Aerial and Terrestrial Locomotion”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2005, pp. 3959–3964.

[43] Mirko Kovač, Jean-Christophe Zufferey, and Dario Floreano. “Towards a Self-Deploying and Gliding Robot”. In: *Flying Insects and Robots*. Springer, 2009, pp. 271–284.

[44] Ludovic Daler. “Adaptive Morphology for Multi-Modal Locomotion”. PhD thesis. Lausanne, Switzerland: École polytechnique fédérale de Lausanne, 2015.

[45] Alex Kossett and Nikolaos Papanikolopoulos. “A Robust Miniature Robot Design for Land/Air Hybrid Locomotion”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2011, pp. 4595–4600.

[46] Arash Kalantari and Matthew Spenko. “Design and Experimental Validation of HyTAQ, a Hybrid Terrestrial and Aerial Quadrotor”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2013, pp. 4445–4450.

[47] Christopher J. Pratt and Kam K. Leang. “Dynamic Underactuated Flying-Walking (DUCK) Robot”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2016, pp. 3267–3274.

[48] Scott Morton and Nikolaos Papanikolopoulos. “A Small Hybrid Ground-Air Vehicle Concept”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2017, pp. 5149–5154.

[49] He Wang, Jiadong Shi, Jianzhong Wang, Hongfeng Wang, Yiming Feng, and Yu You. “Design and Modeling of a Novel Transformable Land/Air Robot”. In: *International Journal of Aerospace Engineering* 2019 (2019), pp. 1–10.

[50] Morgan T. Pope, Christopher W. Kimes, Hao Jiang, Elliot W. Hawkes, Matt A. Estrada, Capella F. Kerst, William RT. Roderick, Amy K. Han, David L. Christensen, and Mark R. Cutkosky. “A Multimodal Robot for Perching and Climbing on Vertical Outdoor Surfaces”. In: *IEEE Transactions on Robotics* 33.1 (2016), pp. 38–48.

[51] Paul Beardsley, Roland Siegwart, M. Arigoni, Michael Bischoff, Silvan Fuhrer, David Krummenacher, Dario Mammolo, and Robert Simpson. “Vertigo: A Wall-Climbing Robot Including Ground-Wall Transition”. In: *Disney Research* 29 (2015).

[52] Sepehr Ghassemi and Dennis Hong. “Feasibility study of a novel robotic system BALLU: Buoyancy assisted lightweight legged unit”. In: *Proceedings of the 16th IEEE International Conference on Humanoid Robots*. 2016, pp. 144–144.

[53] Kevin Y. Ma, Pakpong Chirarattananon, Sawyer B. Fuller, and Robert J. Wood. “Controlled Flight of a Biologically Inspired, Insect-Scale Robot”. In: *Science* 340.6132 (2013), pp. 603–607.

[54] Aditya A. Paranjape, Soon-Jo Chung, and Joseph Kim. “Novel Dihedral-based Control of Flapping-Wing Aircraft with Application to Perching”. In: *IEEE Transactions on Robotics* 29.5 (2013), pp. 1071–1084.

[55] Alireza Ramezani, Soon-Jo Chung, and Seth Hutchinson. “A Biomimetic Robotic Platform to Study Flight Specializations of Bats”. In: *Science Robotics* 2.3 (2017), eaal2505.

[56] Anick Abourachid and Elizabeth Höfling. “The Legs: A Key to Bird Evolutionary Success”. In: *Journal of Ornithology* 153.1 (2012), pp. 193–198.

[57] William R.T. Roderick, Mark R. Cutkosky, and David Lentink. “Touchdown to Take-Off: At the Interface of Flight and Surface Locomotion”. In: *Interface Focus* 7.1 (2017), p. 20160094.

[58] Kathleen D. Earls. “Kinematics and Mechanics of Ground Take-Off in the Starling Sturnis vulgaris and the Quail Coturnix coturnix”. In: *Journal of Experimental Biology* 203.4 (2000), pp. 725–739.

[59] Evan Ackerman. *Delivery Drones Use Bird-Inspired Legs to Jump into the Air*. Ed. by IEEE Spectrum. [Online; accessed 06-November-2020]. Jan. 2019.

[60] K. Peterson, P. Birkmeyer, R. Dudley, and R.S. Fearing. "A Wing-Assisted Running Robot and Implications for Avian Flight Evolution". In: *Bioinspiration & Biomimetics* 6.4 (2011), p. 046008.

[61] Matthew A. Woodward and Metin Sitti. "Multimo-bat: A Biologically Inspired Integrated Jumping–Gliding Robot". In: *The International Journal of Robotics Research* 33.12 (2014), pp. 1511–1529.

[62] Kyunam Kim*, Patrick Spieler*, **Elena-Sorina Lupu**, Alireza Ramezani, and Soon-Jo Chung. "A Bipedal Walking Robot that can Fly, Slackline, and Skateboard". In: *Science Robotics* 6.59 (2021), eabf8136.

[63] Duncan W Haldane, Mark M Plecnik, Justin K Yim, and Ronald S Fearing. "Robotic Vertical Jumping Agility via Series-Elastic Power Modulation". In: *Science Robotics* 1.1 (2016).

[64] Duncan W Haldane, Justin K Yim, and Ronald S Fearing. "Repetitive Extreme-Acceleration (14-g) Spatial Jumping with Salto-1P". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3345–3351.

[65] Justin K Yim, Bajwa Roodra Pratap Singh, Eric K Wang, Roy Featherstone, and Ronald S Fearing. "Precision Robotic Leaping and Landing Using Stance-Phase Balance". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3422–3429.

[66] Azumi Maekawa, Ryuma Niiyama, and Shunji Yamanaka. "Pseudo-Locomotion Design with a Quadrotor-Assisted Biped Robot". In: *Proceedings of the IEEE International Conference on Robotics and Biomimetics*. 2018, pp. 2462–2466.

[67] Pravin Dangol, Alireza Ramezani, and Nader Jalili. "Performance Satisfaction in Harpy, a Thruster-Assisted Bipedal Robot". In: *arXiv preprint arXiv:2004.14337* (2020).

[68] A. Vance Tucker. "Energetic Cost of Locomotion in Animals". In: *Comp. Biochem. Physiol.* Vol. 34. 1970, pp. 841–846.

[69] Nathan Kau, Aaron Schultz, Natalie Ferrante, and Patrick Slade. *Stanford Doggo: An Open-Source, Quasi-Direct-Drive Quadruped*. 2019. arXiv: 1905.04254 [cs.RO].

[70] Jacob P. Reher, Ayonga Hereid, Shishir Kolathaya, Christian M. Hubicki, and Aaron D. Ames. "Algorithmic Foundations of Realizing Multi-Contact Locomotion on the Humanoid Robot DURUS". In: *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*. Cham: Springer International Publishing, 2020, pp. 400–415.

[71] J. Kiefer, M. Ward, and M. Costello. "Rotorcraft Hard Landing Mitigation Using Robotic Landing Gear". In: *Journal of Dynamic Systems, Measurement, and Control* 138.3 (2016).

[72] Yuri S. Sarkisov, Grigoriy A. Yashin, Evgeny V. Tsykunov, and Dzmitry Tsetserukou. “Dronegear: A Novel Robotic Landing Gear with Embedded Optical Torque Sensors for Safe Multicopter Landing on an Uneven Surface”. In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1912–1917.

[73] Shuuji Kajita, Hirohisa Hirukawa, Kensuke Harada, and Kazuhito Yokoi. *Introduction to Humanoid Robotics*. Springer, 2014.

[74] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[75] Hassan K. Khalil. *Nonlinear Systems*. 2nd. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[76] Jean-Jacques E. Slotine and Weiping Li. *Applied Nonlinear Control*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[77] Saptarshi Bandyopadhyay, Soon-Jo Chung, and Fred Y. Hadaegh. “Nonlinear Attitude Control of Spacecraft with a Large Captured Object”. In: *Journal of Guidance, Control, and Dynamics* 39.4 (2016), pp. 754–769.

[78] Winfried Lohmiller and Jean-Jacques E. Slotine. “On Contraction Analysis for Non-Linear Systems”. In: *Automatica* 34.6 (1998), pp. 683–696.

[79] Hiroyasu Tsukamoto, Soon-Jo Chung, and Jean-Jaques E. Slotine. “Contraction Theory for Nonlinear Stability Analysis and Learning-Based Control: A Tutorial Overview”. In: *Annual Reviews in Control* 52 (2021), pp. 135–169. ISSN: 1367-5788.

[80] Hassan K. Khalil. *Nonlinear Control*. Always Learning. Pearson, 2014.

Chapter 8

DESIGN, MOTION PLANNING, AND CONTROL OF A BIPED WITH DEFORMATION COMPENSATION

Acknowledgments: I would like to extend my gratitude to Patrick Spieler for his invaluable contributions to the design and implementation of the robot, to Leo Zhang for his assistance with software and hardware development, and to Jedi Alindogan and Mathieu Dekker for help with the hardware manufacturing and design.

Abstract

This paper presents the design, development, and control of an open-source bipedal robot intended as a platform for research. We introduce two distinct walking control strategies implemented on the same hardware platform. The first approach follows a model-based pipeline incorporating capture point for foot selection, trajectory optimization, and nonlinear tracking control to generate stable walking motions. The second approach employs reinforcement learning, specifically Proximal Policy Optimization (PPO), to learn control policies from the robot interacting with a simulator. For both approaches, we model the mechanical compliance (deformation) and evaluate the robot's performance under both control schemes, demonstrating successful locomotion tasks on hardware.

8.1 Introduction

Autonomous robots have the potential to be effectively used in a wide range of applications, such as agricultural applications [1, 2], wilderness and fire search and rescue missions [3–6], and planetary exploration [7]. However, successful execution of these applications requires that robots have advanced agility while traversing unstructured environments. Legged robots are better suited for such tasks compared to wheeled-only or tracked-only vehicles because they can traverse more diverse environments (i.e., complex terrains, stairs, ladders, etc).

Both academia and industry have been progressively redirecting their resources towards the research and development of legged robots, but mostly with a focus towards factory automation. This shift is evident in the successful demonstrations of robots such as Cassie [8–11], Digit [12], ANYmal [13], Spot, Atlas or Unitree H1 [14], with recent applications like inspection of offshore power plants [15], cave

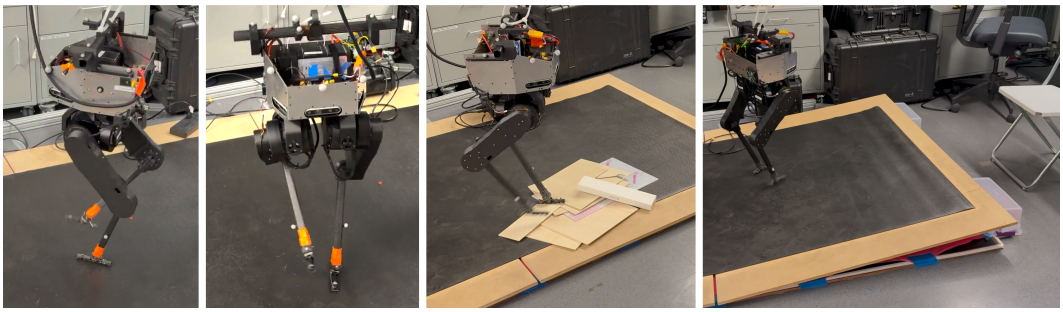


Figure 8.1: Showcase of different capabilities such as walking, traversing obstacles, and slopes. Video https://youtu.be/wBJqv6rsT_k?si=M7TZU-XICmYf211Z

mapping [16], or package delivery.

Despite recent improvements, most commercially available humanoid robots remain poorly suited for research and development. Many are closed-source, require multiple operators and a gantry to prevent falls, and must be returned to the manufacturer for repairs. When support is discontinued, researchers are left without spare parts or technical help. Lastly, the lack of open-source hardware and software further limits rapid iterations.

Contributions

To make rapid iteration and development of learning-based controllers more accessible, we introduce a cost-effective and agile bipedal robot (Fig. 8.1) designed to be manufactured only with 3D-printed materials and waterjet aluminum and carbon fiber plates. We propose two control strategies: (1) a model-based controller combined with a foot placement scheme based on reduced order models that compensates for the structural compliance of the 3D-printed design and (2) a reinforcement learning-based controller that learns locomotion policies in simulation. Both the robot's hardware design and the control algorithms will be released as open-source to support the research in the broader robotics community.

Notations

Unless otherwise noted, all vector norms are Euclidean. All matrices and vectors are written in bold. Joints abbreviations: HFE: Hip Flexion/Extension, HAA: Hip Abduction/Adduction, KFE: Knee Flexion/Extension

8.2 Related Work

Mid-size Bipeds and Humanoid Robots

A mid-size bipeds or humanoid robots, as defined in [17], refers to a robot that has the size of a child. Examples of such robots are the MIT Humanoid [18], HECTOR [19], Bruce [20], Adam [21], Leonardo [22], and the Berkeley Humanoid [17], as well as a slightly taller version, the Unitree G1. These mid-size bipeds have the advantage of providing a balance between mechanical simplicity and the ability to perform dynamic behaviors, such as walking, jumping, or recovering from perturbations, while being safer, easier to prototype, and more cost-effective than full-scale humanoids.

One of the most closely related efforts is the Berkeley Humanoid, which shares a similar vision of serving as a research platform for learning-based control [17]. However, unlike our approach, the Berkeley Humanoid is not open-source and incorporates many custom-machined parts, which increase both the complexity of fabrication and the overall cost.

Compliant Robot Control

To the best of our knowledge, all high performance humanoid and bipedal robots are constructed with rigid metal components. When elasticity is present, it is typically confined to specific areas that can be precisely modeled, such as the knee and heel springs in Cassie [23]. However, the control problem becomes significantly more complex when compliance arises in unanticipated areas, as is often the case when 3D printed materials are used.

To compensate for this intrinsic deformation, we incorporate a deformation compensation block in both the model-based and the learning-based controller. This block adjusts the placement of the joints accordingly, ensuring the robot is stable while walking.

8.3 Methods: Model-based Control and Planning

In this section, we introduce the methods for the model-based framework consisting of capture point for foot placement, nonlinear tracking control (Sec. 8.3), inverse dynamics, inverse kinematics (Sec. 8.3), and trajectory optimization for the feet (Sec. 8.3). The overall architecture is shown in Fig. 8.2.

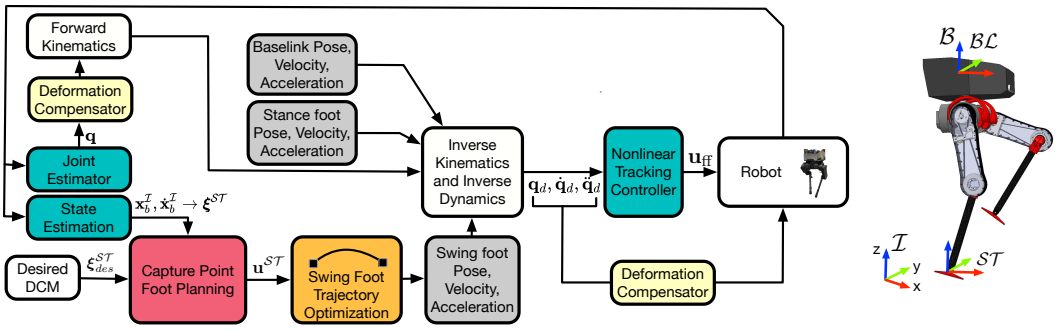


Figure 8.2: a) Control, estimation architecture, and motion planning. $\mathbf{q}$ is the actual joint angle, $\mathbf{x}_b^{\mathcal{I}}, \mathbf{x}_b^{\mathcal{I}}$ are the position and velocity of the body expressed in the inertial frame, $\mathbf{u}_{\mathcal{ST}}$ is the foot placement expressed in stance foot frame, $\mathbf{u}_{\text{ff}}$ is the feedforward torque applied to the motors, as computed by the nonlinear tracking controller, while $\boldsymbol{\xi}_{des}^{\mathcal{ST}}$ and $\boldsymbol{\xi}^{\mathcal{ST}}$ are the desired and the actual divergence component of motion. b) Frames of reference for the biped.

Modeling

Frames of Reference

We define the following frames of reference, as shown in Fig. 8.2. Let the inertial frame, $\mathcal{I}$, be a fixed reference frame and the body frame $\mathcal{B}$ a moving reference frame attached to the robot's main body. We assume the center of mass of the robot coincides with the origin of $\mathcal{B}$. We denote $\mathcal{BL}$ as the body level frame, which has the origin at the body frame, the vertical z-component aligned with the z-component of the inertial frame, and the $x-$ and $y-$ components rotated by the yaw orientation of the body. Let $\mathcal{ST}$ be the stance foot (foot in contact) frame, which has the origin at the stance foot location and is aligned with the $\mathcal{BL}$ frame. Note that this frame will change with every impact, depending on which foot is in contact with the ground.

Dynamics

The dynamics of the bipedal robot in contact with the ground (8.1) is modeled as a set of equations: one modeling the multi-body dynamics in the Lagrangian form, and another one modeling the contacts with the ground [24, 25]. A common assumption is that the foot-ground contact point, denoted as $\mathbf{x}_C$, does not change at impact (often called the no-slip condition).

$$\begin{aligned} \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) &= \mathbf{B}\mathbf{u} + \mathbf{J}_C^\top \boldsymbol{\lambda}, \\ \mathbf{x}_C &= \mathbf{J}_C\dot{\mathbf{q}} = \mathbf{0}, \\ \mathbf{x}_C &= \mathbf{J}_C\ddot{\mathbf{q}} + \dot{\mathbf{J}}_C\dot{\mathbf{q}} = \mathbf{0}, \end{aligned} \tag{8.1}$$

where, $\mathbf{q}$ is the combined set of coordinates $\left(\mathbf{x}_b, \boldsymbol{R}_b, \mathbf{q}_j\right)^\top \in \mathcal{Q} = \mathbb{R}^3 \times SO(3) \times \mathcal{Q}_j$, where $\mathbf{x}_b$ is the position of the floating base of the robot with respect to $\mathcal{I}$, $\boldsymbol{R}_b$ is the rotation matrix from $\mathcal{B}$ to $\mathcal{I}$, and $\mathbf{q}_j$ are the joint angles. $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{(n+6)\times(n+6)}$ is the inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{(n+6)\times(n+6)}$ contains the centripetal and Coriolis torques, $\mathbf{G} \in \mathbb{R}^{n+6}$ is the vector of gravitational torques, $\mathbf{B} \in \mathbb{R}^{(n+6)\times n}$ is the actuation matrix, and $\mathbf{u} \in \mathbb{R}^n$ is the control input. $\mathbf{J}_C(\mathbf{q}) = \begin{bmatrix} \frac{\partial \mathbf{x_C}}{\partial \mathbf{x}_b} & \frac{\partial \mathbf{x_C}}{\partial \mathbf{q}_j} \end{bmatrix} \in \mathbb{R}^{k\times(n+6)}$ is the Jacobian of k linearly independent constraints and $\boldsymbol{\lambda} \in \mathbb{R}^k$ is the constraint wrench vector (vector of forces and moments) [24, 25]. The value of k corresponds to the number of independent contact constraints enforced at the foot-ground or feet-ground contact points.

Foot Placement and Nonlinear Joint Tracking Control

Linear Inverted Pendulum Model

For foot placement, we model the robot using the Linear Inverted Pendulum (LIP) simplified model, which assumes a constant Center of Mass (CoM) height, massless legs, and all mass concentrated at the base. These assumptions align well with our robot's physical design. This simplified model is applied in both the sagittal (side view) and frontal planes.

Let $\mathbf{x}_c^{\mathcal{ST}} = [x_c, y_c]^\top$ be the planar CoM position relative to the stance foot frame. We then derive the following continuous time second-order linear dynamics

$$\ddot{\mathbf{x}}_c^{\mathcal{ST}}(t) = \omega^2 \mathbf{x}_c^{\mathcal{ST}}(t), \tag{8.2}$$

where ω is the natural frequency of the inverted pendulum, defined as $\sqrt{g/z_c}$, with z_c the constant height relative to the stance foot and g the gravitational acceleration. The foot stepping strategy is achieved by computing the divergence component of motion (DCM), which is a point on the ground where the robot should step to in order to bring itself to a complete stop [26]. This point can be calculated by imposing $\mathbf{x}_c^{\mathcal{ST}}(t) \to \mathbf{0}$ as $t \to \infty$ in the solution of the differential equation (8.2). Letting the DCM be $\boldsymbol{\xi}^{\mathcal{ST}} = [\xi_x, \xi_y]^\top$ with

$$\boldsymbol{\xi}^{\mathcal{ST}} = \mathbf{x}_c^{\mathcal{ST}} + \frac{1}{\omega}\dot{\mathbf{x}}_c^{\mathcal{ST}}, \tag{8.3}$$

then (8.2) becomes

$$\dot{\boldsymbol{\xi}}^{\mathcal{ST}}(t) = \omega \boldsymbol{\xi}^{\mathcal{ST}}(t). \tag{8.4}$$

To use (8.4) for planning footsteps, we need to discretize it with the discretization being the stepping time T_s, assumed constant, as follows

$$\boldsymbol{\xi}_{m+1}^{\mathcal{ST}} = e^{\omega T_s}\boldsymbol{\xi}_m^{\mathcal{ST}} - \mathbf{u}_m^{\mathcal{ST}}, \tag{8.5}$$

where m is an integer, counting foot steps, and $\mathbf{u}_m^{\mathcal{ST}}$ is the foot position in the stance foot frame. The position and velocity transitions between the steps (i.e., velocities shortly after and before contact) are assumed smooth. Designing a foot planning strategy for the system in (8.5) summarizes to modifying the swing foot location in the current footstep, such that the end-of-step DCM follows a desired DCM

$$\mathbf{u}_m^{\mathcal{ST}} = -\boldsymbol{\xi}_{\mathrm{d},m+1}^{\mathcal{ST}} + \boldsymbol{\xi}_m^{\mathcal{ST}} e^{\omega T_s}. \tag{8.6}$$

Interstep Foot Trajectories

To achieve continuous walking, a smooth trajectory for the swing foot should be generated that starts at its current position and ends at the desired DCM location such that $\boldsymbol{\xi}^{\mathcal{ST}} \rightarrow \boldsymbol{\xi}_{\mathrm{d}}^{\mathcal{ST}}$ in (8.5). For this, we solve the following quadratic program (8.7) in a model predictive control (MPC) fashion

$$\begin{aligned}
\min_{\mathbf{p},\mathbf{v},\mathbf{a}} \quad & \sum_{i}^{\{x,y,z\}} \left(\sum_{n=0}^{N} \|\mathbf{a}_n^i\|^2 + \|\mathbf{p}_N^i - \mathbf{p}^{i,\mathrm{d}}\|^2 + \|\mathbf{v}_N^i - \mathbf{v}^{i,\mathrm{d}}\|^2 \right) \\
\text{s.t.} \quad & \mathbf{p}_{n+1} = \mathbf{p}_n + \mathbf{v}_n \Delta \mathrm{t} \\
& \mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{a}_n \Delta \mathrm{t} \\
& \mathbf{p_0} = \mathbf{p}_0^{\mathrm{d}}; \mathbf{v_0} = \mathbf{v}_0^{\mathrm{d}} \\
& \mathbf{p}_n \in \mathcal{P}; \mathbf{v}_n \in \mathcal{V}; \mathbf{a}_n \in \mathcal{A} \\
& p^z_{n_k:n_{k+\xi}} = p_{\text{height}} \quad \text{(trapezoidal profile)},
\end{aligned} \tag{8.7}$$

where $\mathbf{p} = [p_x, p_y, p_z]$, $\mathbf{v} = [v_x, v_y, v_z]$, $\mathbf{a} = [a_x, a_y, a_z]$ are the Cartesian position, velocity, and acceleration of the swing foot in the $\mathcal{ST}$ frame, N is the number of discretization steps, Δt is the timestep, $\mathbf{p}_0^{\mathrm{d}}$ is the initial positions and $\mathbf{v_0} = \mathbf{v}_0^{\mathrm{d}}$ is the initial velocity of the swing foot, and $\mathbf{p}^{i,\mathrm{d}}$, with $i = \{x, y\}$ is the desired DCM location from (8.6). The desired z-component of $\mathbf{p}^{i,\mathrm{d}}$ is 0, signifying foot contact.

