UNIVERSITY OF CALIFORNIA SAN DIEGO

Learning Environment and Dynamics Representations for Autonomous Robot Navigation

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

in

Electrical Engineering (Intelligent Systems, Robotics, and Control)

by

Thai Phu Duong

Committee in charge:

Professor Nikolay Atanasov, Chair
Professor Henrik Christensen
Professor Melvin Leok
Professor Michael Yip

2024

The Dissertation of Thai Phu Duong is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2024

DEDICATION

To my family,

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Dr. Nikolay Atanasov, for his unwavering support and guidance throughout the years. He gave me an opportunity to join his lab, taught me how to be an independent researcher, and more importantly showed me how to find joy and stay resilient through the ups and downs of my PhD journey. He also offered me the freedom to pursue my research passion and motivated me to excite other people with my ideas. He provided constant support day and night for many projects that we worked on together, leading to several interesting ideas, papers and experiments. His relentless patience and encouragement have been the driving force behind my research throughout the years.

I thank my PhD committee members, Dr. Henrik Christensen, Dr. Melvin Leok, and Dr. Michael Yip, for accommodating my qualifying exam and final defense schedule, showing interest in my research, reviewing and providing insightful comments on my work. I thank Dr. Michael Yip for working with me on learning sparse environment representations for navigation, and giving me valuable advice during my PhD. I thank Dr. Eduardo Montijano, Dr. Carlos Sagues, Dr. George Pappas, and Dr. Quan Nguyen for collaborating with me in various projects, ranging from learning distributed controllers and planning with large language models to learning dynamics for quadruped jumping. I thank my former advisors, Dr. Thinh Nguyen and Dr. Thang Hoang, for introducing me to research and inspiring me to pursue my passion.

I thank my co-authors, Nikhil Das, Zhichao Li, Eduardo Sebastian, Valentin Duruisseaux, Abdullah Altawaitan, Jason Stanley, Sambaran Ghosal, Zhirui Dai, Arash Asgharivaskasi, and Chuong Nguyen for brainstorming ideas and solutions, performing experiments, and especially sharing both tough and joyful moments while working on our papers. I thank my ERL labmates for interesting discussions on various research topics and challenges of graduate school. Special thanks go to Zhichao Li, Abdullah Altawaitan, Jason Stanley, and Minh Pham for their relentless help with real experiments on the ground and aerial robots through several days and nights. I thank Qiaojun Feng, Tianyu Wang, and Eduardo Sebastian for midnight discussions on random problems in our life. I thank Ahmed Qureshi and Carlos Nieto-Granda for guiding me through the difficult times at the beginning of my PhD.

I would like to express my appreciation to our Racecar and Jackal ground robots, and

our quadrotors, Penguin and Kiwi, for always staying strong through several crashes and putting their life at risk for the success of our demonstrations. My dissertation would not have been possible without them working their batteries out.

I also had great pleasure of having many friends at UC San Diego, Ninh Tran, Loan Le, Duc Tran, Tan Trinh, Phuong Nguyen, Trung Tran, Chi Nguyen, Uyen Mai, Bao Lam, Ha Pham, Ly Tran, Duong Hoang, Duong Nguyen, Vy Nguyen, Hoa Phan, Son Nguyen, Thu Phan, Son Le, Nam Vu, Hieu Pham, Huong Hoang, Tri Hoang, and others. I thank them for the never-ending support and encouragement they have brought to my life. I thank my old friends, Thuan Duong-Ba, Duong Nguyen-Huu, Tram Hoang and Nam Pham, for always rooting for my career. I thank Nate, Will, Catherine, Maria, Vivek, Mi, Duy and Mohammad for sharing the great pleasure and challenges of our student-parent life.

Finally, I would like to express my deepest appreciation to my family for supporting my research career. I thank my wife, Huong Luong, and my children, Chi Duong and Minh Duong, for their understanding, love and unconditioned support. I thank my parents and parents-in-law for believing in me and for their immeasurable help with our kids. I thank my grandparents for giving me my first lessons of algebra and passing their interests in academics on to me.

Chapters 1, 2 and 5, in part, are reprints of the material as it appears in T. Duong, M. Yip, N. Atanasov, "*Autonomous Navigation in Unknown Environments with Sparse Bayesian Kernel-based Occupancy Mapping*", IEEE Transactions on Robotics (T-RO), vol. 38, no. 6, pp. 3694-3712, 2022, in T. Duong, N. Das, M. Yip, N. Atanasov, "*Autonomous Navigation in Unknown Environments using Sparse Kernel-based Occupancy Mapping*", International Conference on Robotics and Automation (ICRA), pp. 9666-9672, 2020, in T. Duong, N. Atanasov, "*Hamiltonian-based Neural ODE Networks on the SE(3) Manifold For Dynamics Learning and Control*", Robotics: Science and Systems (RSS), 2021, in T. Duong, N. Atanasov, "*Physics-guided Learning-based Adaptive Control on the SE(3) Manifold*", Physical Reasoning and Inductive Biases for the Real World Workshop, 2021, and in T. Duong, N. Atanasov, "*Adaptive Control of SE(3) Hamiltonian Dynamics with Learned Disturbance Features*", IEEE Control Systems Letters (L-CSS), vol. 6, pp. 2773-2778, 2022. Chapters 1, 2 and 5, in part, have been submitted for publication of the material as it may appear in T. Duong, A. Altawaitan, J.

Stanley, N. Atanasov, "*Port-Hamiltonian-based Neural ODE Networks on Lie Groups For Robot Dynamics Learning and Control*", Transactions on Robotics, 2024. The dissertation author was the primary investigator and author of these papers.

Chapter 3, in part, is a reprint of the material as it appears in T. Duong, M. Yip, N. Atanasov, "*Autonomous Navigation in Unknown Environments with Sparse Bayesian Kernel-based Occupancy Mapping*", IEEE Transactions on Robotics (T-RO), vol. 38, no. 6, pp. 3694-3712, 2022, in T. Duong, N. Das, M. Yip, N. Atanasov, "*Autonomous Navigation in Unknown Environments using Sparse Kernel-based Occupancy Mapping*", International Conference on Robotics and Automation (ICRA), pp. 9666-9672, 2020. The dissertation author was the primary investigator and author of these papers.