The sets $\mathcal{P}$, $\mathcal{V}$ and $\mathcal{A}$ are the safe sets for the position, velocity, and acceleration. $\mathcal{P}$ is chosen such that the feet do not cross and intersect, while $\mathcal{V}$ and $\mathcal{A}$ are chosen based on the desired velocity and acceleration limits. As the foot position is expressed in the stance foot frame ($\mathcal{ST}$), the safe set for the position $\mathcal{P}$ will change with each impact depending on which foot is touching the ground. The position and velocity are also following kinematic constraints (the 3$^{\text{rd}}$ and 4$^{\text{th}}$ constraints). in (8.7). The last constraint in (8.7) ensures the swing foot follows a trapezoidal profile (lift, keep constant, descent).

In case the optimization problem in (8.7) fails or becomes infeasible, we add several safety and robustness measures. Specifically, if the optimization problem in (8.7)

becomes infeasible, the robot's foot keeps its previous position computed before the problem became infeasible. The robot might still fall in that situation, but will not generate aggressive maneuvers. In addition, in case the current time step n exceeds the planned step duration N (i.e., $n > N$, meaning that impact did not happen in the prescribed time), the foot will hold its planar position, but keep lowering the vertical z-component with constant velocity until impact. In this way, we ensure robustness by accounting for steps with slightly different durations T_s.

Once (8.7) solves, these optimized position, velocity, and acceleration of the swing foot are used to compute the desired joint motion primitives $\mathbf{q}_d, \mathbf{q}_d, \mathbf{q}_d$ using inverse kinematics and inverse dynamics, as explained in the next section (Sec. 8.3).

Inverse Kinematics (IK) and Inverse Dynamics (ID)

Inverse Kinematics (IK) For computing the inverse kinematics, we use the iterative Levenberg-Marquardt (LM) damped least squares method with clamped error per solving step. Algorithm 6 operates concurrently on three bodies: the base link, the swing foot and the stance foot. In Line 2, we input the homogeneous transforms $\mathbf{T}_\mathrm{d} \in \mathrm{SE}(3)$, which represent the desired configuration (position and orientation) of these bodies expressed in the $\mathcal{ST}$ frame. The base link maintains a level orientation, positioned planarly above the stance foot at a constant height z_c. The desired position of the swing foot is computed from (8.7). In each iteration i, we compute the local Jacobian (Line 8) for each body and its transform $\mathbf{T}_b \in \mathrm{SE}(3)$ using forward kinematics (Line 9). Next, we then compute the error twist $\mathbf{e} \in \mathfrak{se}$ using the pseudo-inverse of the exponential map and clamp it using (8.8). By concatenating the individual Jacobians and error vectors of all bodies, we construct a large Jacobian matrix and a unified error vector (Line 12). These are utilized to solve for the evolution of the configuration $\delta\mathbf{q}_i$ via the damped least squares approach in Line 14.

This method is suitable when the robot follows smooth trajectories, and thus the computed $\mathbf{q}$ does not deviate substantially from its previous value. Therefore, for this numerical method to converge quickly, the initial guess in Line 3 is chosen to be the measured $\mathbf{q}$. For repeatability purposes, we choose to run Algorithm 6 with a fixed number of iterations IT_max. The clamping operator, defined as

$$\mathrm{clamp}(\mathbf{w}, D) = \begin{cases} \mathbf{w} & \text{if } \|\mathbf{w}\| \leq D \\ D\frac{\mathbf{w}}{\|\mathbf{w}\|} & \text{otherwise} \end{cases}, \tag{8.8}$$

with $D \in \mathbb{R}$ a positive threshold and $\mathbf{w} \in \mathbb{R}^3$ the input vector, reduces oscillations in

case the target positions are beyond reach and allows the use of a smaller damping constant λ, which in turn ensures faster convergence.

Next, we compute the desired joint velocities. Let the twist of the base link and the two feet expressed in the stance foot frame be $\boldsymbol{\nu}_b^k$, with $k = \{0, 1, 2\}$ and the corresponding Jacobians be $\mathbf{J}_b^k(\mathbf{q})$. We compute the desired joint velocities $\dot{\mathbf{q}}_d$ by solving the following damped least-squares

$$\min_{\dot{\mathbf{q}}_d} \|\mathbf{J}(\mathbf{q})\dot{\mathbf{q}}_d - \boldsymbol{\nu}\|^2 + \lambda_v \|\dot{\mathbf{q}}_d\|^2, \tag{8.9}$$

where $\mathbf{J}(\mathbf{q})$ and $\boldsymbol{\nu}$ are the concatenated Jacobians and twist vectors of the three bodies, similar to Line 12 in Algorithm 6 and λ_v is the damping coefficient. Note that to compute the Jacobian in (8.9), we use the measured joint angles $\mathbf{q}$, not the one computed in Algorithm 6.

Similarly, we compute the desired joint acceleration given the acceleration of the base link and the two feet expressed in the stance foot frame, $\mathbf{a}_b^k$, with $k = \{0, 1, 2\}$. For each body, the acceleration is

$$\mathbf{a}_b^k = \mathbf{J}_b^k \ddot{\mathbf{q}}_d + \dot{\mathbf{J}}_b^k \dot{\mathbf{q}}_d, \quad \forall k = \{0, 1, 2\}, \tag{8.10}$$

and as before, we compute the concatenated Jacobians $\mathbf{J}$, the concatenated Jacobian derivative $\mathbf{J}$ and the concatenated acceleration vector of the three bodies $\mathbf{a}$. Lastly, we solve for $\ddot{\mathbf{q}}_d$ in $\mathbf{a} = \mathbf{J}(\mathbf{q})\ddot{\mathbf{q}}_d + \dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}}$. Note that again we use the measured velocity of the joints $\dot{\mathbf{q}}$ from the encoders, and not the one computed from (8.9).

Inverse Dynamics (ID) We input the measured joint angles and velocities $\mathbf{q}$, $\dot{\mathbf{q}}$, the desired accelerations $\ddot{\mathbf{q}}_d$ from Sec. 8.3, the robot model, and output the feedforward torque $\mathbf{u}_{\text{ff}}$. We derive this feedforward torque for the case of one contact point (the stance foot). First, we compute the projection operator $\mathbf{P}$, which is introduced in (8.13), as well as the inertial matrix $\mathbf{M}$ using the composite rigid body algorithm (CRBA), the gravity term $\mathbf{G}$, and the Coriolis term $\mathbf{C}$. Using these terms and (8.14), we compute the feedforward torque.

Deformation Compensator Block

For the robot to walk accurately, it needs to precisely place its feet at the precise DCM location computed in (8.6). However, if there are slight deformations in the leg, the contact will happen at different locations than predicted. Therefore, we propose the following strategy to offset the joint angles based on the deformation

$$\mathbf{q}_j^{\text{corr.}} = \mathbf{q}_j + \bar{\mathbf{K}}_p \mathbf{f}_{db}(\bar{\boldsymbol{\tau}}_j), \quad \mathbf{q}_{d,j}^{\text{corr.}} = \mathbf{q}_{d,j} - \bar{\mathbf{K}}_p \mathbf{f}_{db}(\bar{\boldsymbol{\tau}}_j), \tag{8.11}$$

Algorithm 6 Inverse Kinematics

Input
  Desired $\mathbf{T}_d$ for the two feet and the baselink.
  Initial guess $\mathbf{q_0}$, empty Jacobian $\mathbf{J}$.
  Constants $\mathrm{IT}_{\max}$: number of iterations, λ: damping constant, $\mathrm{e}_{\mathrm{mag}}$: clamping error, Δt: integration step
Output Joint angles $\mathbf{q}_d$
for $i < \mathrm{IT}_{\max}$ **do**
  for $k < 3$ **do**
    Compute the Jacobian $\mathbf{J}_b^k$.
    Compute the transform $\mathbf{T}_b^k$.
    Compute error $\mathbf{e} = \log((\mathbf{T}_b^k(\mathbf{q}_i))^{-1}\mathbf{T}_d^k(\mathbf{q}_i))$.
    Clamp the translation component of $\mathbf{e}_k$ with $\mathrm{e}_{\mathrm{mag}}$.
    $\mathbf{J} = \mathrm{concat}(\mathbf{J}, \mathbf{J}_b^k)$, $\mathbf{e} = \mathrm{concat}(\mathbf{e}, \mathbf{e_k})$.
  end for
  $\delta\mathbf{q}_i = -\mathbf{J}^\top(\mathbf{J}\mathbf{J}^\top + \lambda\mathbf{I})\mathbf{e}$.
  $\mathbf{q}_{i+1} = \mathbf{q}_i + \delta\mathbf{q}_i\Delta t$.
end for

where $j = \{\mathrm{KFE}, \mathrm{HFE}, \mathrm{HAA}\}$ for both the right and left legs (see Sec. 8.1 for the joints abbreviations), $\mathbf{q}$ is the measured joint angle from the encoder, $\mathbf{q}_d$ denotes the desired joint angles computed through inverse kinematics, as in Sec. 8.3, $\bar{\boldsymbol{\tau}}$ is the torque from the motors to which we applied a low-pass filter, $\mathbf{f}_{db}$ is a deadband function necessary to protect for the noise at low torques, and $\bar{\mathbf{K}}_p$ is the corrective linear map. The deadband is necessary because lower torque levels indicate minimal load on the robot, resulting in negligible deformation, which should not be compensated for. We learn the $\bar{\mathbf{K}}_p$ map by comparing the actual deformation with the predicted one. The integration of this correction strategy within the robot's control architecture is illustrated in Fig. 8.2.

Nonlinear Joint Tracking Controller

Given the set of desired joint motion primitives $\mathbf{q}_d$, $\mathbf{q}_d$, and $\mathbf{q}_d$, we design a feedback controller $\mathbf{u}$ to track these desired motions. Designing $\mathbf{u}$ directly from (8.1) is not possible because the contact wrench is unknown. Therefore, we employ the method of projected inverse dynamics by projecting (8.1) into the null space of the constraints, and thus annihilating the unknown contact wrenches $\boldsymbol{\lambda}$ [27]. Let $\mathbf{P}$ be the orthogonal projection operator such that $\mathbf{P}\mathbf{J}_C^\top = \mathbf{0}$. Because the robot is walking dynamically, we apply the projection operator only in the case of one single contact (one foot on the ground). For this particular case, the projection operator eliminates

5 equations from the system in (8.1). The dynamics in (8.1) using the projection operator become

$$\mathbf{PM}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{PC}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{PG}(\mathbf{q}) = \mathbf{PBu} + \cancelto{\mathbf{0}}{\mathbf{PJ}_C^\top \boldsymbol{\lambda}} . \tag{8.12}$$

It is straightforward to compute the projection operator as

$$\mathbf{P} = \mathbf{I} - \mathbf{J}_C^+ \mathbf{J}_C, \tag{8.13}$$

where + is the Moore-Penrose right pseudoinverse of the contact Jacobian $\mathbf{J}_C$. Using this operator, we design the following nonlinear tracking controller

$$\mathbf{u} = \mathbf{u}_{\text{ff}} - \mathbf{K}_d(\dot{\mathbf{q}}_j - \dot{\mathbf{q}}_{d,j}) - \mathbf{K}_{\text{p}}(\mathbf{q}_j - \mathbf{q}_{d,j}), \tag{8.14}$$

with feedforward torque $\mathbf{u}_{\text{ff}} = [\mathbf{PB}]^+ \mathbf{P}\,(\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{G})$ where $\mathbf{K}_p$ and $\mathbf{K}_d$ are positive gain matrices and $\mathbf{q}_j$ are the joint angles, with $j = \{\text{YAW}, \text{HAA}, \text{HFE}, \text{KFE}\}$, $\dot{\mathbf{q}}_j$ are the joint angles derivative.

Walking Strategy

Next, we introduce the walking strategy (Algorithm 7). The objective of this controller is to compute the desired transforms, twists, and accelerations $\mathbf{T}_d, \mathbf{v}_d, \mathbf{a}_d$ of the base link and the two feet. These are then transformed into joint angles, velocities, and feedforward torques by the IK and ID algorithms, as shown in Algorithm 6 and Sec. 8.3.

First, we initialize the walking parameters in Lines 2-6: T_{s}, which is the step size, T_{buffer}, which is a time buffer used to avoid rapid contact switching at impact, $t_{\text{since last step}}$ as the time since the previous impact, and z_{c}, the desired height of the robot, selected as in Sec. 8.5. We initialize the robot in a starting-to-walk configuration (i.e., base link is level, stance foot is on the ground such that the robot's height is the desired walking height, and the swing foot raised).

The desired position and velocity from either a joystick or a desired trajectory is used to compute the desired DCM $\xi_d^{\mathcal{ST}}$. Next, the contact switches mounted onto the feet are read to detect impacts. The logic for foot switching is in Lines 12-18. At each timestamp, we compute the desired safe footstep, as shown in Lines 22-24 and solve the optimization problem in (8.7) to compute the swing foot trajectory. Using the swing foot states, the stance foot states and the baselink states, we solve the IK and the ID in Line 28.

Algorithm 7 Walking Controller

Input
  Step size T_s, time to ignore contact T_{buffer}.
  Desired height z_c.
  Set $t_{\text{since last step}} = T_s/2$.
  Starting walk configuration $\mathbf{T}_d$, $\mathbf{v}_d$, $\mathbf{a}_d$ for the baselink
  and the two feet.
Output
  Motor inputs: $\mathbf{q}_{j,\text{d}}$, $\mathbf{q}_{j,\text{d}}$ $\mathbf{u}_{\text{ff}}$.
while walking **do**
  Get desired DCM $\boldsymbol{\xi}_{\text{d}}^{\mathcal{ST}}$ from joystick/desired traj.
  Read contact states $c_{\text{swing-foot}}$ and $c_{\text{stance-foot}}$.
  if $t_{\text{since last step}} > T_{\text{buffer}}$ and $c_{\text{swing-foot}}$ is True **then**
    Swap swing foot $\longleftrightarrow$ stance foot.
    Set initial position of the swing foot $\mathbf{p}_0^d$ with the
    previous stance foot position.
    Set $\mathbf{p}_0^d$ and $\mathbf{v}_0^d$ in the optimization problem (8.7).
    Reset $t_{\text{since last step}} := 0$.
  end if
  Compute baselink position and velocity $\mathbf{x}_b^{\mathcal{ST}}$ and $\mathbf{v}_b^{\mathcal{ST}}$
  from Sec. 8.3.
  Compute DCM $\boldsymbol{\xi}^{\mathcal{ST}}$ using (8.3) and set $\boldsymbol{\xi}_z^{\mathcal{ST}} := 0$.
  Compute remaining time $t := T_{\text{step}} - t_{\text{since last step}}$.
  Compute desired footstep

$$\mathbf{u}^{\mathcal{ST}} := -\boldsymbol{\xi}_{\text{des}}^{\mathcal{ST}} + \boldsymbol{\xi}^{\mathcal{ST}} e^{\omega t}.$$

  Project $\mathbf{u}^{\mathcal{ST}}$ into a safe radius (Fig. 8.7).
  Set final $\mathbf{p}_N$ with $\mathbf{u}^{\mathcal{ST}}$ in optimization problem (8.7).
  Get optimal $\mathbf{p}^{\mathcal{ST}}$, $\mathbf{v}^{\mathcal{ST}}$, and $\mathbf{a}^{\mathcal{ST}}$ from solving (8.7).
  Set $\mathbf{T}_d$, $\mathbf{v}_d$, $\mathbf{a}_d$ for the baselink and the two feet.
  Solve IK (Algorithm 6) and ID (Sec. 8.3).
end while

State Estimation

To estimate the velocity of the body in the inertial frame $\mathbf{x}_b^{\mathcal{I}} := \mathbf{v}_b^{\mathcal{I}}$, the quaternion associated with the rotation matrix $\boldsymbol{R}_b$, $\boldsymbol{q}_b$, and the angular velocity of the body, $\boldsymbol{\omega}_b$, we use an error-state Kalman Filter [28] using the proprioceptive sensors such as the IMU and the joint encoders. We define the errors between the true state and the nominal state as

$$\mathbf{x}_{b,t}^{\mathcal{I}} = \mathbf{x}_b^{\mathcal{I}} + \delta\mathbf{v}, \quad \mathbf{q}_{b,t} = \mathbf{q}_b \otimes \delta\mathbf{q}.$$

We also estimate the accelerometer and gyroscope biases, and thus, we define their errors as

$$\mathbf{a}_{\text{bias},t} = \mathbf{a}_{\text{bias}} + \delta\mathbf{a}_{\text{bias}}, \quad \boldsymbol{\omega}_{\text{bias},t} = \boldsymbol{\omega}_{\text{bias}} + \delta\boldsymbol{\omega}_{\text{bias}}.$$

The kinematics equations for the error-state Kalman filter are

$$\begin{aligned}
\dot{\delta\mathbf{v}} &= -\mathbf{R}_b \left[\mathbf{a}_m - \mathbf{a}_{\text{bias}}\right]_\times \delta\boldsymbol{\theta} - \mathbf{R}_b \delta\mathbf{a}_{\text{bias}} - \mathbf{R}_b \mathbf{a}_n, \\
\dot{\delta\boldsymbol{\theta}} &= -\left[\boldsymbol{\omega}_m - \boldsymbol{\omega}_{\text{bias}}\right]_\times \delta\boldsymbol{\theta} - \delta\boldsymbol{\omega}_{\text{bias}} - \boldsymbol{\omega}_n, \\
\delta\dot{\mathbf{a}}_{\text{bias}} &= \mathbf{a}_w, \\
\delta\dot{\boldsymbol{\omega}}_{\text{bias}} &= \boldsymbol{\omega}_w,
\end{aligned}$$

where $\delta\boldsymbol{\theta}$ is the angle vector error, defined such that $\delta\mathbf{q} = e^{\delta\boldsymbol{\theta}/2}$, $\boldsymbol{\omega}_m$, $\boldsymbol{\omega}_w$ and $\boldsymbol{\omega}_n$ are the gyroscope measurements, bias noise and angular rate noise, respectively, $\boldsymbol{a}_m$, $\boldsymbol{a}_w$ and $\boldsymbol{a}_n$ are the accelerometer measurements, bias noise, and acceleration noise, respectively, and the notation $[\mathbf{r}]_\times$ is the cross products with the vector $\mathbf{r}$ with itself.

The measurement equations depend on the foot contact. We assume that the measurement of the velocity of the foot in contact ($\mathbf{v}_c^{\mathcal{I}}$) with the ground is zero, and use it in the design of the error-state Kalman Filter as follows

$$\mathbf{v}_c^{\mathcal{I}} = \mathbf{R}_b \left[\mathbf{v}_c^{\mathcal{B}} + (\boldsymbol{\omega}_m + \boldsymbol{\omega}_{\text{bias}}) \times \mathbf{p}_c^{\mathcal{B}}\right] + \mathbf{v}_b^{\mathcal{I}} = 0.$$

The position of the foot link in the base link frame is computed using forward kinematics. The orientation of the foot link frame cannot be measured because the joint is passive. Lastly, the joint angles $\mathbf{q}_j$ and their velocities are estimated using encoders.

8.4 Methods: Reinforcement Learning-based Control

In this section, we introduce our approach to controlling the biped using reinforcement learning (RL). We first present the problem formulation and the main algorithm in Sec. 8.4, followed by a discussion of the modifications required to adapt the algorithm for successful deployment on the biped. These include adding

input-output histories, modelling time-delay and deformations, and domain randomization. Where appropriate, we also draw connections to the model-based control framework from Sec. 8.3.

Problem Formulation

We formulate the locomotion problem as a Markov Decision Progress (MDP). The underlying state at time t is defined as $\mathbf{s}_t = \{\mathbf{q}_t, \mathbf{q}_t\}$, with a corresponding observation $\mathbf{o}_t$. The policy, parametrized by $\boldsymbol{\theta}$, is defined as

$$\pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t \mid \mathbf{h}_t\right) \qquad \text{where} \qquad \mathbf{h}_t = \left(\mathbf{o}_{t-h}, \mathbf{a}_{t-h}, \ldots, \mathbf{o}_t\right),$$

and maps a history of past observations and actions to the next action $\mathbf{a}_t$. The learning objective is to find the policy that maximizes the expected cumulative return

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_t\left(\mathbf{s}_t, \mathbf{a}_t\right)\right],$$

where γ is the discount factor and $R_t\left(\mathbf{s}_t, \mathbf{a}_t\right)$ is the reward function. Although the true state $\mathbf{s}_t$ is not fully observed, the policy is trained based on the observed rewards. To optimize the policy, we use the Policy Proximal Optimization (PPO) [29, 30] algorithm. PPO is an on-policy actor-critic method that leverages the temporal difference (TD) error [31] from the critic to estimate the advantage function, which in turn guides the actor's policy updates. In our setup, we use an asymmetric actor-critic variant of PPO, where the critic has access to privileged information (e.g., full simulator state) during training to improve the value estimation, while the actor relies solely on onboard observations available at test time.

Observations and Actions

In contrast to the model-based approach presented in Sec. 8.3, where the control inputs include desired joint positions, velocities, and feedforward torques, our RL framework outputs residuals on the target joint angles. These residuals are defined relative to a static bent-knee configuration in which the robot stands. The residuals are added to this nominal configuration to form the final joint angle targets, which are then tracked by the motors' internal PD controllers to regulate the actual joint motion. Learning to output full desired trajectories (joint positions, velocities, and feedforward torques) can make the policy space unnecessarily large. Residual joint angles reduce this to a compact, interpretable action space.

The observation vector includes a history of the following measurements: linear velocity in the body frame, estimated as in Sec. 8.3, angular velocity given by the

gyroscope, the upward direction vector expressed in the body frame, joystick commands, joint angular velocities, and a foot phase signal that encourages sinusoidal foot motion. During training, noise is added to these measurements to improve robustness. The privileged observation vector extends this by including the same measurements without noise, along with additional information: contact states, actuator forces, foot velocities in the inertial frame, and the duration each foot has been in the air.

Modeling Time Delay

Time delay is an inherent characteristic in many real-world robotic systems, including our biped, due to sensor latency and actuator response time. If not properly accounted for, it causes instabilities. In this work, we account for time delay during training by modeling it. We consider a discrete-time delay of d timesteps between the execution of an action and the corresponding observation of its effect. To incorporate this delay into our learning framework, we modify the history used by the policy to include a delayed action-observation pair

$$\mathbf{h}_t = (\mathbf{o}_{t-h}, \mathbf{a}_{t-h}, \ldots, \mathbf{o}_{t-d}, \mathbf{a}_{t-d}) ,$$

where d is the time delay. Empirically, we found that a time delay of one step ($d = 1$) is sufficient to mitigate instabilities (jitter in the motors) when the policy was deployed on hardware. For the model-based approach presented in Sec. 8.3, we have not found necessary to incorporate any time delay in the planning or control architecture.

Modeling Deformations

We model the deformation by adding two critically damped springs in series with the hip actuator. We use the same $\bar{\mathbf{K}}_p$ as the one learned in the model-based control and planning case (Sec. 8.3) and compute $\bar{\mathbf{K}}_v$ from the critically damped solution in the standard damped harmonic oscillator equation. The robot is then trained with this deformation.

Domain Randomization

For sim-to-real transfer, we apply domain randomization during training. Specifically, we randomize the floor friction coefficient to simulate varying terrain conditions and slip dynamics. We perturb the mass of individual robot links, including the torso, to account for manufacturing variability and changes in payload distribution. To encourage generalization to slight pose estimation or initialization errors, we also

randomize the initial joint positions at the start of each episode. Additionally, we vary the actuator gain parameters ($\mathbf{K}_p$ and $\mathbf{K}_d$). Finally, we perturb the stiffness and damping parameters of the added joints used to model structural deformations, as described in Sec. 8.4.

Rewards

The reward function consists of multiple terms designed to encourage stable, energy-efficient, and task-relevant locomotion. The reward function is formulated as

$$R_t = \text{clip}\left(\sum_{i=1}^{n_r} w_i r_i \Delta t, R_{max}\right),$$

where w_i and r_i are the weight and the reward for each component, and Δt is the discretizations step. The final reward is clipped to remain within reasonable bounds specified by R_{max}.

Rewards for tracking We include tracking rewards that incentivize the robot to follow the commanded linear and angular velocities: a reward for tracking the linear velocity in the horizontal plane and a reward for tracking the yaw angular velocity, both shaped as exponential of the squared error.

Rewards for smoothing To promote energy efficiency and smoothness, we penalize the actuator torques and the rate of change of actions across consecutive time steps.

Rewards for the feet In terms of feet-related behaviors, we include several reward components: a reward for longer feet air time, which encourages proper swing phases; a feet phase reward that encourages sinusoidal foot trajectories aligned with the current gait phase; and a penalty for feet slipping when in contact with the ground, discouraging lateral movement during stance.

Rewards for the posture We penalize deviations of the hip and knee joints from nominal joint angles during walking.

8.5 Hardware Design

The robot has a weight of 7.5 kg and a height of 100 cm. It is composed of two main subsystems: a torso and two legs with elongated feet, as shown in Fig. 8.1. Each leg contains 4 motors, 3 of which are used for the hip and knee actuation,

while the fourth motor is used for the yaw actuation. The robot operates completely autonomously using its onboard computers, sensors, and battery.

When sizing the robot, two primary design criteria must be considered: (1) selecting actuators capable of generating sufficient torque to support the robot's weight and enable locomotion, and (2) ensuring the actuators can achieve the necessary joint velocities to facilitate rapid stepping. The latter is particularly critical given the robot's low center of mass, which necessitates frequent stepping to maintain dynamic stability. These design considerations are discussed in this section.

Legs and Feet Sizing

The legs are designed and built to be lightweight with a low moment of inertia, enabling agile walking. Therefore, the KFE motor is connected at the top, as seen in Fig. 8.3, and transmits the movement to the knee via a belt mechanism. The lower leg connects to the foot through a passive spring-loaded hinge, as the ankles are not actuated. An overview of the leg and its main components can be seen in Fig. 8.3.

A crucial design aspect of the leg lies in the torque produced at the knee joint τ_{knee}, shown in Fig. 8.3. This torque is a function of the mass of the robot m, the leg length L, and the desired knee flexion/extension angle θ. A trade-off against all these parameters was performed to find the optimal torque requirement for the motor. Let m be the total mass of the robot, θ the knee flexion/extension half angle, l the half length of the leg, and g the gravitational acceleration, as shown in Fig. 8.3. The torque at the knee can thus be computed as

$$\tau_{\text{knee}} = mgl\cos\theta.$$

Considering this torque requirement for different operating points and the different COTS motors available, MJBots QDD100 quasi-direct drive brushless motor, equipped with a planetary gear as the transmission reducer, was identified as the best option. The torque at the motor τ_{motor} is increased to match the requirements of the τ_{knee}. For this, we use a belt drive connecting the driver pulley and the driven pulley, mounted at the knee. The driven pulley is 1.5 larger than driver pulley, which increases the torque by a 1.5 factor. In Fig. 8.7, we describe the different operating points for the chosen motor as a function of the leg length and the knee angle.

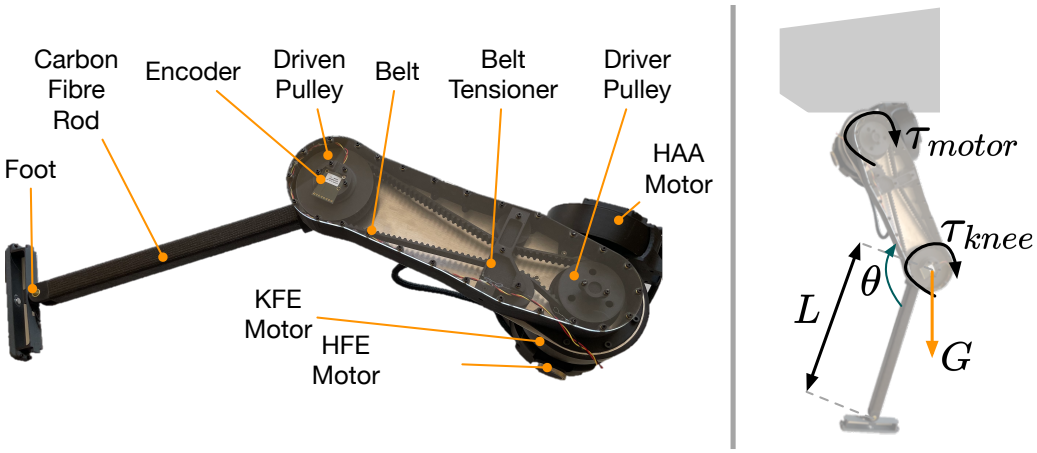


Figure 8.3: Leg structure. (left) Leg design with its main components (right) Leg parameters and torques

Hip Actuation

Apart from the knee motor, each leg contains two additional motors: one controlling the abduction/adduction movement and the other regulating flexion/extension movement. A key requirement for the hip actuation is to ensure the motor is fast enough to enable fast walking and even running. The required speed of the motor r is computed as

$$r_{\text{motor}} = \frac{\theta_{\text{swing}}}{T_s},$$

where θ_{swing} is the maximum swing leg angle. For simplicity, we used the same motor as for the knee, but without the belt drive transmission system.

Yaw Actuation

At the base of the torso, there are two TMotor 6040V motors, which enables each leg to rotate in yaw. Unlike the QDD100 motors used for the other parts, these motors are more lightweight and lack integrated electronics and gears, necessitating the attachment of an external belt-driven gear to yield the desired reduction rate. The chosen reduction rate for these actuators is 1/5. These hip motors are interfaced with an MjBots motor controller, and an external encoder is attached above the motor's shaft.

Lower Leg and Foot

The shin consists of a lightweight carbon fibre tube, which connects to the foot through a hinge. The foot is not actuated, but kept level using a spring. The foot is

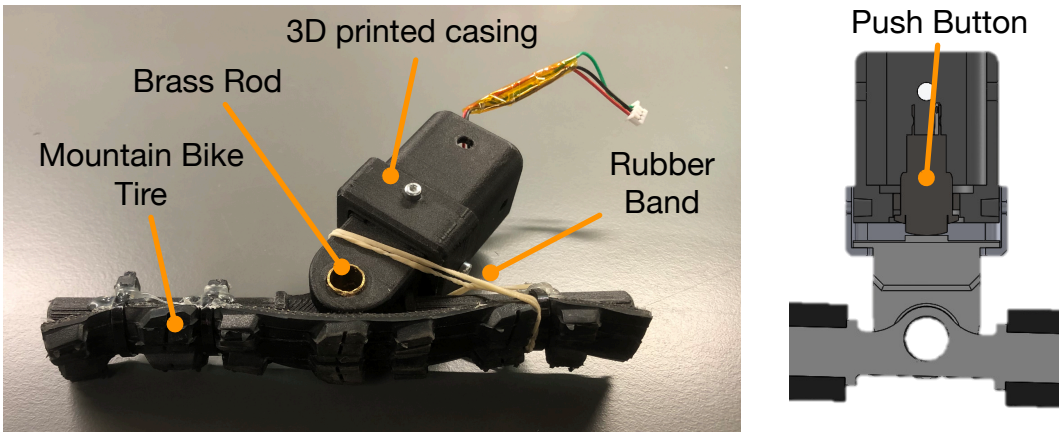


Figure 8.4: Foot Design. Left: Main components of the foot. Right: Close-up of the contact trigger button used to detect impacts. Note: This trigger is used exclusively in the model-based planning and control approach, and not in the reinforcement learning method.

equipped with a button, which allows the robot to detect the impact. An important design aspect was the material of the foot. After testing several options, we decided to use mountain bike tires, as shown in Fig. 8.4, due to their high friction coefficient.

Torso Design and Electronics Components

The torso is build out of several plates of waterjet aluminum connected with 3D printed corners. The torso contains all of the electronics and sensors such as 3 stereo vision cameras, 2 on-board computers, an IMU, power management, and a battery, as well as the yaw motors. In front of the yaw motors, we mounted the 3 cameras, configured to approximate the human field of view—close to 180 degrees. These cameras interface with a Jetson Orin computer equipped with 2048 NVIDIA CUDA cores. This setup enables the execution of lightweight, on-board machine learning algorithms. Additionally, the robot can be tethered via Ethernet to a more powerful external computer for more computationally intensive tasks. The secondary on-board computer is a Raspberry Pi, augmented with a CANFd hat. All motors are daisy-chained, communicating with the Raspberry Pi via the CANFd protocol. The Raspberry Pi has also the role to hardware-synchronize the IMU and the three cameras. Power is generated on-board using a Lithium Ion battery. A power distribution board from MJBots is used to transfer power to all the components. The entire electronics architecture of the robot can be seen in Fig. 8.5.

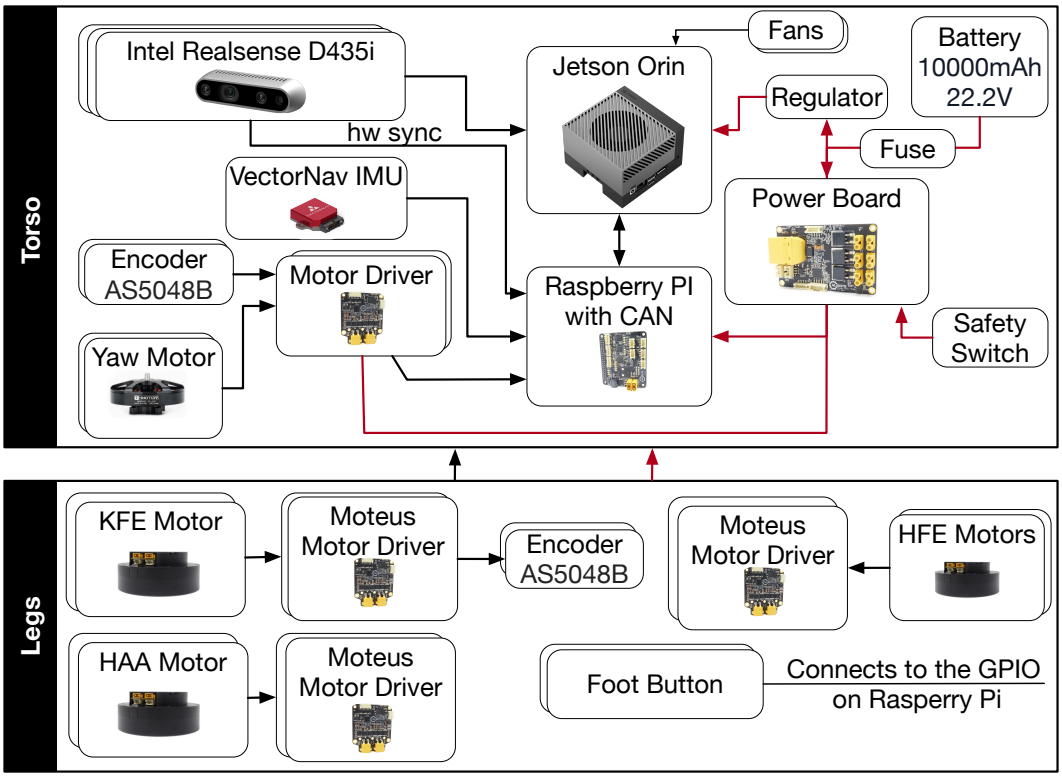


Figure 8.5: Main electronics components of the robot placed in the torso and the legs.

8.6 Empirical Results

Software Architecture

The robot runs ROS 2 on both the Jetson Orin and the Raspberry Pi. For the model-based planning and control approach described in Sec. 8.3, the Jetson Orin handles the execution of the walking controller, including inverse kinematics and dynamics computations. In the reinforcement learning-based approach presented in Sec. 8.4, the PPO policy is also deployed on the Jetson Orin. The Raspberry Pi is responsible for low-level motor control in both cases. Communication between the Jetson Orin and the Raspberry Pi is established via Ethernet. To control the robot in both simulation and hardware, we use an Xbox controller to provide joystick commands.

Simulation Environment

We develop a high-fidelity simulation in MuJoCo [32] based on a URDF model exported from the CAD design of the robot. Accurate mass properties were incorporated into the CAD model by inputting the true weights of most components. As a result, the simulated mass closely matches the physical robot: the CAD-estimated

Table 8.1: Inverse Kinematics Coefficients used in Algorithm 6 and robot parameters

IT_{max}	λ	λ_v	e_{mag}	Δt	T_s [s]	z_c [m]
400	1e-5	1e-5	0.002	0.1	0.25	0.55

weight is 7.354 kg, compared to the actual robot's weight of 7.412 kg. To further align simulation with reality, we modeled the actuators to reflect the hardware, allowing us to use the same control gains in both simulation and hardware. Finally, we incorporated structural deformations observed in the hardware, as described in Sections 8.3–8.4.

Implementation Details for the Model-based Planning and Control

Interstep Foot Trajectories

The problem in (8.7) is put in the form of a Quadratic Program (QP) required by the OSQP package [33], and is solved at a rate of $f = 100$ Hz. The number of steps N in (8.7) is computed as the fraction between the step time and the period $\frac{T_s}{1/f}$.

Inverse Kinematics and Dynamics

The robot's inverse kinematics and dynamics are computed using the open-source software package, Pinocchio, which is a fast implementation of Rigid Body Dynamics algorithms and their analytical derivatives [34, 35]). The parameters used in Algorithm 6 are shown in Table 8.1. One key challenges in selecting these parameters is ensuring the iterative algorithm runs at a consistent rate of 100 Hz and has no delay.

Walking Algorithm

The walking algorithm presented in Algorithm 7 runs at 100 Hz on-board the Jetson Orin.

Implementation Details for the Reinforcement Learning Controller

We implement the training pipeline using JAX [36], leveraging Brax [37], MuJoCo Playground [38], and MuJoCo MJX as frameworks. We simulate 8192 biped environments in parallel, using a batch size of 256. The policy and value function are both modeled as multi-layer perceptrons with hidden layers of sizes [512, 256, 128]. Training is conducted for a total of 200 million steps, although performance tends to

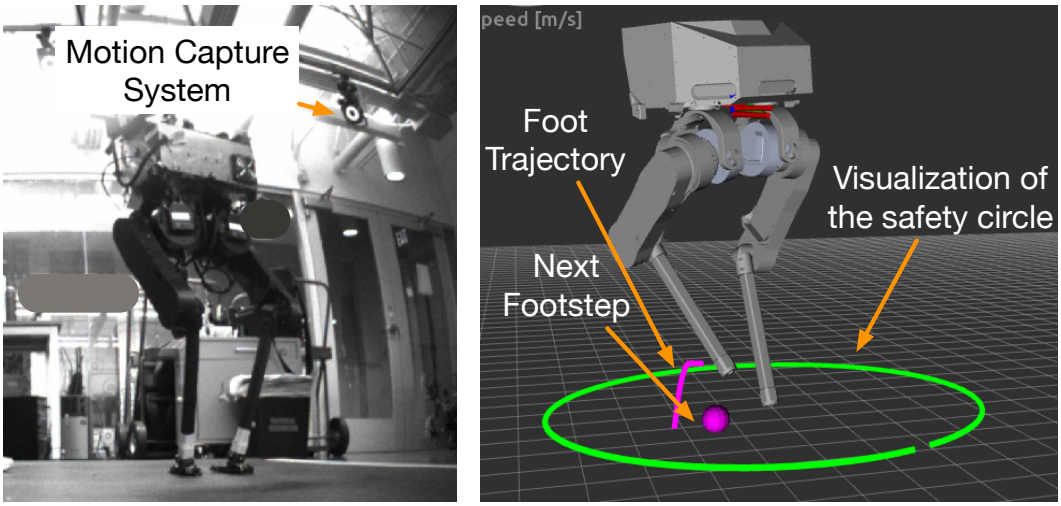


Figure 8.6: Testing setup. a) FLIR camera view of the robot while walking for debugging purposes. b) Visualization of the robot in RViz with markers.

plateau after approximately 15 million steps. To improve stability and learning efficiency, we normalize the observations, privileged inputs, and estimated advantages. At deployment, the policy runs at 100 Hz.

Hardware Experiments

Walking Experiments for the Model-based Planning and Control

We evaluate the performance of the model-based planning and control algorithm (Sec. 8.3) through hardware walking experiments on flat ground and we show qualitative results. As shown in Fig. 8.7, the controller accurately tracks both the position and velocity trajectories of the knee and hip joints for the left and right legs. In Fig. 8.8, we analyze the footstep timing during walking. The top plot shows that the timing of actual foot impacts closely follows the planned contact sequence, computed in (8.6). The middle and bottom plots further illustrate that the DCM remains aligned with its desired DCM.

Walking Experiments with Reinforcement Learning

We also evaluate the reinforcement learning-based controller through hardware walking experiments on flat terrain. As shown in Fig. 8.9, the learned policy successfully enables walking, with the top two plots demonstrating consistent joint position trajectories for the knee and hip joints of both legs. The bottom two plots show the corresponding joint velocities, indicating smooth periodic motion with minimal oscillations.

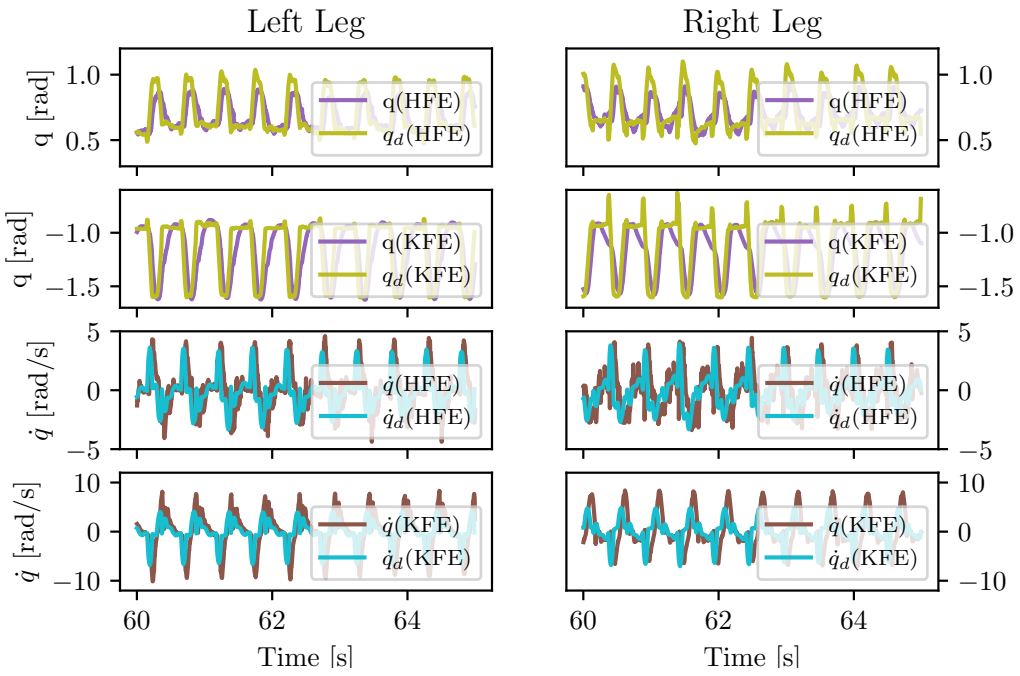


Figure 8.7: Performance of the model-based planning and control algorithm. Position and velocity tracking of the knee and hip joints while walking on flat ground for the left and right leg.

Limitations

We present several limitations of our platform, as follows.

1. Lack of ankle actuators: The robot currently lacks ankle actuators, requiring frequent stepping to maintain balance. This choice was made to simplify the initial design. In future iterations, we plan to incorporate ankle actuators to enhance stability and performance.

2. The "no slip" assumption at the contact point, as shown in equation (8.1), presents a significant limitation. This assumption does not hold on slippery surfaces, where the robot often experiences slips leading to balance loss and falls. This issue highlights the need for revising this assumption to better accommodate real-world conditions.

3. Walking with bent knees: The robot adopts a gait where the knees are bent due to the LIP model assumption. This behaviour is present in the RL framework due to the design of the reward function. This walking style is energy inefficient. Addressing this inefficiency in future designs could improve the robot's energy consumption.

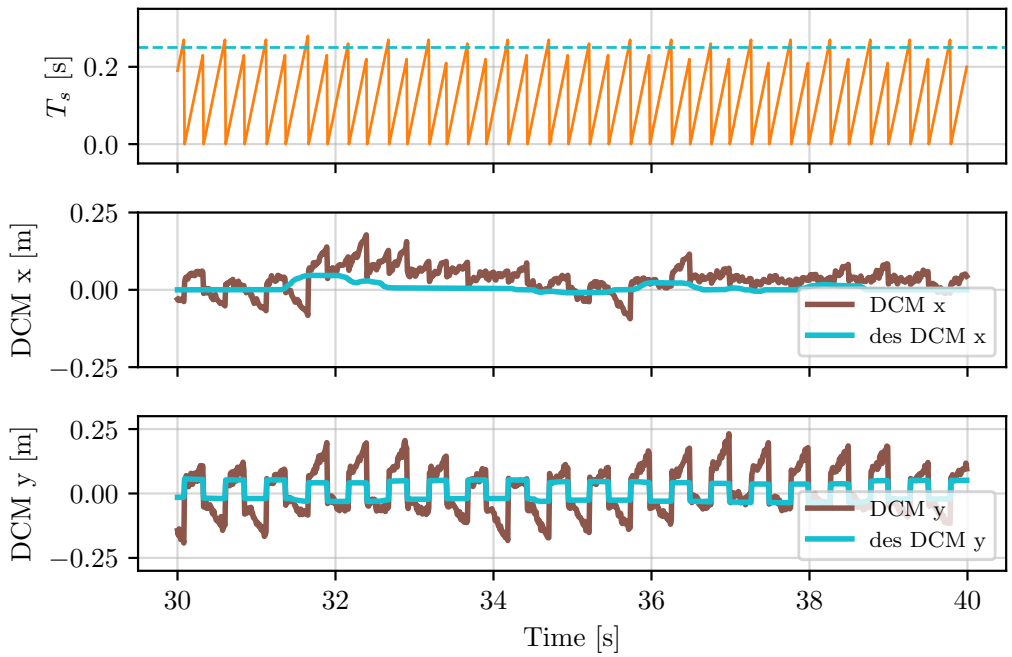


Figure 8.8: Performance of the model-based planning and control algorithm. Top plot: In orange, the actual foot impact time. In blue, the desired impact time. Middle and bottom plots: Comparison of the desired DCM versus actual one during walking on flat ground.

4. For detecting foot impact in the model-based framework, we utilize contact switches, which incorporate springs to register contact. However, these switches pose challenges due to the high acceleration at impact, which can damage the springs. This necessitates considering more durable alternatives in future designs to enhance the robustness of our impact detection system or eliminate the use of buttons altogether.