Chapter 4, in part, is a reprint of the material as it appears in T. Duong, N. Atanasov, "*Hamiltonian-based Neural ODE Networks on the SE(3) Manifold For Dynamics Learning and Control*", Robotics: Science and Systems (RSS), 2021, in T. Duong, N. Atanasov, "*Physics-guided Learning-based Adaptive Control on the SE(3) Manifold*", Physical Reasoning and Inductive Biases for the Real World Workshop, 2021, and in T. Duong, N. Atanasov, "*Adaptive Control of SE(3) Hamiltonian Dynamics with Learned Disturbance Features*", IEEE Control Systems Letters (L-CSS), vol. 6, pp. 2773-2778, 2022. Chapter 4, in part, has been submitted for publication of the material as it may appear in T. Duong, A. Altawaitan, J. Stanley, N. Atanasov, "*Port-Hamiltonian-based Neural ODE Networks on Lie Groups For Robot Dynamics Learning and Control*", Transactions on Robotics, 2024. The dissertation author was the primary investigator and author of these papers.

VITA

2011	Bachelor of Science, Hanoi University of Science and Technology, Vietnam
2015	Master of Science, Oregon State University
2015–2018	Software Engineer, Microsoft Corporation
2018–2024	Graduate Student Researcher, University of California San Diego
2024	Doctor of Philosophy, University of California San Diego

PUBLICATIONS

T. Duong, A. Altawaitan, J. Stanley, N. Atanasov, "Port-Hamiltonian-based Neural ODE Networks on Lie Groups For Robot Dynamics Learning and Control", under review, 2024.

E. Sebastian, **T. Duong**, N. Atanasov, E. Montijano and C. Sagues, "Physics-Informed Multi-Agent Reinforcement Learning for Distributed Multi-Robot Problems", under review, 2024.

T. Duong, M. Yip, N. Atanasov, "Autonomous Navigation in Unknown Environments with Sparse Bayesian Kernel-based Occupancy Mapping", IEEE Transactions on Robotics (T-RO), vol. 38, no. 6, pp. 3694-3712, 2022.

T. Duong, N. Atanasov, "Adaptive Control of SE(3) Hamiltonian Dynamics with Learned Disturbance Features", IEEE Control Systems Letters (L-CSS), vol. 6, pp. 2773-2778, 2022.

Z. Li, **T. Duong**, N. Atanasov, "Robust and Safe Autonomous Navigation for Systems with Learned SE(3) Hamiltonian Dynamics", IEEE Open Journal of Control System (OJ-CSYS), vol. 1, pp. 164-179, 2022.

Z. Dai, A. Asgharivaskasi, **T. Duong**, S. Lin, M. Tzes, G. Pappas, N. Atanasov, " Optimal Scene Graph Planning with Large Language Model Guidance", IEEE International Conference on Robotics and Automation (ICRA), 2024.

A. Altawaitan, J. Stanley, S. Ghosal, **T. Duong**, N. Atanasov, " Hamiltonian Dynamics Learning from Point Cloud Observations for Nonholonomic Mobile Robot Control", IEEE International Conference on Robotics and Automation (ICRA), 2024.

E. Sebastian, **T. Duong**, N. Atanasov, E. Montijano and C. Sagues, "Learning to Identify Graphs from Node Trajectories in Multi-Robot Networks", International Symposium on Multi-Robot and Multi-Agent Systems (MRS), 2023.

V. Duruisseaux, **T. Duong**, N. Atanasov, M. Leok, "Lie Group Forced Variational Integrator Networks for Learning and Control of Robot Systems", Learning for Dynamics & Control Conference (L4DC), pp. 731-744, 2023.

E. Sebastian, **T. Duong**, N. Atanasov, E. Montijano and C. Sagues, "LEMURS: Learning Distributed Multi-robot Interactions", IEEE International Conference on Robotics and Automation (ICRA), pp. 7713-7719, 2023.

Z. Li*, **T. Duong***, N. Atanasov, "Safe Autonomous Navigation for Systems with Learned SE(3) Hamiltonian Dynamics", Learning for Dynamics & Control Conference (L4DC), pp. 981-993, 2022. *Equal contribution.

T. Duong, N. Atanasov, "Hamiltonian-based Neural ODE Networks on the SE(3) Manifold For Dynamics Learning and Control", Robotics: Science and Systems (RSS), 2021.

T. Duong, N. Das, M. Yip, N. Atanasov, "Autonomous Navigation in Unknown Environments using Sparse Kernel-based Occupancy Mapping", International Conference on Robotics and Automation (ICRA), pp. 9666-9672, 2020.

T. Duong, N. Atanasov, "Physics-guided Learning-based Adaptive Control on the SE(3) Manifold", Physical Reasoning and Inductive Biases for the Real World Workshop at NeurIPS, 2021.

FIELDS OF STUDY

Major Field: Electrical and Computer Engineering

Studies in Intelligent Systems, Robotics & Control
Professor Nikolay Atanasov

ABSTRACT OF THE DISSERTATION

Learning Environment and Dynamics Representations for Autonomous Robot Navigation

by

Thai Phu Duong

Doctor of Philosophy in Electrical Engineering (Intelligent Systems, Robotics, and Control)

University of California San Diego, 2024

Professor Nikolay Atanasov, Chair

Robot systems have become prevalent and transformative in many areas, such as environment surveillance and reconnaissance, search and rescue, industrial manufacturing, and transportation. In these applications, it is critical for robots to navigate autonomously and reliably in the environment in order to execute their tasks. This requires efficient maintenance of an environment model, offering minimal storage footprint and fast inference time, and an accurate robot dynamics model, enabling stable and robust control policies in novel operating conditions. This dissertation proposes a novel autonomous navigation approach that utilizes machine learning techniques to develop sparse probabilistic occupancy maps of the environment and learn robot dynamics accurately and efficiently from data by preserving prior knowledge in the dynamics model.

The first part of the dissertation develops a compact machine learning model, trained online from streaming sensory data, to represent the occupancy of the environment. While common occupancy maps might have high storage requirements for large environments, we propose a novel approach that models the obstacle boundary as the decision boundary of a machine learning classifier, and thus, scales with the complexity of the boundary instead of the environment size. We develop online training algorithms of kernel perceptron and relevance vector machine classifiers to incrementally build sparse binary and probabilistic occupancy maps, respectively, from local observations.

The second part of the dissertation proposes a machine learning model for learning accurate robot dynamics from state-control trajectories. While hand-designed models might over-simplify the dynamical system, black-box models recently have become increasingly popular but require a large amount of data for training. We develop a data-efficient hybrid approach by encoding prior knowledge such as universal laws of physics and the kinematic structure of the state manifold in the dynamics model. The encoded prior knowledge is guaranteed by design instead of being inferred from data. In novel operating conditions, this approach is extended to learn a disturbance model to handle dynamics changes.

The dissertation finally develops efficient collision checking algorithms for motion planning with the learned sparse map representations and trajectory-tracking control policies based on the learned robot dynamics and disturbance models, offering a fast, reliable, and long-term solution for autonomous navigation. The autonomous navigation approach is verified extensively with datasets, simulated and real robot experiments.