8.7 Conclusion

We present the design, control, and hardware implementation of a bipedal robot capable of dynamic walking. To address structural deformations, we introduced a compensator block that adjusts joint angles based on estimated deflections in the robot's frame. The hardware design was detailed, with a focus on the leg architecture, hip and yaw actuation, as well as the lower leg and foot components. We implemented two distinct walking control strategies on the same platform: a model-based controller and a reinforcement learning-based controller. Finally, we demonstrated the robot's walking capabilities through hardware experiments conducted on flat terrain.

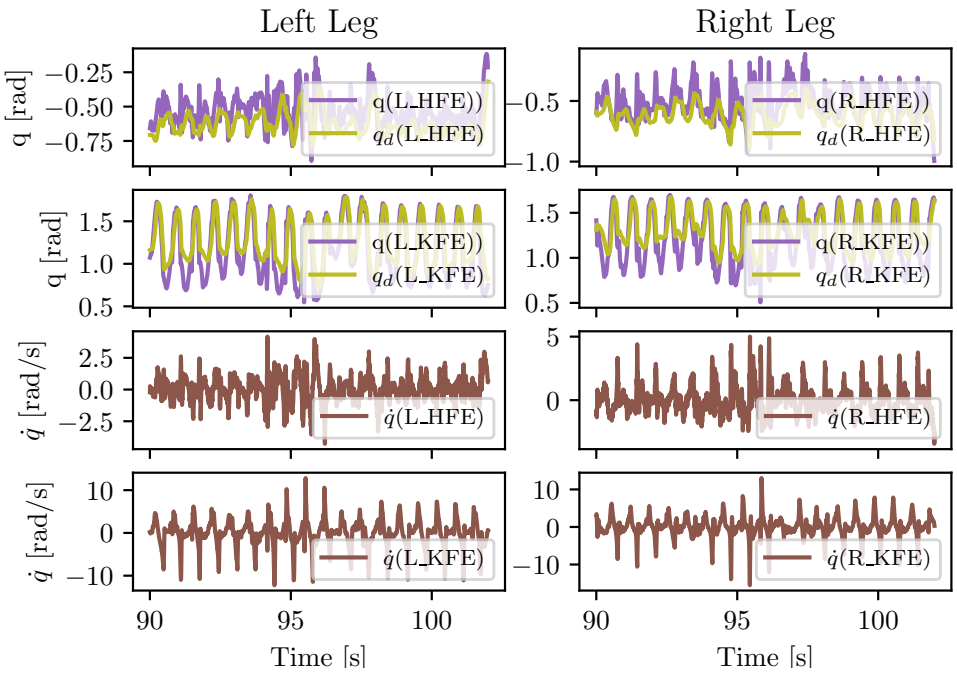


Figure 8.9: Performance of the RL controller. (top two plots) Position tracking of the knee and hip joints while walking on flat ground for the left and right leg. (bottom two plots) Measured velocity for the knee and hip joints.

Lessons Learned

Selecting Motors and their Drivers

Understanding how the specifications of a motor were measured under various operating conditions is crucial for selecting the right component. For instance, when a vendor claims that a motor delivers a specific torque, they must specify the conditions under which this measurement was made. This was a key factor in our decision to choose the MJBots QDD100. The vendor provided detailed operating conditions, as summarized in Table 8.2. These specifications include measurements taken at zero speed in a controlled environment (ambient temperature of 20°C, 24V input, no airflow or heatsink) until thermal limiting occurs. If this information is not clearly provided by the vendor, then bench tests are required.

Another essential criterion was the integration of a motor controller, preferably with a controller area network (CAN) interface, along with open-source firmware and hardware. The availability of open-source hardware was particularly beneficial during testing; if components were damaged, we could quickly replace them using a soldering gun instead of purchasing a new board. This not only saved costs but also enhanced our flexibility in managing hardware issues.

Table 8.2: Torque specifications of the QDD100 motor.

Indefinite	400 [s]	60 [s]	< 1 [s]
3.3 [Nm]	6 [Nm]	10 [Nm]	16 [Nm]

Trajectory Optimization

Initially, we opted for an easier solution to solve the problem presented in (8.7). Specifically, we used a third-order polynomial

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3,$$

where t is the time and a_0, a_1, a_2 and a_3 are the coefficients. For the planar motion, we compute the polynomial coefficients in closed form solution based on the following initial and final conditions: $s(0) = p_0$, $\dot{s}(0) = v_0$, $s(T_s) = p_f$ and $\dot{s}(T_s) = 0$, where p_0 and p_f are the initial and final positions and v_0 is the initial velocity of the swing foot. Note that p_f will be the DCM location. For the vertical motion of the foot, we use a trapezoidal motion profile [39]. For the final condition, we do not impose $\dot{s}(T_s) = 0$, but $\ddot{s}(T_s) = 0$, such that the foot would continue to penetrate into the ground until the impact is detected. While being much simpler in practice, this method posed issues as velocity and position limits could not have been imposed.

Monitoring Motor Sensors

It is important to monitor the temperature and voltage of the motors during operation. The temperature of the motors is a crucial parameter to monitor, as it can indicate when the motor is close to overheating. Fig. 8.10 shows the temperature and voltage of the motors during walking. We can thus conclude that the motors keep an overall low temperature during operations and there are no spikes in the voltage, which would indicate a malfunctioning motor.

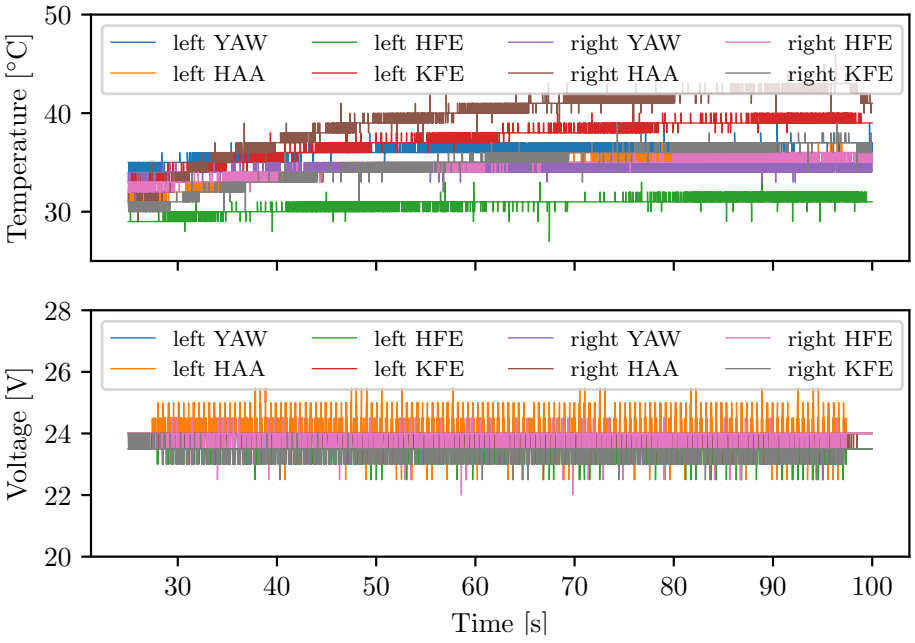


Figure 8.10: Motor sensors (temperature and voltage) during walking

BIBLIOGRAPHY

[1] Mario M. Foglia and Giulio Reina. “Agricultural Robot for Radicchio Harvesting”. In: *Journal of Field Robotics* 23.6-7 (July 2006), pp. 363–377.

[2] “Agricultural Robots for Field Operations: Concepts and Components”. In: *Biosystems Engineering* 149 (2016), pp. 94–111.

[3] Jeffrey Delmerico, Elias Mueggler, Julia Nitsch, and Davide Scaramuzza. “Active Autonomous Aerial Exploration for Ground Robot Path Planning”. In: *IEEE Robot. Autom. Letters* 2.2 (Apr. 2017), pp. 664–671.

[4] Zendai Kashino, Goldie Nejat, and Beno Benhabib. “Aerial Wilderness Search and Rescue with Ground Support”. In: *Journal Intelligent Robotic Syst.* 99.1 (July 2020), pp. 147–163.

[5] Hailong Qin et al. “Autonomous Exploration and Mapping System Using Heterogeneous UAVs and UGVs in GPS-denied Environments”. In: *IEEE Trans. Veh. Technol.* 68.2 (Feb. 2019), pp. 1339–1350.

[6] Abhijit Gadekar et al. “Rakshak: A Modular Unmanned Ground Vehicle for Surveillance and Logistics Operations”. In: *Cognitive Robotics* 3 (Mar. 2023), pp. 23–33.

[7] Vandi Verma et al. “Autonomous Robotics is Driving Perseverance Rover’s Progress on Mars”. In: *Science Robotics* 8.80 (July 2023).

[8] Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan W. Hurst, and Michiel van de Panne. “Feedback Control For Cassie With Deep Reinforcement Learning”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), pp. 1241–1246.

[9] Jonah Siekmann, Kevin R. Green, John Warila, Alan Fern, and Jonathan W. Hurst. “Blind Bipedal Stair Traversal via Sim-to-Real Reinforcement Learning”. In: *ArXiv* abs/2105.08328 (2021).

[10] Patrick Clary, Pedro Morais, Alan Fern, and Jonathan W. Hurst. “Monte-Carlo Planning for Agile Legged Locomotion”. In: *International Conference on Automated Planning and Scheduling*. 2018.

[11] Fangzhou Yu, Ryan Batke, Jeremy Dao, Jonathan W. Hurst, Kevin R. Green, and Alan Fern. “Dynamic Bipedal Maneuvers through Sim-to-Real Reinforcement Learning”. In: *ArXiv* abs/2207.07835 (2022).

[12] Guillermo A. Castillo, Bowen Weng, Wei Zhang, and Ayonga Hereid. “Robust Feedback Motion Policy Design Using Reinforcement Learning on a 3D Digit Bipedal Robot”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 5136–5143.

[13] Marco Hutter et al. “ANYmal - a Highly Mobile and Dynamic Quadrupedal Robot”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Daejeon, South Korea: IEEE, Oct. 2016, pp. 38–44.

[14] Tairan He, Zhengyi Luo, Wenli Xiao, Chong Zhang, Kris Kitani, Changliu Liu, and Guanya Shi. “Learning Human-to-Humanoid Real-Time Whole-Body Teleoperation”. In: *arXiv preprint arXiv:2403.04436* (2024).

[15] Christian Gehring, Péter Fankhauser, L. Isler, R. Diethelm, S. Bachmann, M. Potz, L. Gerstenberg, and Marco Hutter. “ANYmal in the Field: Solving Industrial Inspection of an Offshore HVDC Platform with a Quadrupedal Robot”. In: Jan. 2021, pp. 247–260.

[16] Ali Agha et al. “NeBula: Quest for Robotic Autonomy in Challenging Environments; TEAM CoSTAR at the DARPA Subterranean Challenge”. In: *CoRR* abs/2103.11470 (2021). arXiv: 2103.11470. URL: https://arxiv.org/abs/2103.11470.

[17] Qiayuan Liao, Bike Zhang, Xuanyu Huang, Xiaoyu Huang, Zhongyu Li, and Koushil Sreenath. *Berkeley Humanoid: A Research Platform for Learning-based Control*. 2024. eprint: arXiv:2407.21781.

[18] Matthew Chignoli, Donghyun Kim, Elijah Stanger-Jones, and Sangbae Kim. “The MIT Humanoid Robot: Design, Motion Planning, and Control For Acrobatic Behaviors”. In: *CoRR* abs/2104.09025 (2021). arXiv: 2104.09025. URL: https://arxiv.org/abs/2104.09025.

[19] Junheng Li, Omar Kolt, and Quan Nguyen. *Continuous Dynamic Bipedal Jumping via Real-time Variable-model Optimization*. 2024. arXiv: 2404.11807 [cs.RO].

[20] Yeting Liu, Junjie Shen, Jingwen Zhang, Xiaoguang Zhang, Taoyuanmin Zhu, and Dennis Hong. “Design and Control of a Miniature Bipedal Robot with Proprioceptive Actuation for Dynamic Behaviors”. In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 8547–8553.

[21] Adrian B. Ghansah, Jeeseop Kim, Kejun Li, and Aaron D. Ames. *Dynamic Walking on Highly Underactuated Point Foot Humanoids: Closing the Loop between HZD and HLIP*. 2024. arXiv: 2406.13115 [cs.RO].

[22] Kyunam Kim, Patrick Spieler, Elena-Sorina Lupu, Alireza Ramezani, and Soon-Jo Chung. “A Bipedal Walking Robot that can Fly, Slackline, and Skateboard”. In: *Science Robotics* 6.59 (2021), eabf8136.

[23] Jake Reher, Wen-Loong Ma, and A. Ames. “Dynamic Walking with Compliance on a Cassie Bipedal Robot”. In: *2019 18th European Control Conference (ECC)* (2019), pp. 2589–2595. URL: https://api.semanticscholar.org/CorpusID:131777271.

[24] Jenna Reher and Aaron D. Ames. *Dynamic Walking: Toward Agile and Efficient Bipedal Robots*. 2020. arXiv: 2010.07451 [cs.RO].

[25] Michael Mistry, Jonas Buchli, and Stefan Schaal. "Inverse Dynamics Control of Floating Base Systems Using Orthogonal Decomposition". In: *2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 3406–3412.

[26] Jerry E. Pratt and Russ Tedrake. "Velocity-Based Stability Margins for Fast Bipedal Walking". In: *Fast Motions in Biomechanics and Robotics: Optimization and Feedback Control*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 299–324.

[27] Farhad Aghili. "A Unified Approach for Inverse and Direct Dynamics of Constrained Multibody Systems Based on Linear Projection Operator: Applications to Control and Simulation". In: *IEEE Transactions on Robotics* 21.5 (Oct. 2005), pp. 834–849.

[28] Joan Solà. *Quaternion Kinematics for the Error-State Kalman Filter*. 2017. arXiv: 1711.02508 [cs.RO].

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[30] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. "Trust Region Policy Optimization". In: *CoRR* abs/1502.05477 (2015). arXiv: 1502.05477. URL: http://arxiv.org/abs/1502.05477.

[31] Richard S. Sutton. "Learning to Predict by the Methods of Temporal Differences". In: *Mach. Learn.* 3.1 (Aug. 1988), pp. 9–44. ISSN: 0885-6125. DOI: 10.1023/A:1022633531479. URL: https://doi.org/10.1023/A:1022633531479.

[32] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A Physics Engine for Model-based Control". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst.* Oct. 2012, pp. 5026–5033.

[33] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. "OSQP: An Operator Splitting Solver for Quadratic Programs". In: *Mathematical Programming Computation* 12.4 (Feb. 2020), pp. 637–672. ISSN: 1867-2957.

[34] Justin Carpentier, Florian Valenza, Nicolas Mansard, et al. *Pinocchio: A Fast and Flexible Implementation of Rigid Body Dynamics Algorithms and Their Analytical Derivatives*. https://stack-of-tasks.github.io/pinocchio. 2015–2021.

[35] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. "The Pinocchio C++ Library – A Fast and Flexible Implementation of Rigid Body Dynamics Algorithms and Their Analytical Derivatives". In: *IEEE International Symposium on System Integrations (SII)*. 2019.

[36] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. *JAX: Composable Transformations of Python+NumPy Programs*. Version 0.3.13. 2018. URL: http://github.com/jax-ml/jax.

[37] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. *Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation*. Version 0.12.3. 2021. URL: http://github.com/google/brax.

[38] Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A. Kahrs, Carlo Sferrazza, Yuval Tassa, and Pieter Abbeel. *MuJoCo Playground: An Open-Source Framework for GPU-Accelerated Robot Learning and Sim-to-Real Transfer*. 2025. URL: https://github.com/google-deepmind/mujoco_playground.

[39] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. 1st. USA: Cambridge University Press, 2017.

Chapter 9

CONCLUSIONS

9.1 Summary of Contributions

To enable autonomous operation in unstructured and dynamic settings, two key technologies are essential: **vision-based autonomy** and **real-time adaptation**. Therefore, a core belief I developed during my PhD was that control theory methods, like adaptive control or nonlinear control, should integrate perception. To address this issue, we developed MAGIC-VFM, published in IEEE Transactions of Robotics [1]. To model and adapt to ground disturbances, this work introduces a meta-learning algorithm with a visual foundation model [2], which is integrated with composite adaptive control at runtime to: (1) adapt to unseen terrain changes and (2) adjust to internal robot dynamics, such as faults. For this method, we also prove exponential stability and robustness against model errors using Lyapunov theory. We are able to achieve these mathematical guarantees by enforcing the Lipschitz continuity [3] of the deep neural network used as the basis function of the adaptive controller. A key finding is that a learned environment model significantly reduces the tracking error (the error between the desired trajectory and the actual trajectory), and incorporating a visual foundation model enhances performance by enabling the robot to better understand the terrain it operates on.

A key motivation driving this work, one I personally prioritized, was to demonstrate that methods with strong empirical validation in robotics can be adapted for space applications. My goal was to bridge the gap between the traditionally conservative space domain and the rapid, iterative development practices of the robotics community. Therefore, the second part of the thesis focuses on the problem of detecting spacecraft parts in space using thermal and RGB cameras. We propose a method for knowledge distillation of a Visual Foundation Model, which is a technique to transfer knowledge from a large teacher network to a smaller student network. We show that the student network is able to achieve very good performance on the task of detecting spacecraft parts in space using both thermal data and RGB data. We launched our algorithms in space on-board a Jetson TX2 inside the Edgenode Lite cubesat.

Enabling rapid space exploration requires careful consideration of the robot's form

factor, particularly for systems with multiple degrees of freedom. The third part of this thesis focuses on the design and control of two such platforms: Leonardo and a custom-designed biped. For Leonardo, my primary contributions include the development of a nonlinear tracking controller with adaptation to enhance its walking performance, a slacklining foot planner, and the execution of the hardware experiments. For the biped, the thesis presents both the hardware design and the planning and control pipeline using two methods: model-based and RL, developed to support agile locomotion.

9.2 Open Questions

Based on the research presented in this thesis, we have identified several open questions that could be addressed in the near-future work. These questions are related to the limitations of the current work presented in this thesis. In Sec. 9.3, we present a series of future directions that could be pursued to address these limitations and further advance the state-of-the-art in the field of robotics. These directions are on a larger timescale and are not limited to the specific research presented in this thesis.

MAGIC-VFM

- Investigate the performance of the MAGIC-VFM controller for more diverse sets of terrains and environments. Extend the experiments to planning on real-world robots.
- Extend the framework to legged robots by learning the residual contact force model based on the terrain features. Some preliminary work addressing this can be seen in Appendix C.
- Embed uncertainty quantification into the MAGIC-VFM framework to account for the uncertainty in the terrain model. Use this to generate safe trajectories in a planning framework. Some preliminary work addressing this can be seen in Appendix D.
- Investigate a suitable way to merge the features from a VFM into an elevation map as the robot is traversing the environment and investigate an efficient way to store the VFM features in the elevation map.
- Develop a better controller for the car with front steering that does not decouple the longitudinal and lateral dynamics.

Vision-based Detection of Spacecraft Parts

- Investigate other methods of doing knowledge distillation to improve the performance of the student network.
- Investigate the robustness of distilling with DINO, which is trained on RGB data, when the student network is further trained on thermal data.
- Extend the detection algorithms to other areas of interest, such as graspable areas for manipulation tasks in space.

Leonardo

- Currently the robot supports about 50% of its weight with its legs. The remaining weight is supported by the propellers, which makes the robot be energy inefficient. The robot should be able to support its weight with its legs and use the propellers only for stabilization or flying.
- Development of a more robust foot design.
- Development of a high level hybrid planner that is able to plan trajectories for the robot and switch between different modes of locomotion based on the environment.

Biped

- Currently, the legs have some elasticity caused by the 3D printed materials. This elasticity is modeled in the controller and compensated for. A better solution would be to use more rigid materials for the legs, but they come at the price of being harder to manufacture.
- Adding ankle actuators to the robot would allow it not have to constantly step in place to maintain balance.
- Development of better, more lightweight feet that are able to adapt to different terrains.

9.3 Future Directions

Algorithms for Learning-based Control and Planning

Problem 1: Understanding the Reward Process for Agents that can Learn Continuously

The reward function plays an important role in planning, as it impacts the gradients that guide the policy learning. Reinforcement Learning (RL) agents often discover unintended strategies to maximize this reward [4], which makes understanding the

impact of reward choices complex. Moreover, sparse rewards pose a significant challenge in RL. For instance, when training a humanoid robot, a sparse reward like accomplishing a goal, for example "opened door", is unlikely to succeed. The robot's initial actions are likely to result in falling, providing little useful feedback. Humans often experiment with different reward functions until the desired behavior is achieved. Sparse rewards are frequently replaced with dense, intermediary rewards to provide better guidance; for example, a robot might use a gait library [5]. Rewards are also commonly adjusted during training as learning progresses.

Problem 2: "A major goal of artificial intelligence (AI) is to understand how an AI agent can obtain and reason with a high-level model of the world." [6]. Oftentimes, the agent needs to execute multiple tasks at different time scales [7]. For instance, a walking robot may need to walk or run (which is a low-level task), open the door (higher level task), and then eventually, recharge itself once other goals are completed. The design and practical implementation of algorithms capable of coordinating such diverse tasks remain not fully understood.

Problem 3: Distributional Reinforcement Learning

The models used for planning are often inaccurate. This is caused by insufficient data or the use of function approximators that generalize imperfectly [4]. The reliance on a single point estimate for the model during planning can pose problems because planning algorithms are designed to optimize performance within the constraints of the given model, making them particularly effective at exploiting even small inaccuracies in the dynamics [8]. As a result, these errors propagate and significantly impact the resulting policy.

Problem 4: Novel Hardware Designs

Advanced algorithms alone are not sufficient for enabling robots to perform complex tasks; equally critical is the development of hardware capable of supporting these algorithms. This highlights the need for research in robotics hardware, spanning novel materials, actuation systems, and overall mechanical design. In the context of bipedal locomotion, one persistent challenge is the design of adaptive feet. The human foot integrates muscles, tendons, and bones to provide both stability and adaptability across diverse terrains. With the same hardware, humans can transition between various locomotion modes, such as walking and running, while leveraging highly developed tactile sensing in the feet to perceive surface properties. In contrast, robotic feet are often rigid and lack sensing capabilities. While promis-

ing innovations have emerged, such as biologically inspired foot mechanisms [9], they remain far from replicating the functionality of the human foot and are not yet widely adopted. Similarly, motor design offers substantial room for improvement, particularly through the integration of compliant elements like springs.

Another underexplored area is distributed tactile sensing. Human skin provides continuous, high-resolution feedback that enables precise manipulation and interaction with the environment. Replicating such distributed sensing in robots could significantly enhance their dexterity and adaptability.

Finally, incorporating additional sensory modalities, such as auditory perception, could expand robotic capabilities. Humans rely on sound cues for tasks ranging from spatial awareness to social interaction. Enabling robots to process and act upon auditory information could open new avenues for perception and control in unstructured environments.

BIBLIOGRAPHY

[1] Elena Sorina Lupu, Fengze Xie, James A. Preiss, Jedidiah Alindogan, Matthew Anderson, and Soon-Jo Chung. *MAGIC-VFM: Meta-learning Adaptation for Ground Interaction Control with Visual Foundation Models*. 2024. arXiv: 2407.12304 [cs.RO].

[2] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. "Emerging Properties in Self-Supervised Vision Transformers". In: *CoRR* abs/2104.14294 (2021). arXiv: 2104.14294. URL: https://arxiv.org/abs/2104.14294.

[3] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. "Spectral Normalization for Generative Adversarial Networks". In: abs/1802.05957 (2018). arXiv: 1802.05957. URL: http://arxiv.org/abs/1802.05957.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[5] Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, and Michiel van de Panne. *Feedback Control For Cassie With Deep Reinforcement Learning*. 2018. arXiv: 1803.05580 [cs.RO].

[6] Richard S. Sutton, Marlos C. Machado, G. Zacharias Holland, David Szepesvari, Finbarr Timbers, Brian Tanner, and Adam White. "Reward-Respecting Subtasks for Model-Based Reinforcement Learning". In: *CoRR* abs/2202.03466 (2022). arXiv: 2202.03466.

[7] Richard S. Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial Intelligence* 112.1 (1999), pp. 181–211.

[8] Andrea Krause. *Probabilistic Artificial Intelligence (Lecture Notes)*. 2023.

[9] Nisal Perera, Shangqun Yu, Daniel Marew, Mack Tang, Ken Suzuki, Aidan McCormack, Shifan Zhu, Yong-Jae Kim, and Donghyun Kim. *StaccaToe: A Single-Leg Robot that Mimics the Human Leg and Toe*. 2024. arXiv: 2404.05039 [cs.RO].

Appendix A

EXTENSIONS OF CHAPTERS II-V: MAGIC-VFM

Note: This chapter is based on unpublished work.

In this section, we present several extensions of the MAGIC-VFM algorithm. The first extension (Appendix A.1) adapted specific components of the MAGIC-VFM controller for a high-speed racing car, as part of the Indy Autonomous Challenge. The second extension presents the hardware experiments conducted for the Gradient-based Adaptive Policy Selection (GAPS) algorithm applied to a small-scale RC racing car [1].

A.1 Model-Based Control for Autonomous Racing Cars

Figure A.1: The Caltech Indy Autonomous Challenge car.

I was briefly involved in the Indy Autonomous Challenge where I implemented the tracking controller for a full-scale high-speed racing car, as well as improved the motion planner by integrating a KD-tree for efficient nearest-neighbor search of the next waypoint. In this subsection, I present results from preliminary tests conducted at the Kentucky Motor Speedway in November 2024. Note that the system was further improved and robustified for the actual competition in the Las Vegas track in January 2025 where the car achieved a speed of 145 mph.

Motivation

Autonomous vehicles deployed in urban environments, such as Waymo, are designed to navigate safely and efficiently, focusing on low-speed maneuvers and complex interactions with pedestrians and other traffic elements. In contrast, Tesla's autopilot technology presents advancements in autonomous highway driving, though it still needs the presence of a human driver ready to intervene when needed. On the other end of the spectrum, race cars are designed to maximize performance, often pushing the limits of their sensors and control systems. Racing offers a testing ground for autonomy algorithms to operate under extreme conditions. The Indy Autonomous Challenge exemplifies this, providing a platform for university teams globally to compete and advance technologies that will lead to the commercialization of fully autonomous vehicles and enhance the deployment of advanced driver-assistance systems (ADAS) for improved safety and performance. Caltech joined this competition in late 2024 with our vehicle illustrated in Fig. A.1.

Modelling

We model the dynamics of the car as in (3.24)-(3.25). Compared to a Traxxas RC electric car used in the MAGIC-VFM article, the Indy car's transmission system is much more complex, with the control inputs being not only the steering angle, but also throttle, brake, and gear.

Steering Controller

We define the tracking errors with respect to the desired trajectory as in (3.26)-(3.28). Using these error definitions, we can write the dynamics as

$$\begin{aligned}\dot{\mathbf{e}} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{C_y}{m v_x^{\mathcal{B}}} & \frac{C_y}{m} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\frac{L^2 C_y}{2 I_z v_x^{\mathcal{B}}} \end{bmatrix} \mathbf{e} + \begin{bmatrix} 0 \\ \frac{C_y}{m} \\ 0 \\ \frac{C_y L}{2 I_z} \end{bmatrix} u_\delta + \begin{bmatrix} 0 \\ -v_x^{\mathcal{B}} \\ 0 \\ -\frac{L^2 C_y}{2 I_z v_x^{\mathcal{B}}} \end{bmatrix} \omega_d, \\ &:= \mathbf{A}(v_x^{\mathcal{B}}(t))\mathbf{e} + \mathbf{B}_1 u_\delta + \mathbf{B}_2 \omega_d,\end{aligned} \tag{A.1}$$

where $\mathbf{e} := [e^\perp, \dot{e}^\perp, e_2, \dot{e}_2]$ is the error vector and ω_d is the desired angular velocity of the trajectory. Recall from Sec. 3.5 that $e^\perp$ the distance of the center of gravity of the vehicle from the desired trajectory, e_2 is the orientation error as a difference between the actual orientation ψ and the desired orientation ψ_d, C_y is the cornering stiffness coefficient, L is the car length, m is the mass of the vehicle, I_z is the moment of inertia about the vertical axis, $v_x^{\mathcal{B}}$ is the longitudinal velocity of the

vehicle, and u_δ is the steering angle input. First, we perform the reachability test for the system (A.1) to ensure that the system is controllable. For this, we show that the reachability matrix is full rank

$$\text{rank}\begin{bmatrix} \mathbf{B}_1 & \mathbf{A}\mathbf{B}_1 & \mathbf{A}^2\mathbf{B}_1 & \mathbf{A}^3\mathbf{B} \end{bmatrix} = 4,$$

which is a necessary condition for the system to be controllable. We then design a controller for the system (A.1) using the following control law

$$u_\delta = -\mathbf{K}\mathbf{e} + \delta_{\text{ff}}, \tag{A.2}$$

with $\mathbf{K} \in \mathbb{R}^4$, a diagonal matrix with positive values k_1, k_2, k_3, k_4 and δ_{ff}, a feedforward term. This term is designed to compensate for the effect of the disturbance term $\mathbf{B}_2\omega_d$ in the dynamics (A.1). Note that the effect cannot be fully canceled, as shown next. We will follow a similar technique as in [2]. The steady state of the system is computed as

$$\mathbf{e}_{\text{ss}} = -(\mathbf{A} - \mathbf{B}_1\mathbf{K})^{-1}(\mathbf{B}_1\delta_{\text{ff}} + \mathbf{B}_2\omega_d),$$

where we ensure $\mathbf{K}$ is selected such that the matrix is invertible. After further manipulation, we obtain the following steady state error ψ_e

$$\psi_{e,ss} = \frac{L}{R} - \frac{m(v_x^{\mathcal{B}})^2}{C_y R},$$

where we used the fact that $\omega_d = \frac{v_x^{\mathcal{B}}}{R}$, with R being the radius of the trajectory. We notice that δ_{ff} cannot fully cancel the effect of the affine term in (A.1), as the steady state error $\psi_{e,ss}$ is independent of δ_{ff}. We will therefore select δ_{ff} such that $e^\perp \to 0$ as $t \to \infty$.

Theorem 4 *By applying the controller in* (A.2) *to the dynamics that evolve according to* (A.1) *using the feedforward term*

$$\delta_{\text{ff}} = \frac{L}{R}(1 - k_3) + k_3\frac{m(v_x^{\mathcal{B}})^2}{C_y R}, \tag{A.3}$$

the tracking error converges to a bounded error ball given by

$$\lim_{t\to\infty} \|\mathbf{e}\|_2 \leq \frac{1}{\lambda_{\max}(\mathbf{A} - \mathbf{B}_1\mathbf{K})}\sqrt{4\left|\frac{\xi\,(k_3 - 1.0)}{mR}\right|^2 + \left|\frac{Lk_3\xi}{I_z R}\right|^2}, \tag{A.4}$$

where $\xi := C_y L - m(v_x^{\mathcal{B}})^2$ *is defined for ease of notation.*

Proof 6 *The closed-loop system is*

$$\dot{\mathbf{e}} = (\mathbf{A} - \mathbf{B}_1\mathbf{K})\mathbf{e} + \mathbf{B}_1\delta_{\text{ff}} + \mathbf{B}_2\omega_d.$$

Using a Lyapunov function as $\mathcal{V} = \mathbf{e}^\top\mathbf{e}$*, and computing its time derivative, we obtain*

$$\dot{\mathcal{V}} = \mathbf{e}^\top(\mathbf{A} - \mathbf{B}_1\mathbf{K})\mathbf{e} + \mathbf{e}^\top\mathbf{B}_1\delta_{\text{ff}} + \mathbf{e}^\top\mathbf{B}_2\omega_d,$$

where we select $\mathbf{K}$ *such that* $\mathbf{A} - \mathbf{B}_1\mathbf{K}$ *is Hurwitz. We then bound the terms as follows:*

$$\begin{aligned}\dot{\mathcal{V}} &\leq \lambda_{\max}(\mathbf{A} - \mathbf{B}_1\mathbf{K})\|\mathbf{e}\|_2^2 + |\mathbf{e}^\top(\mathbf{B}_1\delta_{\text{ff}} + \mathbf{B}_2\omega_d)|, \\ &\leq \lambda_{\max}(\mathbf{A} - \mathbf{B}_1\mathbf{K})\|\mathbf{e}\|_2^2 + \|\mathbf{e}\|_2\|\mathbf{B}_1\delta_{\text{ff}} + \mathbf{B}_2\omega_d\|_2.\end{aligned} \tag{A.5}$$

Given that $\dot{\mathcal{V}} = \frac{d}{dt}(\|\mathbf{e}\|_2^2)$*, we can further simplify* (A.5) *as*

$$\frac{d}{dt}\|\mathbf{e}\|_2 \leq \lambda_{\max}(\mathbf{A} - \mathbf{B}_1\mathbf{K})\|\mathbf{e}\|_2 + \|\mathbf{B}_1\delta_{\text{ff}} + \mathbf{B}_2\omega_d\|_2.$$

Using Comparison Lemma [3], we conclude that the tracking error norm converges to a bounded error ball defined by (A.4).

Forward Velocity Controller Design

We model the forward velocity component by considering the aerodynamic forces and the rolling friction acting on the vehicle. The dynamics thus becomes:

$$m\dot{v}_x = u - mgC_{\text{r}}\,\text{sgn}(v_x) - \frac{1}{2}\rho C_{\text{d}}Av_x|v_x|, \tag{A.6}$$

where for ease of notation, we omit the frame notation $\mathcal{B}$ for the forward velocity. Here, u is a virtual input, m is the vehicle's mass, g is the gravitational acceleration, C_{r} is the rolling resistance coefficient, ρ is the air density, C_{d} is the drag coefficient, and A is the frontal area of the vehicle. Note that $\text{sgn}(v_x)$ is replaced by $\tanh(v_x)$ to avoid stiffness issues during numerical integration. We can thus write the true system dynamics in (A.6) as

$$m\dot{v}_x = u - \boldsymbol{\phi}(v_x)\mathbf{a},$$

where the basis function is $\boldsymbol{\phi}(v_x) = \begin{bmatrix} mg\tanh(v_x) & \frac{1}{2}v_x|v_x| \end{bmatrix}^\top$ and $\mathbf{a} = [C_r, \rho C_d A]$ are the true parameters. Let the estimated value of $\mathbf{a}$ be $\hat{\mathbf{a}}$ and define the error function $s_v = v_x - v_{x,\text{ref}}$, with $v_{x,\text{ref}}$ the reference velocity. We design an adaptive controller such that $s_v \to 0$, as follows

$$u = m\dot{v}_{x,\text{ref}} - \boldsymbol{\phi}(v_x)\hat{\mathbf{a}} - k_v s_v, \tag{A.7}$$

Table A.1: Racing vehicle parameters and control parameters

C_y	m	I_z	τ	L	k_1	k_2	k_3	k_4	k_I
70000	815.11	800.0	0.1	2.97	0.97	0.72	4.33	0.0	0.8

where k_v is a positive constant. We then design a tracking-based adaptive law for $\hat{\mathbf{a}}$ as $\dot{\hat{\mathbf{a}}} = -\mathbf{\Gamma}\mathbf{\Phi}s_v$, where $\mathbf{\Gamma}$ is a positive definite matrix. To convert A.7 into a throttle and brake command, we use a mapping function that converts the control input u into throttle and brake commands, as follows

$$u = \begin{cases} k_T u_T & \text{if } u \geq 0, \\ k_B u_B & \text{if } u < 0, \end{cases} \tag{A.8}$$

where k_T and k_B are positive constants, derived from system identification, and u_T and u_B are the throttle and brake commands, respectively.

Motion Planning

The race lines and pits are computed offline using [4]. The output from the trajectory are a list of waypoints that the vehicle should follow, a heading angle, a forward velocity profile, and a curvature profile. We use a KD-tree to efficiently search for the nearest point in the trajectory to the vehicle's current position. The desired forward velocity is $v_{x,\text{ref}}$ in (A.7), and the desired heading angle from the trajectory is used to compute the orientation error e_2 in (A.1).

Empirical Results

Simulation Results We setup a simulation environment using (3.24)-(3.25) to validate the steering controller in (A.2) and ensure the feedforward term performs as intended. The parameters of the vehicle are presented in Table A.1. The forward velocity is modeled as a first order time delay system and the corresponding controller resembles (3.14) without adaptation. The input trajectory is a circle with desired radius and velocity of 40 m/s. We present the results in Fig. A.2, and show that without the feedforward term, the vehicle's perpendicular error $e^{\perp}$ does not converge to 0. Using feedforward and adaptation, $e^{\perp} \to 0$, the convergence rate is higher.

Hardware Results We present the results of the steering controller in (A.2) without the feedforward term and the forward velocity controller in (A.7) with constant

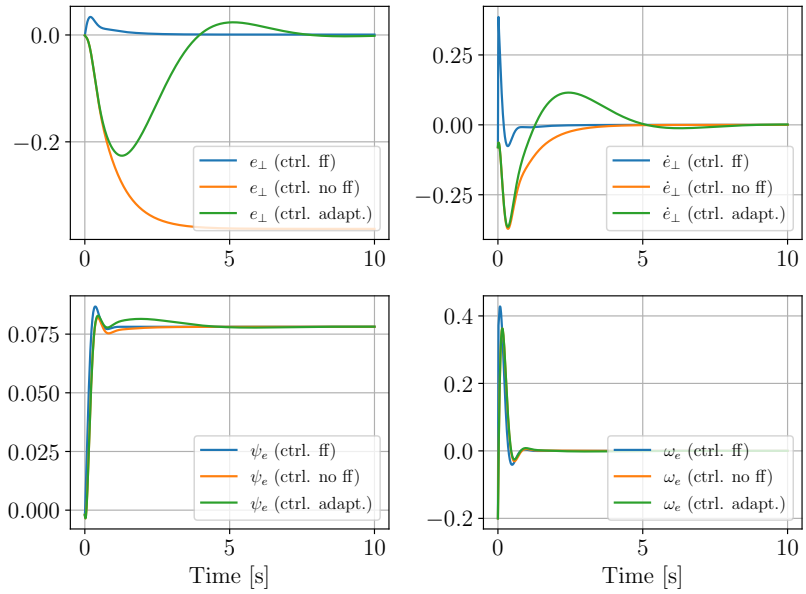


Figure A.2: Comparison between the controller in (A.2) with and without feedforward and with adaptation of $e^{\perp}$.

basis function $\boldsymbol{\phi}$ on-board Dallara vehicle (Fig. A.1). For the throttle and brake commands we use (A.8).

The vehicle was tested at the Kentucky Motor Speedway to follow a pre-defined race line, computed offline using [4]. The performance of the system is shown in Fig. A.3, which presents the actual path of the vehicle compared to the planned trajectory. In Fig. A.4, we present the tracking of the forward velocity as well as the errors terms from (A.1). Note that the maximum speed that the vehicle was tested on was 35 m/s, which corresponds to 80 mph.

Acknowledgements Dr. Matt Anderson for the hardware and software interfaces, Deemo Chen and Prof. Xingxing Zuo for the state estimation, John Lathrop for the trajectory planning, Joshua Cho for the help on the controller and testing, as well as the Caltech Racing Team for the support and collaboration.

A.2 Experiments for the Gradient-based Adaptive Policy Selection (M-GAPS) Algorithm

Introduction

Gradient-based Adaptive Policy Selection (M-GAPS) [5] is a non-episodic model-based reinforcement learning algorithm that studies the problem of online adaptive

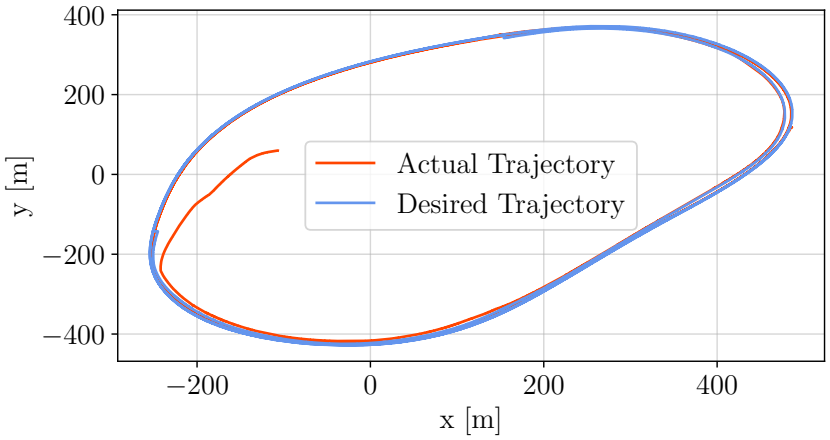


Figure A.3: Results from preliminary tests at the Kentucky Motor Speedway. Actual path vs. planned trajectory.

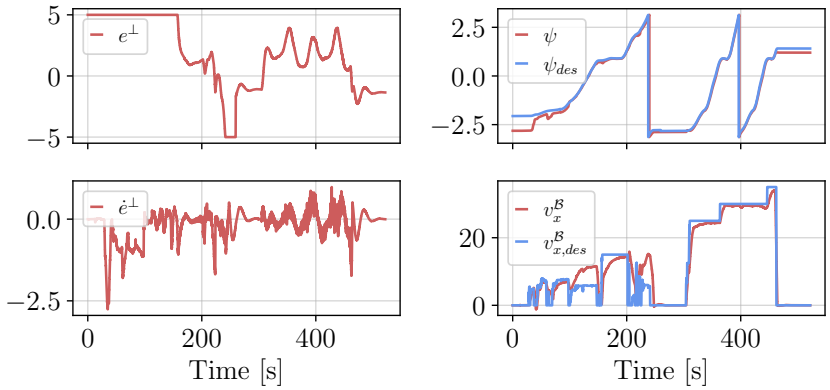


Figure A.4: Forward velocity tracking as well as the errors terms from (A.1) for the actual racing car.

policy selection for nonlinear time-varying dynamical systems. The algorithm requires the knowledge of 4 components: (1) a dynamics model $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$, (2) a policy class $\pi(\mathbf{x}_t, \boldsymbol{\theta}_t)$, (3) a cost function $C_t(\mathbf{x}_t, \mathbf{u}_t)$ and (4) access to the partial derivatives of the dynamics and cost along the visited trajectories. The end goal is to minimize the total cost over a finite time horizon T: $\sum_{t=0}^{T} C_t(\mathbf{x}_t, \mathbf{u}_t)$. The online policy selection algorithm optimizes this total cost by selecting the parameters $\boldsymbol{\theta}$ sequentially.

Results

We demonstrate the effectiveness of the M-GAPS algorithm on-board a vehicle with Ackermann steering. The model used is described in (3.24)-(3.25) and the steering and velocity controllers (policies) in (A.2) and (A.7), respectively. The parameters

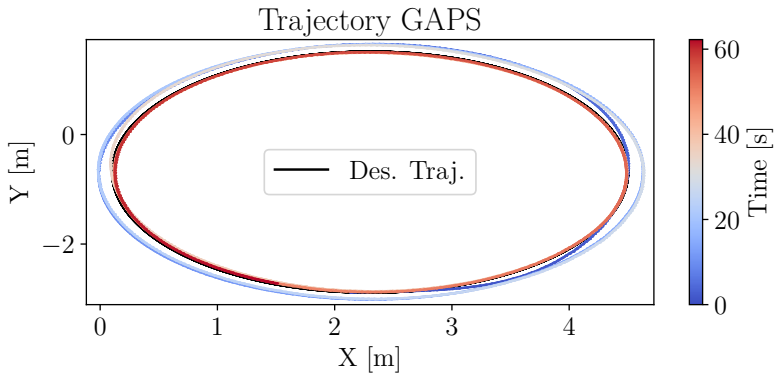


Figure A.5: Planar trajectory color-coded by time to illustrate trajectory tracking improvement.

$\boldsymbol{\theta}$ of the policy are k_1, k_2, k_3, k_4, which parametrize the steering controller and k_v, which parametrizes the velocity controller, as explained in Appendix A.1. During the experiment run, conducted at CAST, M-GAPS performs real-time adjustments of the controller parameters to minimize a certain quadratic cost function defined as

$$\text{cost} = \mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{u}^\top \mathbf{R}\mathbf{u}, \tag{A.9}$$

where $\mathbf{x}$ is the state of the robot and $\mathbf{u}$ is the control input (the steering angle and the forward acceleration). See Chapter 3 for more details on the definitions of $\mathbf{x}$ and $\mathbf{u}$. In this experiment, the algorithm adjusted the controller parameters, effectively performing gain tuning as described in control theory. However, the approach is more general and can be applied to optimize any parameters of a policy, not just gains. Initially, the robot started driving in a circular trajectory using manually optimized gains. After a short period, the M-GAPS algorithm was activated to dynamically adjust the gains k_1, k_2, k_3, k_4, and k_v to further reduce the tracking error. Fig. A.5 illustrates the planar position over time, with the trajectory color-coded by time to visualize the improvement in trajectory tracking under the influence of the M-GAPS algorithm. In Fig. A.6, we outline the cost decrease under the M-GAPS algorithm and the dynamic adjustments of the parameters of the policy. For another experiment, in Fig. A.7, we emphasize the tracking error decrease when M-GAPS is enabled. The M-GAPS code was implemented using Symforce, which is a fast symbolic computation and code generation library for robotics applications [6]. This implementation choice is driven by the need for gradients of the dynamics and cost functions, which are needed for the algorithm's functionality.

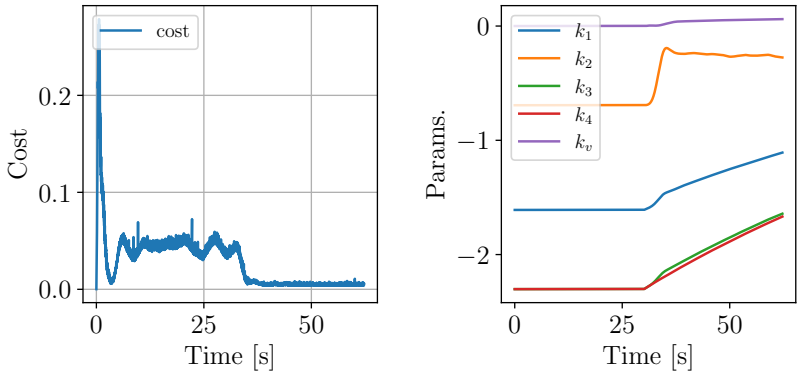


Figure A.6: Performance for M-GAPS. (left plot) Cost decrease with the M-GAPS algorithm. (right plot) Dynamic adjustments of the parameters of the policy.

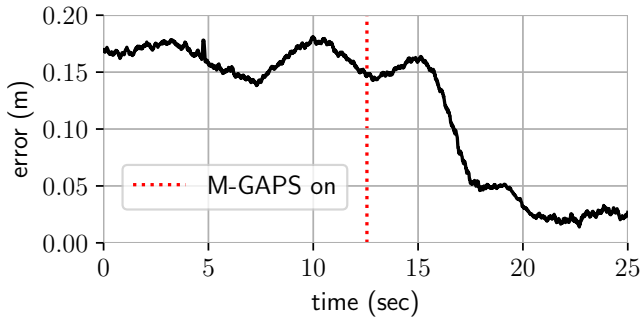


Figure A.7: Tracking error decrease when M-GAPS is enabled (Figure generated by Prof. James Preiss)

Acknowledgments Prof. James Preiss for leading the project and providing the initial code base, as well as guidance throughout the development, Fengze Xie for translating an earlier version of my Python code into the Symforce framework, and Jedidiah Alindogan for the help with the hardware.