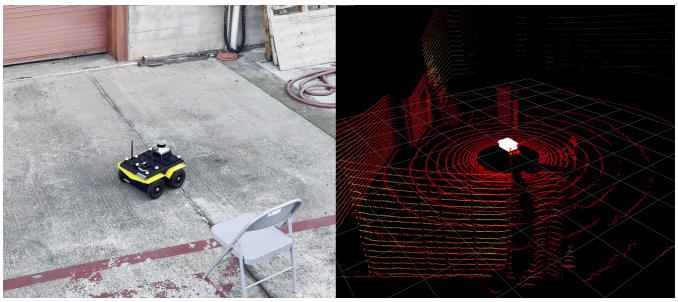
Chapter 1

Introduction

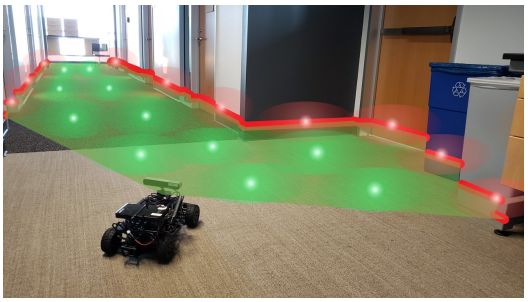
1.1 Motivation

Autonomous navigation in an unstructured and unknown environment is a fundamental problem in many robotics applications such as warehouse automation, transportation, surveillance, and environment monitoring. It depends on the availability of an accurate system dynamics model and a scalable environment representation for motion planning and control. Traditionally, autonomous navigation relies on hand-designed robot dynamics from system identification and occupancy maps that label each point in the environment as free or occupied. However, hand-designed robot dynamics may be insufficiently accurate even after careful parameter tuning while common occupancy representations for navigation may have high storage requirements for large environments.

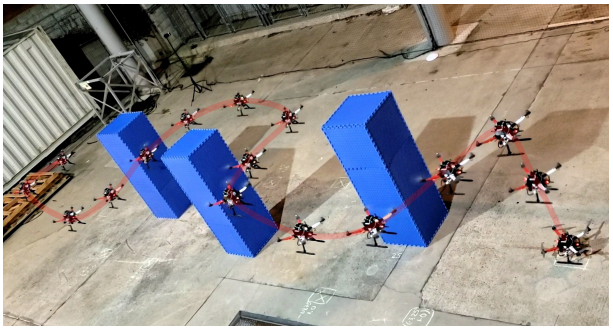
Recently, the abundance of data from onboard sensors offers an incredible opportunity to learn accurate, yet efficient, representations for environments and robot dynamics, enabling safe, scalable and reliable autonomous robot navigation (Figure 1.1). Therefore, the goal of this dissertation is to develop a novel and effective solution for autonomous navigation that: 1) encodes prior knowledge and constraints in machine learning models to *efficiently learn robot dynamics and control from data*; and 2) builds *large-scale, yet compact, environment representations* by modeling obstacle boundaries using a machine learning classifier. In the next chapters, we study both how to learn efficient representations for environment understanding (Chapter 3) and robot dynamics (Chapter 4), and develop planning and control algorithms with the proposed representations to complete the autonomous navigation task (Chapter 5).



(a) A ground robot observes the environment via a Lidar (courtesy of Altawaitan et al. [2]).



(b) The robot builds a map of the environment.



(c) Using the map and robot dynamics, the robot plans and follows a trajectory to finish their tasks.

Figure 1.1. Autonomous navigation task in mobile robots.

1.2 Autonomous Robot Navigation Problem

In this section, we describe the main problems in autonomous robot navigation that we consider in this dissertation, including mapping, dynamics learning, planning and control.

Consider a robot with state $\mathbf{s} \in \mathcal{S}$, consisting of the robot's position $\mathbf{p} \in [0,1]^d$ and other variables such as orientation, velocity, etc., navigating in an unknown environment. Let $\mathcal{O} \subset [0,1]^d$ be a closed set representing occupied space and let $\mathcal{F}$ be its complement, representing free space. Assume that the robot can be enclosed by a sphere of radius $r \in \mathbb{R}_{>0}$ centered at $\mathbf{p}$. In configuration space (C-space), the robot's body becomes a point $\mathbf{p}$, while the obstacle space and free space are transformed as $\bar{\mathcal{O}} = \cup_{\mathbf{x} \in \mathcal{O}} \mathcal{B}(\mathbf{x}, r)$, where $\mathcal{B}(\mathbf{x}, r) = \{\mathbf{x}' \in [0,1]^d : \|\mathbf{x} - \mathbf{x}'\|_2 \leq r\}$, and $\bar{\mathcal{F}} = [0,1]^d \setminus \bar{\mathcal{O}}$. Let $\bar{\mathcal{S}}$ be the subset of the robot state space that corresponds to the collision-free robot positions $\bar{\mathcal{F}}$. The robot is equipped with a sensor, such as a Lidar or depth camera, that provides distance measurements $\mathbf{z}_k$ at time t_k to the obstacle space $\mathcal{O}$ within its field of view. An important task in autonomous navigation is to construct an occupancy map

$m_k : [0,1]^d \to \{-1, 1\}$ of the C-space based on accumulated observations $\mathbf{z}_{0:k}$, where "-1" and "1" mean "free" and "occupied", respectively. As the robot is navigating, new sensor data is used to update the map via a function, $m_{k+1} = g(m_k, \mathbf{z}_k)$, of the previous estimate m_k and a newly received range observation $\mathbf{z}_k$.

Problem 1 (Mapping)**.** Given a previous occupancy map m_k and a local range observation $\mathbf{z}_k$ of the environment, find a function g that updates the map of the environment based on the current measurements $\mathbf{z}_k$: $m_{k+1} = g(m_k, \mathbf{z}_k)$.

Let $\dot{\mathbf{s}} = \mathbf{f}(\mathbf{s}, \mathbf{u})$ characterize the continuous-time robot dynamics with control input trajectory $\mathbf{u} \in \mathcal{U}$. We consider constant control inputs (zero-order hold) applied at discrete time steps t_k for $k = 0, 1, \ldots, N$ so that $\mathbf{u}(t) \equiv \mathbf{u}_k$ for $[t_k, t_{k+1})$. We assume that the state $\mathbf{s}$ is known or estimated by a localization algorithm and let $\mathbf{s}_k := \mathbf{s}(t_k)$ at time t_k.