BIBLIOGRAPHY

[1] Yiheng Lin, James A. Preiss, Fengze Xie, Emile Anand, Soon-Jo Chung, Yisong Yue, and Adam Wierman. *Online Policy Optimization in Unknown Nonlinear Systems*. 2024. arXiv: 2404.13009 [math.OC].

[2] Rajesh Rajamani. *Vehicle Dynamics and Control*. Jan. 2006.

[3] Hassan K. Khalil. *Nonlinear Control*. Always Learning. Pearson, 2014.

[4] Alexander Heilmeier, Alexander Wischnewski, Leonhard Hermansdorfer, Johannes Betz, Markus Lienkamp, and Boris Lohmann. "Minimum Curvature Trajectory Planning and Control for an Autonomous Race Car". In: *Vehicle System Dynamics* 58.10 (2020), pp. 1497–1527.

[5] Yiheng Lin, James A. Preiss, Emile Anand, Yingying Li, Yisong Yue, and Adam Wierman. "Online Adaptive Policy Selection in Time-Varying Systems: No-Regret via Contractive Perturbations". In: *Advances in Neural Information Processing Systems*. Vol. 36. Curran Associates, Inc., 2023, pp. 53508–53521.

[6] Hayk Martiros, Aaron Miller, Nathan Bucki, Bradley Solliday, Ryan Kennedy, Jack Zhu, Tung Dang, Dominic Pattison, Harrison Zheng, Teo Tomic, Peter Henry, Gareth Cross, Josiah VanderMey, Alvin Sun, Samuel Wang, and Kristen Holtz. "SymForce: Symbolic Computation and Code Generation for Robotics". In: *Proceedings of Robotics: Science and Systems*. 2022. DOI: 10.15607/RSS.2022.XVIII.041.

Appendix B

EXTENSION OF CHAPTER VI: ON-ORBIT DEMONSTRATION

Note: This chapter is based on unpublished work.

B.1 Chapter Summary

Flight Hardware

The Edge Node spacecraft, which is a successor of the Edge Node Lite spacecraft flown in space, is presented in Fig. B.1.

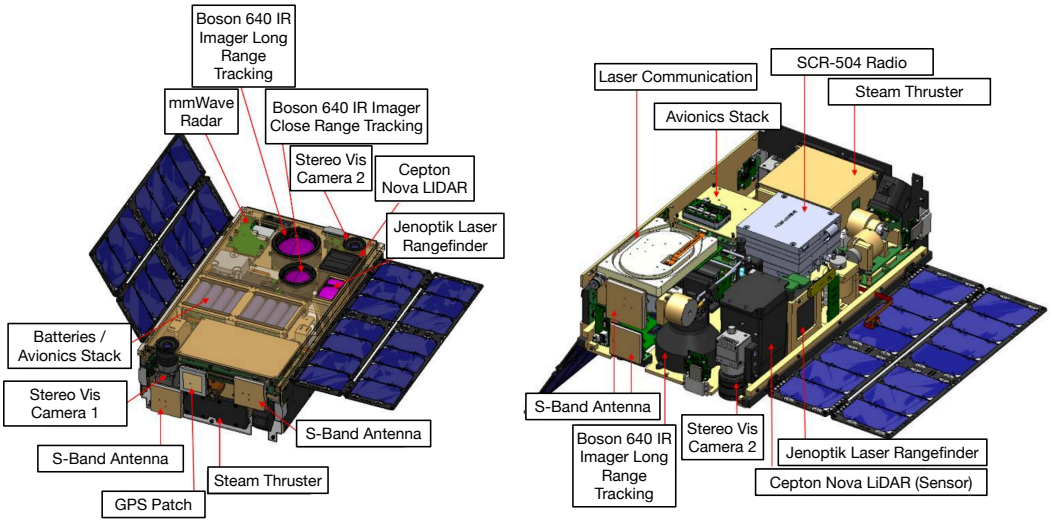


Figure B.1: Edge Node spacecraft [1]

Implementation on-board Flight Hardware

Our experiment is wrapped into a Docker compose, which is a tool that simplifies the process of defining and managing multi-container Docker applications. The Docker file is responsible for the following tasks: (1) Starting the ROS2 nodes. (2) Starting the ROS2 bags. (3) Stopping the ROS2 nodes and bags. The flight computer, a 32-bit ARM Cortex-M7, is in change of scheduling the commands to start and stop the Docker containers.

Power Modes. Each Jetson TX2 module offers several predefined power modes for specific power budgets. These modes also adjust the frequencies of the GPU and CPU. Initially, our experiments led to a brownout on the Jetson TX2 due to its power consumption exceeding the spacecraft's allocated power budget. We resolved this

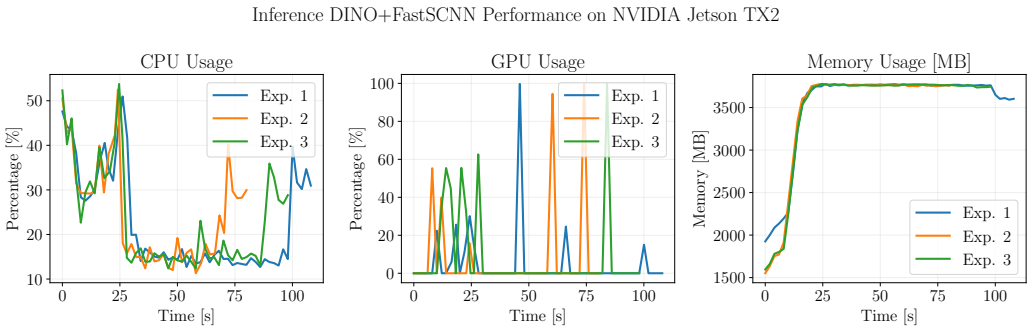


Figure B.2: Performance of the NVIDIA Jetson TX2 computer during inference of the Dino+FastSCNN algorithm.

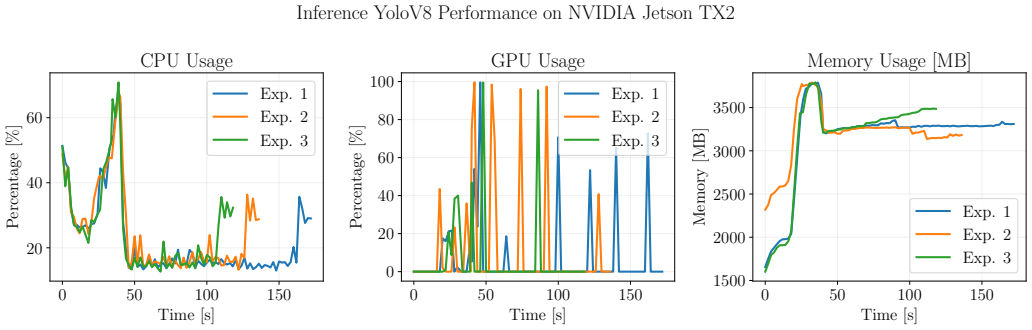


Figure B.3: Performance of the NVIDIA Jetson TX2 computer during inference of the YoloV8 algorithm.

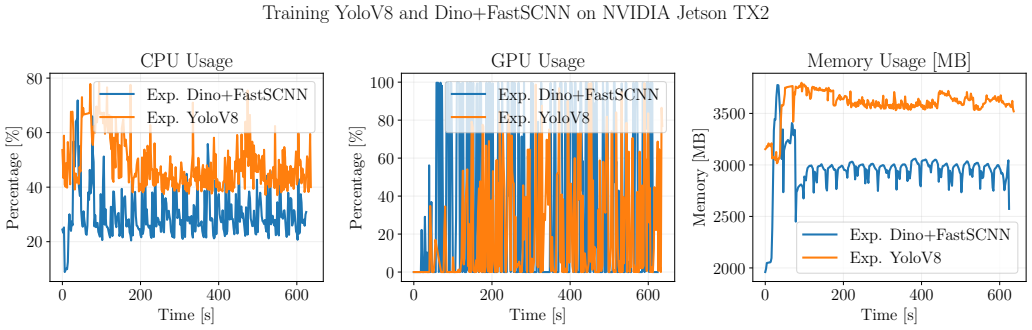


Figure B.4: Performance of the NVIDIA Jetson TX2 computer during training of the YoloV8 and Dino+FastSCNN algorithms.

issue by setting the Jetson TX2 to Power Mode 2 (MAX-P).

Language	Files	%	Code	%	Comment	%
Python	14	73.7	543	61.8	161	18.3
YAML	1	5.3	14	100.0	0	0.0
INI	1	5.3	4	100.0	0	0.0
__empty__	1	5.3	0	0.0	0	0.0
__duplicate__	1	5.3	0	0.0	0	0.0
__binary__	1	5.3	0	0.0	0	0.0
Sum	19	100.0	561	62.5	161	17.9

Language	Files	%	Code	%	Comment	%
Python	18	90.0	1930	64.2	548	18.2
Markdown	1	5.0	8	66.7	0	0.0
__empty__	1	5.0	0	0.0	0	0.0
Sum	20	100.0	1938	64.3	548	18.2

Language	Files	%	Code	%	Comment	%
Python	121	72.0	15924	60.7	6195	23.6
YAML	34	20.2	3937	87.7	282	6.3
Markdown	3	1.8	92	63.4	2	1.4
Bash	4	2.4	82	56.2	45	30.8
__empty__	1	0.6	0	0.0	0	0.0
__duplicate__	3	1.8	0	0.0	0	0.0
__binary__	2	1.2	0	0.0	0	0.0
Sum	168	100.0	20035	64.6	6524	21.0

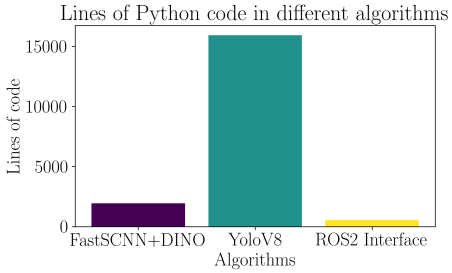


Figure B.5: Number of lines of code for the YoloV8, Dino+FastSCNN, and ROS2 code bases.

Table B.1: Different modes for the NVIDIA Jetson TX2. In orange, we outline the mode used for computer on-board the Edge Node Lite spacecraft.

Property	**MAX-N (Mode 0)**	**MAX-Q (Mode 1)**	**MAX-P (Mode 2)**	**MAX-P* (Mode 3)**	**MAX-P (Mode 4)**
Power Budget	N/A	7.5 W	15 W	15 W	15 W
Online A57 CPU	4	4	4	4	1
Online D15 CPU	2	0	2	0	1
A57 CPU max freq (MHz)	2000	1200	1400	2000	345
D15 CPU max freq (MHz)	2000	N/A	1400	N/A	2000
GPU max freq (MHz)	1300	850	1122	1122	1122
Memory max freq (MHz)	1866	1331	1600	1600	1600

BIBLIOGRAPHY

[1] Darren Rowen et al. “Edge Node: A Multi-User Rendezvous and Proximity Operations On-orbit Testbed”. In: *Small Satellite Conf.* (Aug. 2024).

Appendix C

EXTENSION OF CHAPTER VII: LEARNING-BASED CONTROL FOR BIPEDAL LOCOMOTION

C.1 Chapter Overview

Note: This chapter is based on unpublished work.

In this chapter, we present preliminary work on learning-based control algorithms that incorporate vision information for bipedal locomotion. The experiments and methods are developed using the bipedal robot introduced in Chapter 8. We recommend that readers first review that chapter to familiarize themselves with the robot model, control architecture, and notation used throughout this appendix.

C.2 Motivation

Model-Free Reinforcement Learning Algorithms

Successfully applying Proximal Policy Optimization (PPO) to bipedal robots typically requires two components:

- Reward shaping, often involving hand-crafted reward functions or references to precomputed gait libraries.
- Domain randomization, to address the sim-to-real gap by exposing the policy to varied dynamics during training.

However, even when these components are in place, existing approaches exhibit key limitations. Policies trained in simulation are frozen at deployment time, meaning the bipedal robot does not adapt or learn new behaviors on the fly. Recent work shows that vanilla RL algorithms such as PPO and DDPG tend to lose adaptability in continual learning settings [1].

To illustrate the sensitivity of RL algorithms to reward shaping, we consider a simple case study: the swing-up control of an inverted pendulum using four popular RL algorithms (PPO [2], TD3 [3], SAC [4], and DDPG [5]). We compare their performance under two types of reward signals, a sparse one and a dense one:

$$\begin{aligned} c_1 &= 1.0\theta^2 + 0.1\dot{\theta}^2, \\ c_2 &= 10.0\tanh(10\theta^2) + 0.1\dot{\theta}^2, \end{aligned} \tag{C.1}$$

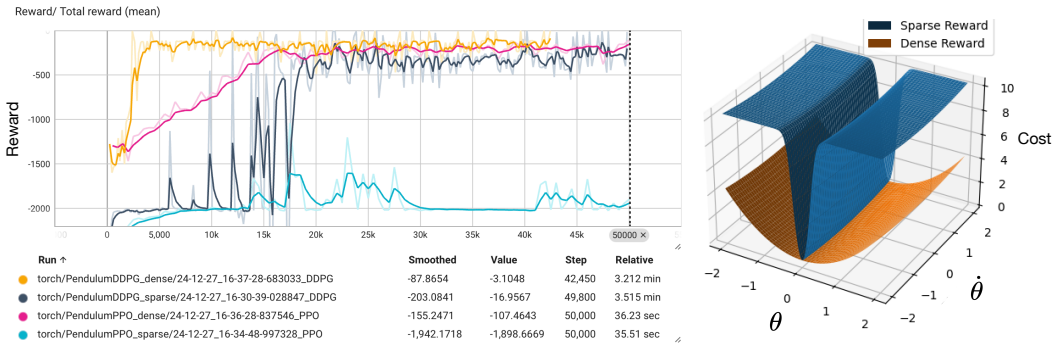


Figure C.1: Performance of learning algorithms under sparse and dense rewards and plot of the rewards from (C.1).

where θ is the angle of the pendulum and $\dot{\theta}$ is the angular velocity. In Fig. C.1A, we show the performance of the algorithms with the two reward functions and in Fig. C.1B, we plot the rewards from (C.1). We observe that the dense reward enables the learning algorithm to learn to swing up the pendulum while the sparse reward does not provide enough feedback for the PPO algorithm to converge. Interestingly, the DDPG algorithm performs better than PPO with the sparse reward. The code for this experiment is available at https://github.com/lupusorina/reward_analysis_ddpg_vs_ppo.

Vision-based Learning

A key ingredient for enabling more adaptive and intelligent locomotion is vision. Unlike feedback from proprioception alone, vision provides rich information about the terrain. This is especially important for bipedal robots, where stability and foot placement are tightly coupled with the structure of the environment. For example, detecting obstacles, slopes, or uneven surfaces in advance allows the controller to make informed decisions about where and how to step. Incorporating vision information into the control pipeline enables the robot not only to react to the current state but also to plan, a capability essential for traversing complex real-world environments. As such, our work investigates how visual inputs can be effectively leveraged in both learning-based and model-based control strategies.

C.3 Methods: Residual Foot Stepping for Bipedal Locomotion with Terrain Information

Context: We propose a method for learning residual footsteps for bipedal locomotion that leverages terrain information to improve the walking capability. This approach utilizes a learning-based controller with vision information to adjust the

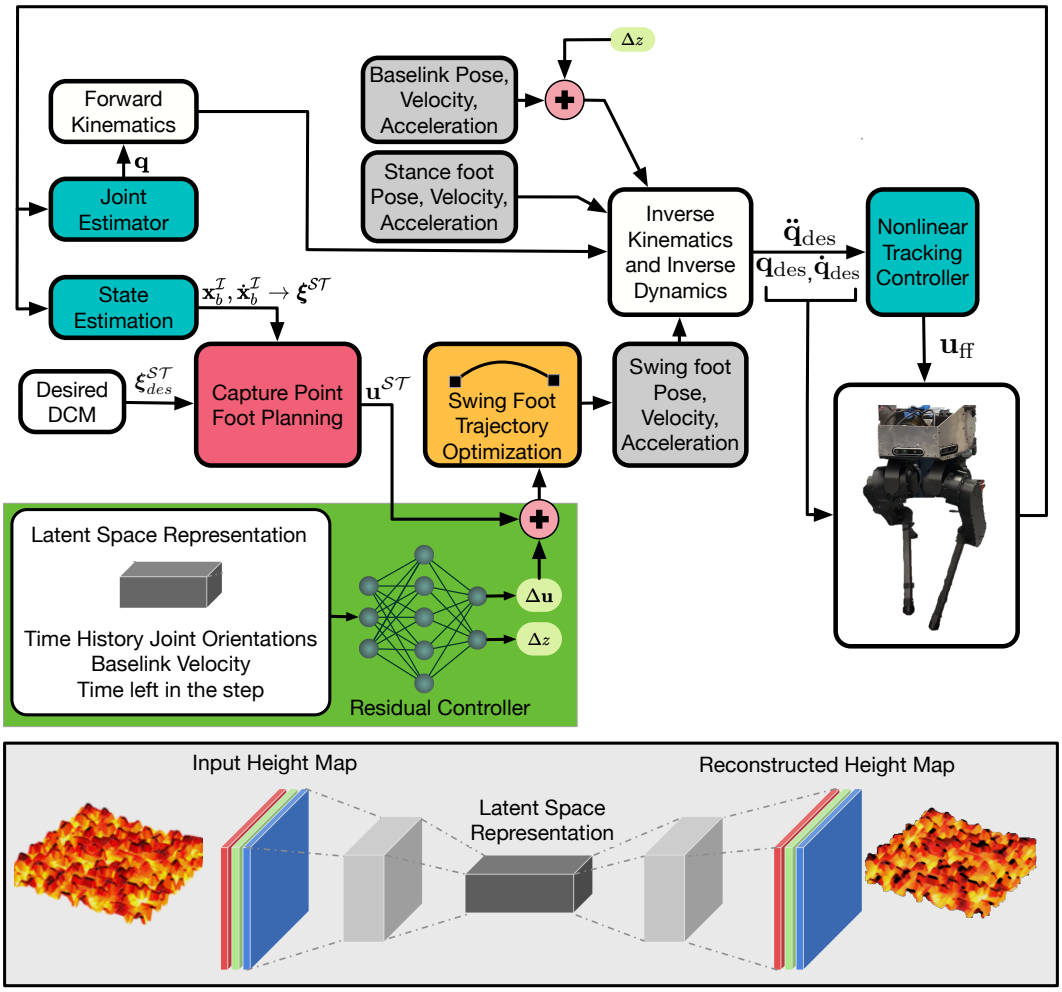


Figure C.2: Top: Learning strategy for residual footsteps and height. Middle: Learning architecture in green, added on top of the existing control architecture in Fig. 8.2. Bottom: Autoencoder Network used for learning a lower dimensional representation of the terrain.

footsteps of the bipedal robot in response to varying terrain conditions. Additionally, this research serves as foundational work for the broader MAGIC-VFM algorithm presented in Chapter 2. Insights gained from this study have significantly informed the development of the MAGIC-VFM framework.

Overview

Our method is inspired by the adaptability of human locomotion across diverse terrains. On flat ground, humans generally maintain a periodic gait characterized by consistent step length and frequency. In contrast, when encountering rough or uneven terrain, they dynamically adjust their stepping patterns based on sensory feedback to preserve balance and ensure stability. Additionally, humans often lower

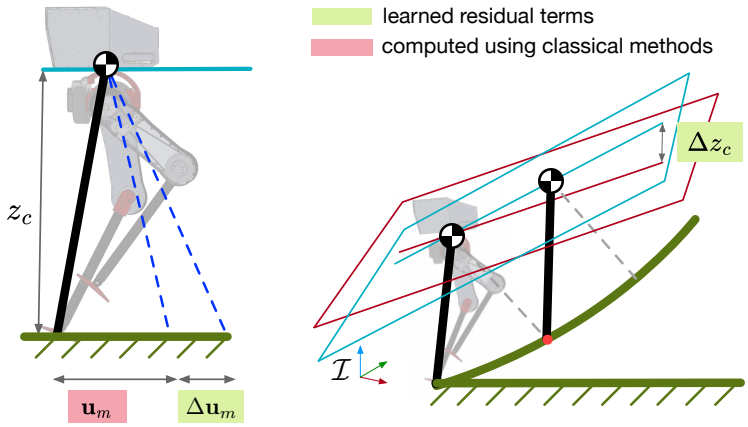


Figure C.3: Illustration of the vision-based control framework for bipedal locomotion. The center of mass height and the foot placement are initially computed using model-based methods (red). Learned residual terms (green) are then added to refine these estimates, resulting in improved foot placement and corrected center of mass trajectory. The right panel visualizes this correction over slopes.

their center of mass to increase reachability and enhance stability in challenging environments.

Method

The learning architecture, illustrated in Fig. 8.2, extends the model-based control and planning framework for walking. A more in-depth description of the control architecture where both a model-based and a learning-based framework are employed is presented in Chapter 8. The learning-based method does not include vision information. The learning component uses policy gradient to learn both a residual footstep $\Delta\mathbf{u}$ and a residual height adjustment $\Delta\mathbf{z}_c$. Using the residual footstep, we adjust (8.6) as follows

$$\begin{aligned}\mathbf{u}'_m &= \mathbf{u}_m + \Delta\mathbf{u}, \\ &= -\boldsymbol{\xi}_{\mathrm{d},m+1} + \boldsymbol{\xi}_m e^{\omega T_s} + \Delta\mathbf{u},\end{aligned} \tag{C.2}$$

where T_s is the step time, $\boldsymbol{\xi}_m$ is the divergence component of motion (DCM), and $\boldsymbol{\xi}_\mathrm{d}$ is the desired DCM, with m counting steps. This is shown in Fig. C.3A. Next, we intuitively explain why we need to learn a residual height adjustment $\Delta\mathbf{z}_c$ in addition to the residual footstep. When the robot needs to extend its footstep further forward than the nominal position, lowering the center of mass may be required. This adjustment enables the robot to achieve a longer stride while maintaining balance and stability.

Network and Learning Strategy

We use the Proximal Policy Optimization (PPO) to optimize the residual policy network. For a description of the PPO algorithm, see [2]. The policy and value function networks architecture are two multi layer perceptron. The policy takes as input a large tensor containing the following

- Remaining time in the footstep cycle T_s.
- Planar base link velocity in the body frame $\mathbf{v}^{\mathcal{B}}$.
- Positions of the links in the body frame.
- Latent vector from the terrain autoencoder network (see Fig. C.3B and Appendix C.3).

and outputs the residual footstep vector $\Delta\mathbf{u}$ and height adjustment Δz_c.

The reward for PPO is defined as follows

$$r = 1 - \omega_1 e_{\text{dcm,y}} - \omega_2 \|\mathbf{u}\|^2 - c_{\text{falling}} + \omega_3 (1 - \|\mathbf{p}^{\mathcal{I}} - \mathbf{p}^{\mathcal{I}}_{\text{goal}}\|^2),$$

where ω_1, ω_2 and ω_3 are weights, $e_{\text{dcm,y}}$ is the error in the DCM in the y direction, $\mathbf{u}$ is the joints control input, c_{falling} is a penalty for falling, $\mathbf{p}^{\mathcal{I}}_{\text{goal}}$ is the goal position in the inertial frame, and $\mathbf{p}^{\mathcal{I}}$ is the position of the base link in the inertial frame, as well. The penalty for falling is applied when the roll or pitch angles exceed a certain threshold.

Terrain Learning

The robot builds a robot-centric terrain grid map around its center of mass. Let $\mathcal{G}$ be this grid map of size $[W, H]$, where W and H are the width and height of the robot-centric map, respectively. Each cell $g_{i,j}$ contains the elevation $e_{i,j}$. This 3D tensor should not be directly fed into the policy network because of the input dimensionality and variability. Therefore, it is often recommended to use a lower representation of the main features of the terrain. Therefore, we choose to design an autoencoder network to learn a lower-dimensional representation (latent vector) of the terrain, as seen in Fig. C.3. The network consists of two primary components: an encoder and a decoder. The encoder contains 2 convolutional networks with ReLU as activation functions and max pooling layers. The decoder reconstructs the original input from this compressed encoding using deconvolutions, which increase the spatial resolution of the tensor. The layers include 2 convolutional layers with ReLU activation functions.

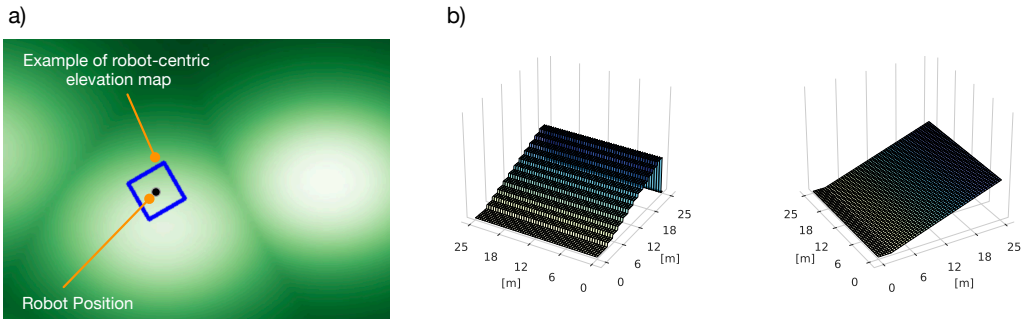


Figure C.4: Terrain representation (a) Heatmap showing a simulated terrain with hills. The blue square is the robot-centric elevation map. (b) Examples of other types of terrains generated: uneven stairs and slopes.

C.4 Results: Residual Foot Stepping for Bipedal Locomotion with Terrain Information

Implementation Details

The architecture is implemented in simulation using MuJoCo [6]. The learning component uses the Stable Baselines 3 framework with allows the interface between a Gym environment in MuJoCo and the PPO reinforcement learning algorithm. The terrain is generated in MuJoCo as a heightfield map and a patch around the robot is extracted, as seen in heat map from Fig. C.4a where a simulated terrain with hills is shown. We have also generated other types of terrains such as uneven stairs and slopes of different inclinations. To train the autoencoder presented in Fig. C.3, we extract a dataset of patches from these terrain types: hills, slopes, and uneven stairs.

Ablation Study: Learning Experiments without Terrain Information

First, we evaluate the performance of our proposed algorithm (Appendix C.3) without terrain information, in which the robot is walking blindly on a slope of certain inclination. The results of this experiment can be seen in Fig. C.5. In the forward direction, Δu is larger than in the sideways direction. This is caused by the fact that the center of mass offset in the forward direction is larger and harder to model apriori than in the sideways stepping. The y-offset for the center of mass is not large as the robot is symmetric right left. Despite a small adjustment in $\Delta \mathbf{u}$, the impact on the DCM correct is significant, as seen in Fig.C.5.

Ablation Study: Learning Experiments with Terrain Information

We train our RL algorithm with the latent vector from the autoencoder and compare the 3 cases: no learning (controller presented in Chapter 8), learning with no terrain information (presented in Appendix C.4), and learning with terrain information.

Table C.1: Performance of the bipedal robot with and without learning on inclined terrain. The experiment was repeated 10 times.

	No learning	RL (no terrain input)	RL (with terrain input)
Mean number of falls for fixed duration	4.79	0.72	0.27

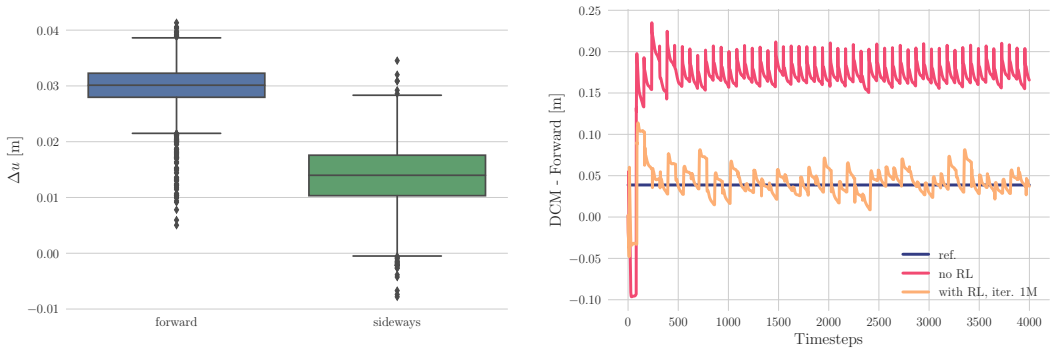


Figure C.5: (Left) $\Delta\mathbf{u}$ distribution (Right) Forward DCM for the two cases: no learning, learning, but no terrain information.

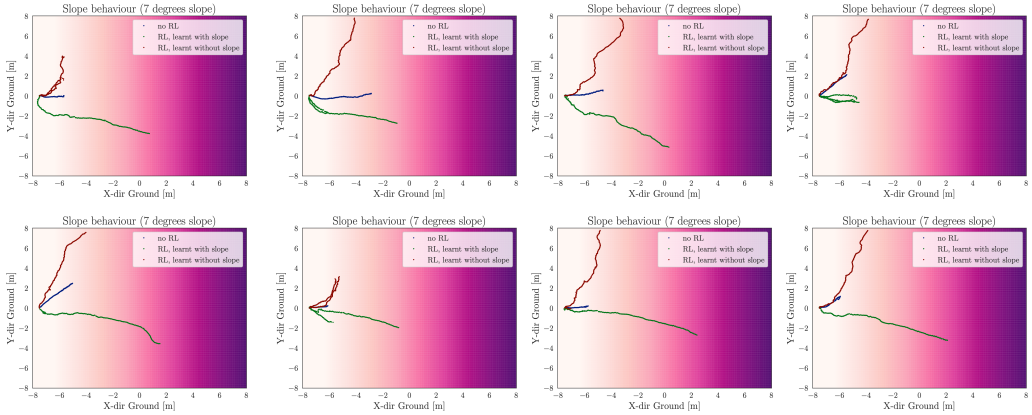


Figure C.6: Performance of the 3 controllers: blue: no learning, green RL (no terrain information) and red: RL with terrain information, on an inclined surface for 8 trials.

We train the robot on a variety of different grounds, such as slopes, hills and stairs and evaluate its performance on slopes of 7 degrees inclination. We perform both a qualitative analysis and a quantitative analysis. For the quantitative analysis, we let the robot walk on an inclined surface for a fixed duration of 10 minutes and 11 trials, and record how many times it fell in each trial. After each fall, the robot restarts its walk from the bottom of the slope. In Table C.1, we compute the mean

number of falls for this fixed duration across the trials. When no learning is used, the robot falls considerably more often than when learning is used. This can be seen qualitatively in Fig. C.6, where we show the trajectories for 8 of the 11 trials as the robot is going up the slope. The blue trajectory (no learning)is considerably shorter than the other 2 trajectories which use learning. When learning is employed, but with no terrain information, the robot already greatly improves its resilience to falls, but exploits a certain walking strategy of going side-ways up the slope. This is common in learning-based algorithms like PPO, where the agent often exploits certain behaviors to maximize its reward. Lastly, the best performance is achieved when terrain information is used, leading to only 0.28 falls over the 11 trials, which translates to 3 falls over the 11 trials.

Limitations

This strategy has several limitations, as follows

1. Training only on elevation information is not representative enough, as the friction coefficient of the terrains plays a large role in whether the robot will fall or not.
2. Training the robot in simulation and deploying it on hardware, even with heavy domain-randomization, is not a robust solution, as the elevation map that is created on hardware is difficult to simulate.
3. There are infinite number of terrains and terrain structures so creating a comprehensive dataset is difficult.
4. The learning algorithm does not adapt in real time: once the PPO is trained, it is fixed at deploy time.
5. Learning just a $\Delta\mathbf{u}$ and Δz is not representative enough, as often times, if there is an obstacle, the robot should learn to lift its foot more.

Based on the aforementioned issues, we developed a more comprehensive solution, as shown in Chapter 2-3 where we use Visual Foundation Models to encode terrain information, which is more general. In addition, we use composite adaptation to adapt the pre-trained neural network of terrain residual at run time. In this way, we adjust to data not captured at training time, as well as other disturbances that can appear on the robot. This framework was demonstrated on-board two robots: a tracked vehicle and a vehicle with Ackermann steering.

BIBLIOGRAPHY

[1] Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Ruapm Mahmood, and Richard S. Sutton. "Loss of Plasticity in Deep Continual Learning". In: *Nature* 632 (2024), pp. 768–774.

[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[3] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *CoRR* abs/1802.09477 (2018). arXiv: 1802.09477. URL: http://arxiv.org/abs/1802.09477.

[4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

[5] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[6] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A Physics Engine for Model-based Control". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst.* Oct. 2012, pp. 5026–5033.

Appendix D

UNCERTAINTY QUANTIFICATION FOR LEARNING-BASED PLANNING AND CONTROL WITH MODEL LEARNING

D.1 Chapter Overview

Note: This chapter is based on unpublished work and the results are preliminary.

In this chapter, we start by giving an overview of different uncertainty quantification (UQ) methods for dynamics learning, with a focus on Bayesian Neural Networks as a method for UQ. We then present a model-based reinforcement learning framework that uses Bayesian Neural Networks to quantify uncertainty in the dynamics of a system and use it in planning.

D.2 Introduction

Reliable model learning is important for model-based planning and control, particularly in safety-critical applications such as autonomous driving, where inaccuracies in the learned dynamics can lead to catastrophic failures [1]. Models are typically learned using either parametric approaches, like deep neural networks (DNNs), or non-parametric methods, such as kernel-based techniques. For instance, several recent works [2, 3] employ neural networks to capture the residual dynamics of complex systems. However, the performance of these models is highly dependent on the quality and diversity of the training data. In practice, models are often deployed in conditions that deviate from the training distribution, leading to poor generalization and unreliable predictions. To address this, planning algorithms must incorporate mechanisms for quantifying and accounting for model uncertainty during decision-making, thereby improving robustness in out-of-distribution scenarios.

We exemplify this limitation in a model prediction in Fig. D.1, where we show an example of a regression problem using a Multi-Layer Perceptron (MLP) with data generated from the true function with noise $y = f(x) = x^2 + \cos(5x)$, where $x \in \mathbb{R}$. The network is not able to capture the symmetry of the function, and thus predicts wrongly in the "unseen" region. Planning with such a model can lead to unwanted behaviors.

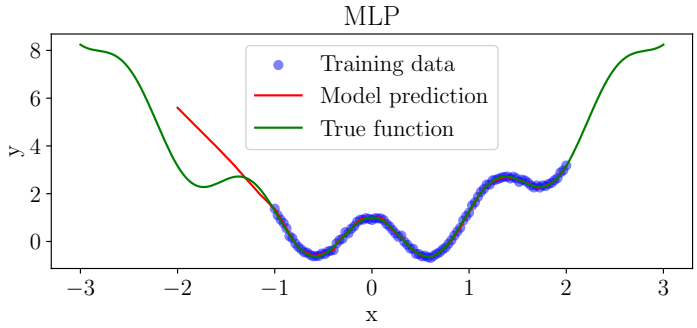


Figure D.1: Performance of a multilayer perceptron (MLP) on a nonlinear regression task. The blue dots represent noisy training data sampled from the true function (green). The red curve shows the MLP's prediction, which closely fits the training data in the observed region but fails to generalize in extrapolation regions, highlighting a common limitation of standard neural networks in out-of-distribution scenarios.

D.3 Related Work

Methods for Uncertainty Quantification in Dynamics Learning for Planning and Control

Model-based methods are more data-efficient than model-free RL methods like DDPG [4] or PPO [5], as they can leverage the model to simulate the environment and generate data for training the policy. However, model-based methods suffer from model bias (the error that arises when the learned model does not accurately reflect the true dynamics of the environment). This bias can compound over multiple prediction steps, leading to poor planning decisions and ultimately suboptimal policies that fail in deployment [6] To address this issue, several methods have been proposed to quantify the uncertainty in the model predictions and take this uncertainty into account when planning. For example, in [7, 8], the dynamics residual is modeled as a multivariate Gaussian distribution, which is further used in a chance-constrained planning framework.

Gaussian Processes in Model-based Reinforcement Learning (MBRL)

A Gaussian Process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution [9]. The process is specified by its mean function and covariance function (also sometimes referred as kernel). The generalization property of a GP is controlled by the choice of the kernel function. For example, in [10], a DNN is used to transform the inputs into the GP kernel.

Several works propose using GPs in a model-based RL framework by learning a probabilistic dynamics model and explicitly incorporating model uncertainty into planning. For instance, in the PILCO framework, a probabilistic dynamics model is constructed using GPs [6]. Due to the properties of GPs, analytical solutions are integrated into the policy improvement by computing analytical gradients.

Deep Ensembles

The idea behind deep ensembles is to train several distinct models (for example DNNs initialized with different weights) on the same data and then combine their outputs. The mean and variance in the predictions of the ensemble items are then used to quantify the uncertainty in the model predictions. Several methods use Deep Ensemble in a MBRL framework. For example, Probabilistic Ensembles with Trajectory Sampling (PETS) combines ensemble-based deep network dynamics models with sampling-based control [11].

D.4 Methods

System Model

Consider a system with state $\mathbf{x} \in \mathbb{R}^n$ and input $\mathbf{u} \in \mathbb{R}^m$ that evolves according to the following dynamics

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t), \tag{D.1}$$

where $\mathbf{f} : \mathbb{R}^{n+m+1} \to \mathbb{R}^n$ is the true dynamics. For generality, let $\hat{\mathbf{f}}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{u})$ be the learned dynamics model, parameterized by some weights $\boldsymbol{\theta}$, but the same framework can be applied to residual dynamics learning (i.e., known model derived from physics and a learned residual dynamics model). In this case, $\boldsymbol{\theta}$ are the weights of a neural network that approximates the true dynamics with a model that contains a form of uncertainty quantification. The goal is to design a controller that uses this uncertainty in the model predictions to optimize a cost function. We use the following form of the controller for the system in (D.1)

$$\mathbf{u} = \mathbf{u}^{\mathrm{ff}} + \mathbf{K}(\mathbf{x} - \mathbf{x}^{\mathrm{d}}), \tag{D.2}$$

where $\mathbf{u}^{\mathrm{ff}}$ is the feedforward term, $\mathbf{K}$ is the feedback gain matrix, and $\mathbf{x}^{\mathrm{d}}$ is the desired state. This form of the controller is standard in classical control theory [12].

Algorithm Description

In Algorithm 8, we propose a model-based learning framework that uses Bayesian Neural Networks (BNNs) to quantify uncertainty in the dynamics of a system and use

Algorithm 8 Model-based Reinforcement Learning with Uncertainty Quantification using Bayesian Neural Networks (BNNs)

1: **Input**
2: $\mathcal{D}$ - empty dataset of trajectories.
3: $\hat{\mathbf{f}}_{\boldsymbol{\theta}}^{\text{BNN}}$, with $\boldsymbol{\theta_0} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ - initial model of the world.
4: N - number of rollouts.
5: T - rollout length.
6: Trajectory of random actions $\mathbf{u}_{[0:T]}^{\text{ff}}$ and a gain matrix $\mathbf{K} = \mathbf{0}$.
7: **Output:** Optimized trajectory $\mathbf{u}_{[0:T]}^{\text{ff}}$, optimized $\mathbf{K}$
8: **while** not done **do**
9: Run the controller on the system

$$\mathbf{u}_{[0:T]} = \mathbf{u}_{[0:T]}^{\text{ff}} + \mathbf{K}(\mathbf{x}_{[0:T]} - \mathbf{x}_{[0:T],\text{nom}}).$$

10: Collect data $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})\}_{t=0}^{T-1}$.
11: Retrain the model $\hat{\mathbf{f}}_{\boldsymbol{\theta}}^{\text{BNN}}$ on the new dataset $\mathcal{D}$.
12: Optimize a new trajectory

$$\begin{aligned} \min_{\mathbf{u}_{[0:T]}^{\text{ff}}, \mathbf{K}} \quad & \sum_{i=0}^{N} \left\| \mathbf{x}_T^i - \mathbf{x}_d \right\|_2^2 + \sum_{i=0}^{N} \sum_{k=0}^{T} \left\| \mathbf{u}_k^i \right\|_2^2 \\ \text{subject to } \mathbf{x}_{k+1,\,\text{nom}} &= \mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})} \left[\hat{\mathbf{f}}_{\boldsymbol{\theta}}^{\text{BNN},i} \left(\mathbf{x}_{k,\,\text{nom}}, \mathbf{u}_k^{\text{ff}} \right) \right], \text{ for } k = \overline{0, T} \\ \mathbf{x}_{k+1}^i &= \hat{\mathbf{f}}_{\text{BNN}}^i \left(\mathbf{x}_k^i, \mathbf{u}_k \right), \text{ for } k = \overline{0, T}, \text{ for } i = \overline{0, N} \end{aligned} \tag{D.3}$$

13: **end while**

this uncertainty in planning. Note that we employ BNNs to model the dynamics of the system, but the same framework can be applied to other methods for uncertainty quantification in dynamics learning, like the ones mentioned in Appendix D.3.

Our algorithm takes as input an empty dataset $\mathcal{D}$ of trajectories and initial model of the world $\hat{\mathbf{f}}_{\boldsymbol{\theta}}^{\text{BNN}}$, where the weights $\boldsymbol{\theta}$ are sampled from a Gaussian distribution, as presented in Lines 2-3. We specify the problem in discrete time, where T is the rollout length with k denoting the time step. N is the number of rollouts (i.e, the number of sampled trajectories from the system). The algorithm outputs a trajectory of feedforward actions $\mathbf{u}_{[0:T]}^{\text{ff}}$ and an optimized feedback gain matrix $\mathbf{K}$, which are used to control the system using (D.2). Here, the initial trajectory is initialized at random, but in practice, it can be a safe trajectory. Next, we run the controller in Line 9 on the real system, where both the feedforward trajectory and the gain matrix $\mathbf{K}$ are the optimized ones from Line 12. We then add the collected data to the dataset $\mathcal{D}$ and retrain the BNN model on this dataset. Lastly, we solve

the optimization problem using Stochastic Gradient Descent (SGD) in Line 12 by optimizing a quadratic cost subject to dynamics. The nominal trajectory $\mathbf{x}_{[0:T],\text{nom}}$ is computed as an expectation over the BNN weights sampled from the posterior distribution.

The intuition of the algorithm is as follows: because the dynamics has uncertainty, sampling different weights for the BNN will result in different predictions for the same input, which can be used to quantify the uncertainty in the model predictions. Note that the uncertainty is higher in regions of the state space that were not explored yet, as the model was not trained on data from these regions. This fact helps the planner to explore the state space more efficiently, as it can focus on the regions where the model is uncertain.

D.5 Results

We validate Algorithm 8 on a simple model: a swing-up pendulum task with the dynamics

$$\ddot{\alpha} = -\frac{g}{l}\sin(\alpha), \tag{D.4}$$

where α is the pendulum angle, measured from the vertical downward axis (i.e., $\alpha = 0$ when the pendulum is down), g is the gravitational acceleration, l is the length of the rod. The states of the system are α and $\dot{\alpha} := \omega$, and the control input is the torque applied to the pendulum u. We approximate the dynamics in (D.4) using a BNN. The input into the BNN is the state in discrete form: α_k, ω_k and the control input u_k at time step k, and the output is the state difference as follows: $\frac{1}{\Delta t}\alpha_{k+1} - \alpha_k, \frac{1}{\Delta t}\omega_{k+1} - \omega_k$, where Δt is the time step. The objective is $\alpha_T = \pi$ and $\omega_T = 0$. During the implementation, we use the trigonometric functions cos and sin to model the dynamics in (D.4) to avoid the angle discontinuities. We run Algorithm 8 for several epochs and record the performance of the angle and angular velocity during 3 intermediary epochs, as shown Appendix D.5. We note that during the earlier epochs, the uncertainty in the model prediction is higher, as the model was not trained on data from these regions. However, this exploration helps the planner to find a better trajectory that optimizes the cost function even after very few episodes.

Future steps

There are several directions for future work to complete this study, as follows. First, Algorithm 8 should be benchmarked against other model-based planning methods that use uncertainty quantification in the dynamics learning, such as PILCO [6],

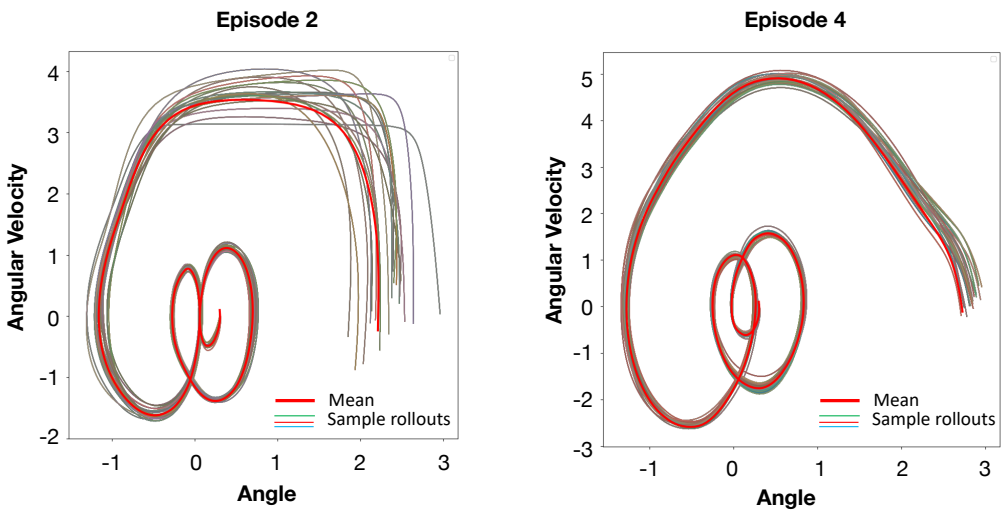


Figure D.2: Trajectories during training in episode 2 and episode 4.

which uses Gaussian Processes. In addition, a study on the computational complexity of the proposed method should be conducted, as Bayesian Neural Networks are known to be computationally expensive due to the sampling process during inference and training. Moreover, an analysis should be conducted to assess the best UQ method for the proposed planning framework, as the choice of the UQ method can significantly impact the performance of the method. Finally, the proposed method should be tested on more complex systems, especially on hardware, to evaluate its performance in real-world scenarios. As extensions, the algorithm can be extended to use a latent space representation rather than the full state $\mathbf{x}$. If a latent space is used, a controller of the form presented in (D.2) might not be appropriate.

D.6 Conclusions

We propose a model-based planning framework with dynamics learning, where the dynamics is learned using BNNs to quantify the uncertainty in the model predictions. We show that the proposed method can be used to optimize a trajectory in a swing-up pendulum task, where the uncertainty in the model predictions is used to inform the planner about the regions where the model is uncertain.

BIBLIOGRAPHY

[1] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. "A Survey of Autonomous Driving: Common Practices and Emerging Technologies". In: *IEEE Access* 8 (2020), pp. 58443–58469.

[2] Elena-Sorina Lupu*, Fengze Shi*, James Preiss, Jedidiah Alindogan, Matthew Anderson, and Soon-Jo Chung. "MAGIC-VFM: Meta-learning Adaptation for Ground Interaction Control with Visual Foundation Models". In: *IEEE Transactions on Robotics* (2024).