As the dynamics function $\mathbf{f}$ is often unknown, an accurate approximated dynamics function $\bar{\mathbf{f}}_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$ is critical for the robot to navigate in the environment. Let $\mathcal{D} = \{t^{(i)}_{0:N}, \mathbf{s}^{(i)}_{0:N}, \mathbf{u}^{(i)}\}_{i=1}^{D}$ consist of D state sequences $\mathbf{s}^{(i)}_{0:N}$, obtained by applying a constant control input $\mathbf{u}^{(i)}$ to the system with initial condition $\mathbf{s}^{(i)}_0$ at time $t^{(i)}_0$ and sampling its state $\mathbf{s}^{(i)}(t^{(i)}_n) =: \mathbf{s}^{(i)}_n$ at times $t^{(i)}_0 < t^{(i)}_1 < \ldots < t^{(i)}_N$. Using the dataset $\mathcal{D}$, we aim to find a function $\bar{\mathbf{f}}_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$ that approximates the true dynamics $\mathbf{f}$ well. To optimize $\boldsymbol{\theta}$, we roll out the approximate dynamics $\bar{\mathbf{f}}_{\boldsymbol{\theta}}$ with initial state $\mathbf{s}^{(i)}_0$ and constant control $\mathbf{u}^{(i)}$ and minimize the discrepancy between the computed state sequence $\bar{\mathbf{s}}^{(i)}_{1:N}$ and the true state sequence $\mathbf{s}^{(i)}_{1:N}$ in $\mathcal{D}$.

Problem 2 (Dynamics Learning)**.** Given a dataset $\mathcal{D} = \{t^{(i)}_{0:N}, \mathbf{s}^{(i)}_{0:N}, \mathbf{u}^{(i)}\}_{i=1}^{D}$ and a function $\bar{\mathbf{f}}_{\boldsymbol{\theta}}$, find the parameters $\boldsymbol{\theta}$ that minimize:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \; & \sum_{i=1}^{D} \sum_{n=1}^{N} \ell(\mathbf{s}^{(i)}_n, \bar{\mathbf{s}}^{(i)}_n) \\ \text{s.t. } & \dot{\bar{\mathbf{s}}}^{(i)}(t) = \bar{\mathbf{f}}_{\boldsymbol{\theta}}(\bar{\mathbf{s}}^{(i)}(t), \mathbf{u}^{(i)}), \;\; \bar{\mathbf{s}}^{(i)}(t_0) = \mathbf{s}^{(i)}_0, \\ & \bar{\mathbf{s}}^{(i)}_n = \bar{\mathbf{s}}^{(i)}(t_n), \;\; \forall n = 1, \ldots, N, \;\; \forall i = 1, \ldots, D, \end{aligned} \tag{1.1}$$

where ℓ is a distance metric on the state space.

Assuming unobserved regions are free, we rely on the current map m_k to plan a robot

trajectory to a goal region $\mathcal{G} \subseteq \bar{\mathcal{S}}$. Applying control action $\mathbf{a}$ at $\mathbf{s}$ incurs a motion cost $c(\mathbf{s}, \mathbf{a})$, e.g., based on traveled distance or energy expenditure. To be able to navigate in the environment, we aim to solve: 1) a *planning problem* where we find a sequence of desired control inputs that minimizes the cumulative cost of navigating safely to the goal $\mathcal{G}$ (Problem 3); and 2) a *control problem* where we find a feedback control policy that tracks the desired robot trajectory, generated from the desired control sequence (Problem 4).

Problem 3 (Planning)**.** Given a start state $\mathbf{s}_0 \in \bar{\mathcal{S}}$ and a goal region $\mathcal{G} \subseteq \bar{\mathcal{S}}$, find a sequence of control inputs that leads the robot to the goal region $\mathcal{G}$ safely, while minimizing the cost:

$$\begin{aligned} \min_{N, \mathbf{u}_0, \ldots, \mathbf{u}_N} \quad & \sum_{k=0}^{N-1} c(\mathbf{s}_k, \mathbf{u}_k) \\ \text{s.t.} \quad & \dot{\mathbf{s}} = \bar{\mathbf{f}}_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{u}), \mathbf{u}(t) = \mathbf{u}_k \text{ for } t \in [t_k, t_{k+1}), \mathbf{s}(t_0) = \mathbf{s}_0,\ \mathbf{s}_N \in \mathcal{G}, \\ & m_k(\mathbf{s}(t)) = -1 \text{ for } t \in [t_k, t_{k+1}), k = 0, \ldots, N. \end{aligned} \tag{1.2}$$

Let $\{\mathbf{u}_i^*\}_{i=0}^N$ be the optimal sequence of open-loop control inputs that solves Problem 3 and generates a desired trajectory $\mathbf{s}^*(t)$ for the robot to follow. However, due to imprecise dynamics model or disturbance in reality, the robot will not be able to accurately track the desired trajectory $\mathbf{s}^*(t)$ using the open-loop control sequence $\{\mathbf{u}_i^*\}_{i=0}^N$. Therefore, we aim to design a feedback controller capable of tracking a desired state trajectory $\mathbf{s}^*(t)$, $t \geq t_0$ for the learned robot dynamics function $\mathbf{f}_{\boldsymbol{\theta}}$.

Problem 4 (Control)**.** Given an initial condition $\mathbf{s}_0$ at time t_0, a desired state trajectory $\mathbf{s}^*(t)$, $t \geq t_0$, and robot dynamics $\bar{\mathbf{f}}_{\boldsymbol{\theta}}$, design a feedback control policy $\mathbf{u} = \boldsymbol{\pi}(\mathbf{s}, \boldsymbol{\theta}, \mathbf{s}^*(t))$ such that the tracking error $\limsup_{t \to \infty} \ell(\mathbf{s}(t), \mathbf{s}^*(t))$ is bounded, where $\ell(\mathbf{s}(t), \mathbf{s}^*(t))$ measures the discrepancy between the desired state $\mathbf{s}^*(t)$ and the actual state $\mathbf{s}(t)$.

In the remainder of the dissertation, we aim to:

- solve Problem 1 in Chapter 3 by developing a sparse Bayesian kernel-based occupancy map representation with efficient storage requirements while providing map uncertainty for other downstream tasks,

- solve Problem 2 in Chapter 4 by learning the robot dynamics $\mathbf{f}_{\boldsymbol{\theta}}$ from data while preserving Hamiltonian structure and the kinematic constraints of Lie groups for generalization, data efficiency, and long-term predictions,

- solve Problem 3 by developing collision checking algorithms for common motion planning methods to generate desired robot trajectories, and solve Problem 4 by designing an energy-based trajectory tracking control policy for the learned dynamics, handling dynamics changes in new operating conditions in Chapter 5.

1.3 Related Work

In this section, we provide an overview of the related work on mapping, collision checking for planning, dynamics and control designs for autonomous navigation.

While perceiving the environment, a robot often builds an environment representation from streaming data from onboard sensors [78]. The environment representation can model the obstacle surfaces explicitly [173, 9, 81, 16, 152, 135, 141] or implicitly via occupancy [43, 70, 181, 171] or signed distance [127, 63]. This dissertation focuses on occupancy maps, which are commonly used for modeling the free and occupied space of an environment. The space is discretized into a collection of cells, whose occupancy probabilities are estimated online using the robot's sensory data. While early work [163, 56] assumes that the cells are independent, Gaussian process (GP) occupancy mapping [126, 172, 77] uses a kernel function to capture the correlation among grid cells and predict the occupancy of unobserved cells. Online training of a Gaussian process model, however, does not scale well as its computational complexity grows cubically with the number of data points. Ramos et al. [137] improve on this by projecting the data points into Hilbert space and training a logistic regression model. Senanayake and Ramos [150] propose a Bayesian treatment of Hilbert maps, called Sequential Bayesian Hilbert Map (SBHM), that updates the map from sequential observations of the environment. They achieve sparseness by calculating feature vectors based on a sparse set of hinged points, e.g., on a coarse grid. Instead of a fixed set of hinged points, localized automatic relevance determination Hilbert maps (LARD-HM) [58] and efficient Hilbert maps (EHM) [59] find the hinged points by clustering the training data points using k-means algorithms and calculate their kernel parameters using

automatic relevance determination. Meanwhile, Relevance Vector Machine (RVM) [164, 165, 166] learns a sparse set of relevance vectors from the training dataset. The original RVM work [164] initially assumes that all data points are relevance vectors and prunes them down, incurring high computation cost. Tipping and Faul [166] derive a fast training algorithm that starts from an empty set of relevance vectors and adds points to the set gradually. Meanwhile, Lopez and How [103] propose an efficient deterministic alternative, which builds a k-d tree from point clouds and queries the nearest obstacles for collision checking. Using spatial partitioning similar to a k-d tree, octree-based maps [70, 19] offer efficient map storage by performing octree compression, while AtomMap [49] stores a collection of spheres in a k-d tree as a way to avoid grid cell discretization of the map. Instead of storing occupancy information, Voxblox [127] stores distance to obstacles in each cell and builds an Euclidean Signed Distance Field, as a map representation, online from streaming sensor data.

Given a map of the environment, navigation tasks requires planning a safe robot trajectory, e.g., using RRT^* [84] or A^* [98], which in turns, needs to evaluate a large amount of collision checks for motion primitives, e.g., points, line segments or polynomials [11, 107, 66]. Common sampling-based collision checking methods are potentially time-consuming as a finite set of points are sampled on each motion primitive and checked for collision separately. Bialkowski et al. [11] improve the checking time by using the distance to the closest obstacle to choose the next sampling point to check. Recently, learning-based collision checking methods [30, 130, 73] train a machine learning model of the obstacle boundaries and show that collision checking time for a point outperforms geometry-based methods such as the Flexible Collision Library (FCL) [129].

Dynamics models obtained from first principles via system identification [102] are commonly used in robotics, but may over-simplify the dynamical system, leading to bias and modeling errors. Meanwhile, data-driven techniques [159, 144, 147, 104, 74] have emerged as a powerful approach to approximate system dynamics with an over-parameterized machine learning model, e.g. neural networks, but typically require large amounts of data and computation time. Recent works [110, 60, 27, 55, 23, 140, 106, 123] have considered a hybrid approach, where prior knowledge of the physics, governing the system dynamics, is integrated into a machine learning model to represent robot dynamics. The encoded prior knowledge ranges from kine-

matic structure [145], symmetry in rotation, scaling, and uniform motion [174], to the law of energy conservation via Lagrangian formulation [140, 111, 60, 27, 109, 110, 106] or Hamiltonian formulation [55, 10, 23, 46, 184, 177, 123, 185, 179]. Given a robot dynamics model, a control policy is often designed for stabilization and trajectory tracking using well-studied control theory techniques [86], such as feedback linearization, control Lyapunov functions, or passivity-based control. For Hamiltonian systems, which are based on the notion of energy, energy-shaping control techniques add additional energy via the control input so that the closed-loop system's total energy is minimized at the the desired state [170, 170, 128, 1, 25]. The dynamics can also be discretized to be used with model predictive control (MPC) [14, 57].

When disturbances and system changes during online operation bring about new out-of-distribution data, it is often too slow to re-train the nominal dynamics model to support real-time adaptation to environment changes. Instead, adaptive control [89, 75] offers efficient tools to estimate and compensate for disturbances and parameter variations online. A key technical challenge in adaptive control is the design of an adaptation law that estimates the disturbance online [89]. The disturbance can be non-parametric [71, 51, 64, 22, 94] or parametric [156, 155, 35, 146], e.g. a linear combinations of *known* nonlinear features, and is updated based on the state errors with stability obtained by sliding-mode theory [156, 155, 35], assuming zero-state detectability [146, 35] or $\mathcal{L}_1$-adaptation [71, 51, 64]. If the system evolves on a manifold (e.g., when the state contains orientation), an adaptation law is designed based on geometric errors, derived from the manifold constraints [53, 12]. A disturbance observer [22, 94] use the state errors introduced by the disturbances to design an asymptotically stable observer system that estimates the disturbance online. A disturbance adaptation law is paired with a nominal controller, derived using Lagrangian dynamics with feedback linearization [156, 157], Hamiltonian dynamics with energy shaping [121, 35], or model predictive control [133, 64].

Recently, there has been growing interest in applying machine learning techniques to design adaptive controllers. As the nonlinear disturbance features are actually *unknown* in practice, they can be estimated using Gaussian processes [51, 54] or neural networks [80, 139]. The features can be learned online in the control loop [51, 80], which is potentially slow for real-time operation, or offline via meta-learning from past state-control trajectories [65] or system

dynamics simulation [139]. Given the learned disturbance features, an adaptation law is designed to estimate the disturbances online, e.g. using $\mathcal{L}_1$-adaptation [51] or by updating the last layer of the feature neural network [80, 139, 153].

1.4 Overview and Contributions

This dissertation develops an effective autonomous navigation approach that learns efficient environment and dynamics representations from data. We focus on developing the following components in autonomous navigation:

- sparse probabilistic occupancy map representations for continuous-space large-scale environments, built online from streaming sensor measurements while providing map uncertainty for other downstream tasks such as exploration (Chapter 3),
- a Hamiltonian-based neural ODE model on Lie groups that efficiently learns robot dynamics and disturbance models from state-control trajectories by embedding the law of energy conservation and Lie group constraints for generalization, data efficiency, and long-term state predictions (Chapter 4),
- efficient collision checking (without sampling) for trajectory planning based on the sparse probabilistic occupancy maps, and stable and adaptive energy-based trajectory-tracking control design based on the learned map and dynamics, leading to fast, reliable and large-scale autonomous navigation (Chapter 5).