[3] Michael O'Connell, Guanya Shi, Xichen Shi, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung. "Neural-Fly Enables Rapid Learning for Agile Flight in Strong Winds". In: *Science Robotics* 7.66 (2022), eabm6597.

[4] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[6] Marc Peter Deisenroth and Carl Edward Rasmussen. "PILCO: a model-based and data-efficient approach to policy search". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 465–472.

[7] Yashwanth Kumar Nakka, Anqi Liu, Guanya Shi, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung. *Chance-Constrained Trajectory Optimization for Safe Exploration and Learning of Nonlinear Systems*. 2020. arXiv: 2005.04374 [cs.RO].

[8] Anqi Liu, Guanya Shi, Soon-Jo Chung, Anima Anandkumar, and Yisong Yue. *Robust Regression for Safe Exploration in Control*. 2020. arXiv: 1906.05819 [cs.LG].

[9] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[10] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. *Deep Kernel Learning*. 2015. arXiv: 1511.02222 [cs.LG].

[11] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. *Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles*. 2017. arXiv: 1612.01474 [stat.ML].

[12] Karl Johan Astrom and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. USA: Princeton University Press, 2008.

Appendix E

BACKGROUND

E.1 Chapter Overview

In this section, we briefly revisit several control theory and robotics kinematics concepts that are shared across the chapters and appendices. For each subsection, we outline several nuances and details that were used to the development of the algorithms and methods presented in this thesis. It assumes that the reader has a basic understanding of key concepts in linear algebra, analysis, as well as linear control theory and basic Lyapunov theory.

E.2 Advanced Stability Theory

We present several concepts in advanced stability theory that were used in proofs from Chapter 3. This section is based on [1].

Comparison Functions

Autonomous systems are systems whose dynamics are independent of time explicitly. Non-autonomous systems, in contrast, have dynamics that explicitly depend on time. Proving stability for non-autonomous systems uses special functions, such as class $\mathcal{K}$ functions, class $\mathcal{KL}$ functions and class $\mathcal{K}_\infty$ functions defined as follows.

Definition 1 *(from Khalil, Chapter 4 [1]) A continuous function* $\alpha : [0, a) \rightarrow [0, \infty)$ *is said to belong to class* $\mathcal{K}$ *if it is strictly increasing and* $\alpha(0) = 0$. *It is said to belong to class* $\mathcal{K}_\infty$ *if* $a = \infty$ *and* $\alpha(r) \rightarrow \infty$ *as* $r \rightarrow \infty$.

Example: $\alpha(r) = tan^{-1}(r)$ is a class $\mathcal{K}$ function. $\alpha(r) = r^2$ is a class $\mathcal{K}_\infty$ function.

Definition 2 *(from Khalil, Chapter 4 [1]) A continuous function* $\beta : [0, a) \times [0, \infty) \rightarrow [0, \infty)$ *is said to belong to class* $\mathcal{KL}$ *if, for each fixed s, the mapping* $\beta(r, s)$ *belongs to class* $\mathcal{K}$ *with respect to r and, for each fixed r, the mapping* $\beta(r, s)$ *is decreasing with respect to s and* $\beta(r, s) \rightarrow 0$ *as* $s \rightarrow \infty$.

Example: $\beta(r, s) = r^2 e^{-s}$ is a class $\mathcal{KL}$ function.

Next, we will show how these functions are used to prove stability Appendix E.2.

Input-to-State Stability (ISS)

ISS answers the question "What can we say about the behavior of a system in the presence of bounded inputs?" and provides a framework to study how robust a system is to external disturbances. We give the next definition of input-to-state stability.

Definition 3 *(from Khalil, Chapter 4 [1]) The system* $\dot{\mathbf{x}} = f(t, \mathbf{x}, \mathbf{u})$ *is said to be input-to-state (ISS) stable if there exist a class* $\mathcal{KL}$ *function* β *and a class* $\mathcal{K}$ *function* γ *such that for any initial state* $x(t_0)$ *and any bounded input* $u(t)$*, the solution* $x(t)$ *exists for all* $t \geq t_0$ *and satisfies*

$$\|\mathbf{x}(t)\| \leq \beta\left(\|\mathbf{x}(t_0)\|, t - t_0\right) + \gamma\left(\sup_{t_0 \leq \tau \leq t} \|\mathbf{u}(\tau)\|\right).$$

Note that the definition in Definition 3 is difficult to use in practice because it requires knowledge of the solution $\mathbf{x}(t)$, which is not always available. Therefore, there exists a *sufficient* Lyapunov-like theorem, which can be used (Theorem 4.19 in [1]), as follows

Theorem 5 *(from Khalil, Chapter 4 [1]) Let* $V : [0, \infty) \times R^n \to R$ *be a continuously differentiable function such that*

$$\alpha_1(\|\mathbf{x}\|) \leq V(t, \mathbf{x}) \leq \alpha_2(\|\mathbf{x}\|),$$
$$\frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}} f(t, \mathbf{x}, \mathbf{u}) \leq -W_3(\mathbf{x}), \quad \forall \|\mathbf{x}\| \geq \rho(\|\mathbf{u}\|) > 0,$$

$\forall (t, \mathbf{x}, \mathbf{u}) \in [0, \infty) \times \mathbb{R}^n \times \mathbb{R}^m$*, where* α_1, α_2 *are class* $\mathcal{K}_\infty$ *functions,* ρ *is a class* $\mathcal{K}$ *function, and* $W_3(\mathbf{x})$ *is a continuous positive definite function on* $\mathbb{R}^n$*. Then, the system* $\dot{\mathbf{x}} = f(t, \mathbf{x}, \mathbf{u})$ *is* ***input-to-state stable*** *with* $\gamma = \alpha_1^{-1} \circ \alpha_2 \circ \rho$*.*

Input-Output Stability. $\mathcal{L}$ stability.

We can model the nonlinear dynamics as an input-output system, and not in the state-space, like in Appendix E.2. The system is viewed as a black box with inputs and outputs. In this case, we define the $\mathcal{L}$ stability. We consider an input-output relationship as $\mathbf{y} = \mathbf{H}\mathbf{u}$, where $\mathbf{y}$ is the output, $\mathbf{u} : [0, \infty] \to \mathbb{R}^m$ is the input, and $\mathbf{H}$ is a mapping. For the space of piecewise continuous, bounded functions, the norm is defined as

$$\|\mathbf{u}\|_{\mathcal{L}_\infty} = \sup_{t \geq 0} \|\mathbf{u}(t)\| < \infty,$$

and the space is denoted by $\mathcal{L}_\infty^m$, where m is the dimension of the input. Similarly, we can define the more general $\mathcal{L}_p$ norm.

I/O Stability refers to the following question: "If $\mathbf{u} \in \mathcal{L}^m$ is a "well-behaved" input, what can we say about the output $\mathbf{y} \in \mathcal{L}^q$?" A system is said to be $\mathcal{L}$ stable if the output is bounded for bounded inputs. It is important to note that $\mathbf{H}$ cannot be defined as a mapping from $\mathcal{L}^m$ to $\mathcal{L}^p$ because we can have systems that are unstable. Therefore $\mathbf{H}$ is defined as a mapping from an extended space $\mathcal{L}_e^m$ to an extended space $\mathcal{L}_e^q$, where $\mathcal{L}_e^m$ is defined as

$$\mathcal{L}_e^m = \{\mathbf{u} \mid \mathbf{u}_\tau \in \mathcal{L}^m, \forall \tau \in [0, \infty)\},$$

and $\mathbf{u}_\tau$ is a truncation of u defined by

$$\mathbf{u}_\tau(t) = \begin{cases} \mathbf{u}(t), & 0 \leq t \leq \tau \\ 0, & t > \tau. \end{cases}$$

The definition of the stability of $\mathbf{H}$ is again using $\mathcal{K}$ special functions and can be seen in Definition 5.1 in [1]. The next question to ask if: Can we employ Lyapunov stability methods to establish the $\mathcal{L}$ stability of a nonlinear system, like in the case of ISS. The answer is yes, and the result is given in Theorem 5.1 in [1]. This theorem resembles Theorem 5, but with the addition of two more conditions for the dynamics and output of the system.

E.3 Adaptive Control

The content of this subsection is based on the following references [2, 3] and from Prof. Soon-Jo Chung's Data-driven Control (CDS245) lecture notes.

A model is a representation of a physical system that can be used for control design or estimation. Oftentimes, that model will have unknown or incorrectly modeled parameters of the real system. In addition, the model can have idealized assumptions and simplifications, such as linearizations, that do not hold in practice. These disturbances tend to negatively impact the performance of the control system, by leading to tracking errors or even instability.

The main idea of adaptive control is to estimate the uncertain parameters of a model in real time and use this information for controls. Examples of applications from this thesis where adaptive control were used are: (1) in adapting to disturbances in the terrain (Chapter 2-4) and in internal robot dynamics, as well as in (2) adapting to the center of mass offsets in the bipedal robot project (Chapter 8) and Leonardo (Chapter 7).

The adaption mechanism should guarantee that the control system remains stable and that the tracking error converges to zero. In some cases, like in composite adaptation, we can also prove that the estimated parameter vector converges to the true parameter vector. Many formalisms can be used to prove stability, such as the Lyapunov theory or Contraction theory [4, 5].

There are several types of adaptive control, as follows:

1. **Model Reference Adaptive Control (MRAC)** In MRAC, the adaptation law tries to ensure the response of the plant is the same as a reference model by using the tracking error between the two. We show this method using a first-order system [6]

$$\dot{x}(t) = -ax(t) + b(u(t) + \theta x(t)), \quad x(0) = x_0$$

where $x(t) \in \mathbb{R}$ is the state of the system, $u(t) \in \mathbb{R}$ is the control input, $a \in (0, \infty)$ is the state scalar, $b \in (0, \infty)$ is the input scalar, and $\theta \in \mathbb{R}$ is a constant matched uncertainty with the known bound $|\theta| \leq \theta_{\max}$ [6]. The reference model is given as

$$\dot{x}_m(t) = -ax_m(t) + ar(t), \quad x_m(0) = x_0$$

with state $x_m(t) \in \mathbb{R}$ and $r(t) \in \mathbb{R}$ a reference input. The MRAC controller is thus

$$u(t) = -\hat{\theta}(t)x(t) + k_g r(t) \tag{E.1}$$

where $\hat{\theta}(t) \in \mathbb{R}$ is an estimate of θ and k_g is a positive constant. The update law is given by

$$\dot{\hat{\theta}}(t) = -\Gamma x(t)e(t), \quad \hat{\theta}(0) = \hat{\theta}_0$$

where $\Gamma \in (0, \infty)$ is the adaptation gain and the tracking error signal $e(t) = x_m(t) - x(t)$.

Example 8.1 in [2] provides a good illustration of MRAC with the control of a mass on a frictionless surface by a motor force u for the plant dynamics

$$m\ddot{x} = u,$$

with m being the mass and x the position. For unknown, estimated mass $\hat{m}$, the objective is to track the reference model $\ddot{x}_m + \lambda_1 \dot{x}_m + \lambda_2 x_m = \lambda_2 r(t)$ where λ_1 and λ_2 are positive constants and $r(t)$ is the reference signal. Several notes

regarding this example, as follows: (1) Even if the dynamics is linear, the adaptation law will end up not being linear, thus the reason why we need to use Lyapunov theory or Contraction theory to prove convergence, (2) the reference model acts as a second-order filter on $r(t)$, thus $r(t)$ can be not smooth, (3) there is no guarantee that $\hat{m}$ will converge to the true m, and how close it gets to the true value depends on the choice of the $r(t)$ signal, (4) In this example, using Lyapunov theory, we can show that $x - x_m \to 0$ and $\dot{x} - \dot{x}_m \to 0$ as $t \to \infty$.

2. $\mathcal{L}_1$ **Adaptive Control** The $\mathcal{L}_1$ adaptive controller is obtained from the predictor-based MRAC by letting the control be given by

$$u(s) = C(s)\hat{\eta}(s)$$

where $C(s)$ is an exponentially stable, strictly proper low-pass filter, while $\hat{\eta}(s)$ is the Laplace transform of the controller in (E.1). The purpose of implementing this filter at the input is to allow for an increase in the adaptation gain, thereby enabling the reduction of tracking errors.

3. **Self-tuning Control (STC)** If the plant parameters are not known, we replace them with their estimates. A controller coupled with an online parameter estimator is called a self-tuning controller. In this case, the objective of STC is to perform simultaneous identification of the unknown plant. STC is also known as Adaptive Pole Placement Control (APPC).

4. **Composite Adaptation** Different sources of parameter information can be combined for adaptation. Therefore, composite adaptation combines online parameter estimation and tracking-error adaptive control to achieve better performance [2, 7]. This approach offers two key advantages: rapid adaptation and guaranteed exponential convergence of both the tracking error and the parameter estimation error.

E.4 Policy Gradient Methods. Proximal Policy Optimization

The content of this subsection is based on insights drawn from the following references [8–10].

Policy Gradient methods are a class of reinforcement learning algorithms that directly learn a policy without explicitly learning the value function. If both a value function and a policy are learned, the method is called actor-critic. Examples of

policy gradient methods include REINFORCE and Trust Region Policy Optimization (TRPO). PPO is oftentimes called a policy gradient method and extends the idea of REINFORCE with "techniques to limit the large changes in the policy action distribution in a single step" [9, 10]. Specifically, it uses a clipped version of the probability ratio between the new and old policies. Policy Gradient tries to solve the following problem: given a policy $\pi(a \mid s, \boldsymbol{\theta})$, with a being the action, s the state, and $\boldsymbol{\theta}$ the policy parameters

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J\left(\boldsymbol{\theta}_t\right)},$$

where $\widehat{\nabla J\left(\boldsymbol{\theta}_t\right)} \in \mathbb{R}^{d'}$ is a stochastic estimate whose expectation approximates the gradient of the performance measure with respect to its argument $\boldsymbol{\theta}_t$.

In off-policy RL methods, there are two policies: a behavior policy and a target policy. Exploration happens at the behavior policy. In policy gradient methods, there is only one policy, typically stochastic to ensure exploration.

For continuous actions, it is common to use a Gaussian policy as follows

$$\pi(a \mid s, \boldsymbol{\theta}) \doteq \frac{1}{\sigma(s, \boldsymbol{\theta})\sqrt{2\pi}} \exp\left(-\frac{(a - \mu(s, \boldsymbol{\theta}))^2}{2\sigma(s, \boldsymbol{\theta})^2}\right), \tag{E.2}$$

where the mean and standard deviation are learned, with $\mu : \mathcal{S} \times \mathbb{R}^{d'} \to \mathbb{R}$ and $\sigma : \mathcal{S} \times \mathbb{R}^{d'} \to \mathbb{R}^+$ being two parameterized function approximators. At runtime, the action is then sampled from the Gaussian distribution, although in practice, the mean is used.

E.5 Robot Kinematics. Lie Groups and Algebra

There are many libraries for rigid body kinematics and dynamics that rely heavily on Lie groups and Lie algebras to handle motions and rotations, for example Pinocchio [11, 12]. Therefore, understanding these concepts is crucial for anyone working with these tools. We provide a short summary of the important concepts in this section that were used in Chapter 8. Many of these concepts are summarized and adapted from [13, 14] and from Prof. Joel Burdick's Introduction to Kinematics course at Caltech.

Common Lie Groups and Algebra

A rotation matrix $\mathbf{R}$ is defined as a 3×3 matrix formed by stacking the coordinates of the principal axes of the body frame $\mathbf{B}$ relative to an inertial frame $\mathbf{A}$. A

rotation matrix has two key properties: the columns are mutually orthogonal and its determinant is 1. The set of all 3×3 matrices with these properties are called the **Special Orthogonal Group** SO(3). It is called special because the determinant of the matrix is 1. The group SO(3) is a Lie group of dimension 3.

Another common group is the **Special Euclidean Group** SE(3), which is the group of all rigid body transformations in 3D space. The group is defined as the set of mappings $\mathbf{T} : \mathbb{R}^3 \to \mathbb{R}^3$ of the form $\mathbf{T}(\mathbf{x}) = \mathbf{R}\mathbf{x} + \mathbf{p}$, where $\mathbf{R} \in \mathrm{SO}(3)$ and $\mathbf{p} \in \mathbb{R}^3$. An element of SE(3) is written as $(\mathbf{p}, \mathbf{R}) \in \mathrm{SE}(3)$. SE(3) can be identified with the space of 4×4 matrices of the form

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ 0 & 1 \end{bmatrix},$$

where $\mathbf{R} \in \mathrm{SO}(3)$ and $\mathbf{p} \in \mathbb{R}^3$. SE(3) is a Lie group of dimension 6.

By studying the Lie algebra associated with a Lie group, we can analyze group properties in terms of linear algebra, simplifying many complex problems in robotics. The Lie algebra of SO(3) is denoted as $\mathfrak{so}(3)$, and it is the set of all 3×3 skew-symmetric matrices of the form

$$\hat{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}. \tag{E.3}$$

The Lie algebra of SE(3) is denoted as $\mathfrak{se}(3)$, and it is the set of all 4×4 matrices of the form

$$\hat{\xi} = \begin{bmatrix} \hat{\boldsymbol{\omega}} & \mathbf{v} \\ 0 & 0 \end{bmatrix} \quad \boldsymbol{\omega}, \mathbf{v} \in \mathbb{R}^3, \tag{E.4}$$

where $\hat{\boldsymbol{\omega}}$ is the skew-symmetric matrix of $\boldsymbol{\omega}$. **An element of $\mathfrak{se}(3)$ is referred to as a twist**, or a (infinitesimal) generator of the Euclidean group. We define the $\vee$ (vee) operator to extract the 6-dimensional vector, which parameterizes a twist,

$$\begin{bmatrix} \hat{\boldsymbol{\omega}} & \mathbf{v} \\ 0 & 0 \end{bmatrix}^{\vee} = \begin{bmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix},$$

and call $\xi := (\mathbf{v}, \boldsymbol{\omega})$ the twist coordinates of $\widehat{\xi}$. The inverse operator, $\wedge$ (wedge), forms a matrix in $\mathfrak{se}(3)$ out of a given vector in $\mathbb{R}^6$:

$$\begin{bmatrix} v \\ \omega \end{bmatrix}^{\wedge} = \begin{bmatrix} \hat{\boldsymbol{\omega}} & \mathbf{v} \\ 0 & 0 \end{bmatrix}.$$

Thus, $\xi \in \mathbb{R}^6$ represents the twist coordinates for the twist $\widehat{\xi} \in \mathfrak{se}(3)$;

The **Exponential map** on SO(3) corresponds to a rotation about the vector $\boldsymbol{\omega} \in \mathbb{R}^3$ by an angle $\|\omega\|$. It is defined as:

$$e^{\hat{\omega}} = \mathbf{I} + \frac{\hat{\boldsymbol{\omega}}}{\|\omega\|} \sin \|\omega\| + \frac{\hat{\boldsymbol{\omega}}^2}{\|\omega\|^2}(1 - \cos \|\omega\|).$$

The exponential map on SE(3) is defined as

$$\exp \hat{\xi} = \begin{bmatrix} \mathbf{I} & \mathbf{v} \\ 0 & 1 \end{bmatrix}, \quad \omega = 0 \quad \text{and} \quad \exp \hat{\xi} = \begin{bmatrix} e^{\hat{\omega}} & \mathbf{Av} \\ 0 & 1 \end{bmatrix}, \quad \omega \neq 0 \tag{E.5}$$

where

$$\mathbf{A} = \mathbf{I} + \frac{\hat{\boldsymbol{\omega}}}{\|\boldsymbol{\omega}\|^2}(1 - \cos \|\boldsymbol{\omega}\|) + \frac{\hat{\boldsymbol{\omega}}^2}{\|\boldsymbol{\omega}\|^3}(\|\boldsymbol{\omega}\| - \sin \|\boldsymbol{\omega}\|).$$

The **Logarithm map** is the inverse of the exponential map.

The structure of Lie groups and their corresponding algebras provides powerful tools for analysis and control. For example, exponential and logarithm maps help the handling of state transitions and error analysis in multi degrees of freedom robotic systems.

Manipulator Jacobian

The Jacobian gives the relationship between the joint angles and the end-effector velocities. There are two types of Jacobians that are typically computed: the spatial Jacobian and the body Jacobian. For a general open-chain robot with n joints, both Jacobians are 6 by n. The space Jacobian related the velocity of the end effector in the Spatial frame to the joint velocities, while the body Jacobian relates the velocity of the end effector in the body frame to the joint velocities, as follows:

$$\boldsymbol{\nu}_s = \mathbf{J}_s(\mathbf{q})\dot{\mathbf{q}}$$
$$\text{where } \mathbf{J}_s(\mathbf{q}) = \begin{bmatrix} \mathbf{J}_{s1}(\mathbf{q}) & \mathbf{J}_{s2}(\mathbf{q}) \cdots & \mathbf{J}_{sn}(\mathbf{q}) \end{bmatrix} \in \mathbb{R}^{6\times n},$$

where $\mathbf{q}$ is the joint angles, $\boldsymbol{\nu}_s$ is the end-effector twist in the spatial frame, and $\mathbf{J}_s$ is the spatial Jacobian.

The manipulator Jacobian can also be used to describe the relationship between wrenches applied at the end-effector and joint torques. [13].

Singularities

The rank of the Jacobian can be no greater than the minimum of 6 and n. We say a Jacobian is singular at a configuration $\mathbf{q}^*$ if the rank of the Jacobian is less the

maximum rank at some configuration. At the singular configuration, the robot loses the ability to move in certain directions.

Singuralities posed some challenges in the development of the bipedal robot controller, which you can read about in Chapter 8. Specifically, when running Inverse Kinematics for the feet, singularities caused the Jacobian to become non-invertible. In these situations, the robot exhibited aggressive foot shaking behavior, which led to motors failures. In Chapter 8, Sec. 8.3 we present how we handled singularities in the bipedal robot controller.

BIBLIOGRAPHY

[1] Hassan K. Khalil. *Nonlinear Control*. Always Learning. Pearson, 2014.

[2] Jean-Jacques E. Slotine and Weiping Li. *Applied Nonlinear Control*. Englewood Cliffs, N.J: Prentice Hall, 1991.

[3] Petros Ioannou and Simone Baldi. "Robust Adaptive Control". In: Dec. 2010, pp. 1–22.

[4] Winfried Lohmiller and Jean-Jacques E. Slotine. "On Contraction Analysis for Non-linear Systems". In: *Automatica* 34.6 (1998), pp. 683–696.

[5] Hiroyasu Tsukamoto, Soon-Jo Chung, and Jean-Jaques E. Slotine. "Contraction Theory for Nonlinear Stability Analysis and Learning-based Control: A Tutorial Overview". In: *Annu. Reviews Control* 52 (Oct. 2021), pp. 135–169.

[6] Naira Hovakimyan and Chengyu Cao. *L1 Adaptive Control Theory*. Society for Industrial and Applied Mathematics, 2010.

[7] Jean-Jacques E. Slotine and Weiping Li. "Composite Adaptive Control of Robot Manipulators". In: *Automatica* 25.4 (1989), pp. 509–519.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[9] Alan James Preiss. *Characterizing and Improving Robot Learning: A Control-Theoretic Perspective*. 2022.

[10] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. *Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO*. 2020. arXiv: 2005.12729 [cs.LG].

[11] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. "The Pinocchio C++ Library – A Fast and Flexible Implementation of Rigid Body Dynamics Algorithms and Their Analytical Derivatives". In: *IEEE International Symposium on System Integrations (SII)*. 2019.

[12] Justin Carpentier, Florian Valenza, Nicolas Mansard, et al. *Pinocchio: A Fast and Flexible Implementation of Rigid Body Dynamics Algorithms and Their Analytical Derivatives*. https://stack-of-tasks.github.io/pinocchio. 2015–2021.

[13] Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*. 1st. USA: CRC Press, Inc., 1994.

[14] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. 1st. USA: Cambridge University Press, 2017.