The organization of the dissertation and the contributions for each chapter are summarized as follows.

Chapter 2 provides the necessary background to develop our techniques in the following chapters, such as matrix Lie groups, Hamiltonian dynamics, neural ODE networks, and machine learning classifiers.

Chapter 3 focuses on online occupancy mapping onboard an autonomous robot navigating in a large unknown environment. Commonly used voxel and octree map representations can be easily maintained in a small environment but have increasing memory requirements as the environment grows. We propose a fundamentally different occupancy mapping approach, where

the obstacle boundary is modeled as the decision boundary of a machine learning classifier. This chapter generalizes a kernel perceptron model which maintains a very sparse set of support vectors to represent the environment boundaries efficiently. We develop a probabilistic formulation based on Relevance Vector Machines, handling measurement noise, and probabilistic occupancy classification, supporting autonomous navigation. In this chapter, we provide an online training algorithm, updating the sparse Bayesian map incrementally from streaming range data. The effectiveness of our mapping algorithms is evaluated with various real and simulated dataset of sequential laser scans in different environments.

Chapter 4 considers the problem of learning accurate robot dynamics from data, which are critical for safe and stable control and generalization to novel operational conditions. While hand-designed models may be insufficiently accurate, machine learning techniques have recently become a common choice to approximate the robot dynamics over a training set of state-control trajectories. The dynamics of many robots are described in terms of their generalized coordinates on a matrix Lie group, e.g. on the $SE(3)$ manifold for ground, aerial, and underwater vehicles, and generalized velocity, and satisfy conservation of energy principles. This chapter proposes a Hamiltonian formulation over a Lie group of the structure of a neural ordinary differential equation (ODE) network [21] to approximate the robot dynamics. In contrast to a black-box ODE network, our formulation preserves energy conservation principle and Lie group's constraints by construction and explicitly accounts for energy-dissipation effects such as friction and drag forces in the dynamics model. While the learned dynamics model can be fine-tuned quickly to handle dynamics changes, it might still be too slow for real-time adaptation. In the second part of this chapter, we learn a parametric model of the dynamics changes or disturbances, described as a linear combination of disturbance features, e.g., neural networks. We train the disturbance feature networks using our Hamiltonian-structured neural ODE networks from a dataset of state-control trajectories under different disturbance realizations. Our Hamiltonian-based neural ODE networks for dynamics learning are verified with various mobile robot platforms.

Chapter 5 describes how we integrate our sparse probabilistic occupancy maps and our learned Hamiltonian dynamics into autonomous navigation tasks. We first propose efficient collision-checking methods associated with our proposed map representations in Chapter 3 for

line segments and general curves, representing robot trajectories. We develop energy shaping and damping injection control for the learned, potentially under-actuated Hamiltonian dynamics to enable a unified approach for stabilization and trajectory tracking with various platforms. In the presence of disturbance or dynamics changes, we develop an adaptation law to estimate the disturbances online based on the geometric tracking errors and the learned disturbance features in Chapter 4, and compensate for them by the energy-based tracking controller. We finally verify the effectiveness of our collision checking methods with our map representations, and our control design with our learned dynamics in autonomous navigation tasks with ground and aerial robot platforms.

The dissertation closes with conclusions and potential future directions in **Chapter 6**.

Acknowledgements

Chapter 1, in part, is a reprint of the material as it appears in T. Duong, M. Yip, N. Atanasov, "*Autonomous Navigation in Unknown Environments with Sparse Bayesian Kernel-based Occupancy Mapping*", IEEE Transactions on Robotics (T-RO), vol. 38, no. 6, pp. 3694-3712, 2022, in T. Duong, N. Das, M. Yip, N. Atanasov, "*Autonomous Navigation in Unknown Environments using Sparse Kernel-based Occupancy Mapping*", International Conference on Robotics and Automation (ICRA), pp. 9666-9672, 2020, in T. Duong, N. Atanasov, "*Hamiltonian-based Neural ODE Networks on the SE(3) Manifold For Dynamics Learning and Control*", Robotics: Science and Systems (RSS), 2021, in T. Duong, N. Atanasov, "*Physics-guided Learning-based Adaptive Control on the SE(3) Manifold*", Physical Reasoning and Inductive Biases for the Real World Workshop, 2021, and in T. Duong, N. Atanasov, "*Adaptive Control of SE(3) Hamiltonian Dynamics with Learned Disturbance Features*", IEEE Control Systems Letters (L-CSS), vol. 6, pp. 2773-2778, 2022. Chapter 1, in part, has been submitted for publication of the material as it may appear in T. Duong, A. Altawaitan, J. Stanley, N. Atanasov, "*Port-Hamiltonian-based Neural ODE Networks on Lie Groups For Robot Dynamics Learning and Control*", Transactions on Robotics, 2024. The dissertation author was the primary investigator and author of these papers.

Chapter 2

Background

This chapter presents the background for further derivation of our work in the next chapters, including matrix Lie groups, Hamiltonian formulation of robot dynamics, and machine learning techniques such as neural ODE networks, kernel perceptron, and relevance vector machine classifiers.

2.1 Matrix Lie Groups

In this section, we cover the background needed to define Hamiltonian dynamics on a Lie group. Please refer to [62, 91, 116] for a more detailed overview of matrix Lie groups.

Definition 1 (Dot Product)**.** The dot product $\langle \cdot, \cdot \rangle$ between two matrices $\boldsymbol{\xi}$ and $\boldsymbol{\psi}$ in $\mathbf{R}^{n \times m}$ is defined as:

$$\langle \boldsymbol{\xi}, \boldsymbol{\psi} \rangle = \mathrm{tr}(\boldsymbol{\xi}^\top \boldsymbol{\psi}). \tag{2.1}$$

Definition 2 (General Linear Group [62])**.** The general linear group $\mathsf{GL}(n, \mathbb{R})$ is the group of $n \times n$ invertible real matrices.

Definition 3 (Matrix Lie Group [62])**.** A matrix Lie group G is a subgroup of $\mathsf{GL}(n, \mathbb{R})$ with identity element $\mathbf{e}$ such that if any sequence of matrices $\{A_n\}_{n=0}^{\infty}$ in G converges to a matrix A, then either A is in G or A is not invertible. A matrix Lie group is also a smooth embedded submanifold on $\mathbb{R}^{n \times n}$.

Definition 4 (Tangent Space and Bundle)**.** The tangent space $\mathsf{T}_{\mathbf{q}}\mathsf{G}$ is the set of all tangent vectors $\boldsymbol{\xi}$ to the manifold G at $\mathbf{q}$. The tangent bundle TG is the set of all the pairs $(\mathbf{q}, \boldsymbol{\xi})$ with $\mathbf{q} \in \mathsf{G}$ and $\boldsymbol{\xi} \in \mathsf{T}_{\mathbf{q}}\mathsf{G}$.

Definition 5 (Lie Algebra and Lie Bracket)**.** A Lie algebra is a vector space $\mathfrak{g}$, equipped with a Lie bracket operator $[\cdot,\cdot]:\mathfrak{g}\times\mathfrak{g}\to\mathfrak{g}$ that satisfies:

$$\begin{aligned}
&\text{bilinearity:}\quad [a\boldsymbol{\xi}_1+b\boldsymbol{\xi}_2,\boldsymbol{\xi}_3]=a[\boldsymbol{\xi}_1,\boldsymbol{\xi}_3]+b[\boldsymbol{\xi}_2,\boldsymbol{\xi}_3],\\
&\qquad\qquad [\boldsymbol{\xi}_3,a\boldsymbol{\xi}_1+b\boldsymbol{\xi}_2]=a[\boldsymbol{\xi}_3,\boldsymbol{\xi}_1]+b[\boldsymbol{\xi}_3,\boldsymbol{\xi}_2],\\
&\text{skew-symmetry:}\quad [\boldsymbol{\xi}_1,\boldsymbol{\xi}_2]=-[\boldsymbol{\xi}_2,\boldsymbol{\xi}_1],\\
&\text{Jacobi identity:}\\
&\qquad\qquad [\boldsymbol{\xi}_1,[\boldsymbol{\xi}_2,\boldsymbol{\xi}_3]]+[\boldsymbol{\xi}_2,[\boldsymbol{\xi}_3,\boldsymbol{\xi}_1]]+[\boldsymbol{\xi}_3,[\boldsymbol{\xi}_1,\boldsymbol{\xi}_2]]=0.
\end{aligned}$$

Every matrix Lie group G is associated with a Lie algebra $\mathfrak{g}$, which is the tangent space at the identity element $\mathsf{T}_{\mathbf{e}}\mathsf{G}$. An element $\mathbf{q}\in\mathsf{G}$ is linked with an element $\boldsymbol{\xi}\in\mathfrak{g}$ via the exponential map $\exp_{\mathsf{G}}:\mathfrak{g}\to\mathsf{G}$ and the logarithm map $\log_{\mathsf{G}}:\mathsf{G}\to\mathfrak{g}$ [62]. Since tangent spaces of G, and in particular the Lie algebra $\mathfrak{g}$, are isomorphic to Euclidean space, we can define a linear mapping $(\cdot)^{\wedge}:\mathbb{R}^n\to\mathfrak{g}$ and its inverse $(\cdot)^{\vee}:\mathfrak{g}\to\mathbb{R}^n$, where n is the dimension of G. Thus, we can map between G and $\mathbb{R}^n$ using the compositions:

$$\exp_{\mathsf{G}}^{\wedge}=\exp_{\mathsf{G}}\circ{}^{\wedge},\quad \log_{\mathsf{G}}^{\vee}={}^{\vee}\circ\log_{\mathsf{G}}. \tag{2.2}$$

Definition 6 (Left Translation and Invariant Vectors)**.** The left translation $\mathsf{L}_{\mathbf{q}}:\mathsf{G}\to\mathsf{G}$ with $\mathbf{q}\in\mathsf{G}$ is defined as:

$$\mathsf{L}_{\mathbf{q}}(\mathbf{h})=\mathbf{q}\mathbf{h}. \tag{2.3}$$

The left-invariant vector $\mathsf{T}_{\mathbf{e}}\mathsf{L}_{\mathbf{q}}(\boldsymbol{\xi})$ is defined as the derivative of the left translation $\mathsf{L}_{\mathbf{q}}$ at $\mathbf{h}=\mathbf{e}$ in the direction of $\boldsymbol{\xi}$. This vector describes the kinematics of the Lie group, which relates the velocity $\boldsymbol{\xi}\in\mathfrak{g}$ to the change $\dot{\mathbf{q}}\in\mathsf{T}_{\mathbf{q}}\mathsf{G}$ of coordinates $\mathbf{q}$:

$$\dot{\mathbf{q}}=\mathsf{T}_{\mathbf{e}}\mathsf{L}_{\mathbf{q}}(\boldsymbol{\xi})=\mathbf{q}\boldsymbol{\xi}. \tag{2.4}$$

The dual map $\mathsf{T}_{\mathbf{e}}^{*}\mathsf{L}_{\mathbf{q}}$ of $\mathsf{T}_{\mathbf{e}}\mathsf{L}_{\mathbf{q}}$ satisfies

$$\langle\boldsymbol{\xi},\mathsf{T}_{\mathbf{e}}^{*}\mathsf{L}_{\mathbf{q}}(\boldsymbol{\eta})\rangle=\langle\mathsf{T}_{\mathbf{e}}\mathsf{L}_{\mathbf{q}}(\boldsymbol{\xi}),\boldsymbol{\eta}\rangle, \tag{2.5}$$

for any $\boldsymbol{\eta} \in \mathfrak{g}^*$ and $\boldsymbol{\xi} \in \mathfrak{g}$.

Definition 7 (Adjoint Operator)**.** The adjoint $\mathsf{Ad}_{\mathbf{q}} : \mathfrak{g} \to \mathfrak{g}$ is defined as:

$$\mathsf{Ad}_{\mathbf{q}}(\boldsymbol{\psi}) = \mathbf{q}\boldsymbol{\psi}\mathbf{q}^{-1}. \tag{2.6}$$

The algebra adjoint $\mathsf{ad}_{\boldsymbol{\xi}} : \mathfrak{g} \to \mathfrak{g}$ is the directional derivative of $\mathsf{Ad}_{\mathbf{q}}$ at $\mathbf{q} = \mathbf{e}$ in the direction of $\boldsymbol{\xi} \in \mathfrak{g}$:

$$\mathsf{ad}_{\boldsymbol{\xi}}(\boldsymbol{\psi}) = \frac{d}{dt}\mathsf{Ad}_{\exp_{\mathsf{G}}(t\boldsymbol{\xi})}(\boldsymbol{\psi})\bigg|_{t=0} = [\boldsymbol{\xi}, \boldsymbol{\psi}]. \tag{2.7}$$

Definition 8 (Cotangent Space and Bundle)**.** The dual space of the tangent space $\mathsf{T}_{\mathbf{q}}\mathsf{G}$, i.e., the space of all linear functionals from $\mathsf{T}_{\mathbf{q}}\mathsf{G}$ to $\mathbb{R}$, is called the cotangent space $\mathsf{T}^*_{\mathbf{q}}\mathsf{G}$. At the identity $\mathbf{e}$, the cotangent space of the Lie algebra $\mathfrak{g} = \mathsf{T}_{\mathbf{e}}\mathsf{G}$ is denoted $\mathfrak{g}^*$. The cotangent bundle $\mathsf{T}^*\mathsf{G}$ is the set of all the pairs $(\mathbf{q}, \mathfrak{p})$ with $\mathbf{q} \in \mathsf{G}$ and $\mathfrak{p} \in \mathsf{T}^*_{\mathbf{q}}\mathsf{G}$.

Definition 9 (Coadjoint Operator)**.** The coadjoint $\mathsf{Ad}^*_{\mathbf{q}} : \mathfrak{g}^* \to \mathfrak{g}^*$ is defined as: $\left\langle \mathsf{Ad}^*_{\mathbf{q}}(\boldsymbol{\varphi}), \boldsymbol{\psi} \right\rangle = \langle \boldsymbol{\varphi}, \mathsf{Ad}_{\mathbf{q}}(\boldsymbol{\psi}) \rangle$. The algebra coadjoint $\mathsf{ad}^*_{\boldsymbol{\xi}} : \mathfrak{g}^* \to \mathfrak{g}^*$ is the dual map of $\mathsf{ad}_{\boldsymbol{\xi}}$, satisfying:

$$\left\langle \mathsf{ad}^*_{\boldsymbol{\xi}}(\boldsymbol{\varphi}), \boldsymbol{\psi} \right\rangle = \langle \boldsymbol{\varphi}, \mathsf{ad}_{\boldsymbol{\xi}}(\boldsymbol{\psi}) \rangle .$$

The next section describes the $SE(3)$ Lie group to illustrate the definitions above. The $SE(3)$ Lie group is used to represent the position and orientation of a rigid body.

2.1.1 Example: SE(3) Manifold

Consider a fixed world inertial frame of reference and a rigid body with a body-fixed frame attached to its center of mass. The pose of the body-fixed frame in the world frame is determined by the position $\mathbf{p} = [x, y, z]^\top \in \mathbb{R}^3$ of the center of mass and the orientation of the body-fixed frame's coordinate axes:

$$\mathbf{R} = \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 \end{bmatrix}^\top \in SO(3), \tag{2.8}$$

where $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \in \mathbb{R}^3$ are the rows of the rotation matrix $\mathbf{R}$. A rotation matrix is an element of the special orthogonal group:

$$SO(3) = \left\{ \mathbf{R} \in \mathbb{R}^{3\times 3} : \mathbf{R}^\top \mathbf{R} = \mathbf{I}, \det(\mathbf{R}) = 1 \right\}. \tag{2.9}$$

The rigid-body position and orientation can be combined in a single pose matrix $\mathbf{q} \in SE(3)$, which is an element of the special Euclidean group:

$$SE(3) = \left\{ \mathbf{q} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}^\top & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} : \mathbf{R} \in SO(3), \mathbf{p} \in \mathbb{R}^3 \right\}. \tag{2.10}$$

The kinematic equations of motion of the rigid body are determined by the linear velocity $\mathbf{v} \in \mathbb{R}^3$ and angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$ of the body-fixed frame with respect to the world frame, expressed in body-frame coordinates. The generalized velocity $\boldsymbol{\zeta} = [\mathbf{v}^\top, \boldsymbol{\omega}^\top]^\top \in \mathbb{R}^6$ determines the rate of change of the rigid-body pose according to the $SE(3)$ kinematics:

$$\dot{\mathbf{q}} = \mathsf{T}_{\mathbf{e}} \mathsf{L}_{\mathbf{q}}(\boldsymbol{\xi}) = \mathbf{q}\boldsymbol{\xi} = \mathbf{q}\hat{\boldsymbol{\zeta}} = \begin{bmatrix} \dot{\mathbf{R}} & \dot{\mathbf{p}} \\ \mathbf{0}^\top & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{R}\hat{\boldsymbol{\omega}} & \mathbf{R}\mathbf{v} \\ \mathbf{0}^\top & 0 \end{bmatrix}, \tag{2.11}$$

where we overload $\hat{\cdot}$ to denote the mapping from a vector $\boldsymbol{\zeta} \in \mathbb{R}^6$ to a 4×4 twist matrix:

$$\boldsymbol{\xi} = \hat{\boldsymbol{\zeta}} = \begin{bmatrix} \hat{\boldsymbol{\omega}} & \mathbf{v} \\ \mathbf{0}^\top & 0 \end{bmatrix}, \tag{2.12}$$

in the Lie algebra $\mathfrak{se}(3)$ of $SE(3)$ and from a vector $\boldsymbol{\omega} \in \mathbb{R}^3$ to a 3×3 skew-symmetric matrix $\hat{\boldsymbol{\omega}}$ in the Lie algebra $\mathfrak{so}(3)$ of $SO(3)$:

$$\hat{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}. \tag{2.13}$$

From Eq. (2.11), we have $\dot{\mathbf{R}} = \begin{bmatrix} \dot{\mathbf{r}}_1 & \dot{\mathbf{r}}_2 & \dot{\mathbf{r}}_3 \end{bmatrix}^\top = \mathbf{R}\hat{\boldsymbol{\omega}}$ or $\dot{\mathbf{r}}_i = \mathbf{r}_i \times \boldsymbol{\omega} = \hat{\mathbf{r}}_i \boldsymbol{\omega}$ for $i = 1, 2, 3$.

On the Lie algebra $\mathfrak{se}(3)$, the algebra adjoint $\mathsf{ad}_{\boldsymbol{\xi}}(\boldsymbol{\psi})$ is defined as:

$$\mathsf{ad}_{\boldsymbol{\xi}}(\boldsymbol{\psi}) = [\boldsymbol{\xi}, \boldsymbol{\psi}] = \boldsymbol{\xi}\boldsymbol{\psi} - \boldsymbol{\psi}\boldsymbol{\xi}. \tag{2.14}$$

The dual map of $\mathsf{ad}_{\boldsymbol{\xi}}$ satisfies: $\left\langle \mathsf{ad}^*_{\boldsymbol{\xi}}(\boldsymbol{\varphi}), \boldsymbol{\psi} \right\rangle = \langle \boldsymbol{\varphi}, \mathsf{ad}_{\boldsymbol{\xi}}(\boldsymbol{\psi}) \rangle$, leading to

$$\begin{aligned} \left\langle \mathsf{ad}^*_{\boldsymbol{\xi}}(\boldsymbol{\varphi}), \boldsymbol{\psi} \right\rangle &= \langle \boldsymbol{\varphi}, [\boldsymbol{\xi}, \boldsymbol{\psi}] \rangle , \\ &= \langle \boldsymbol{\varphi}, \boldsymbol{\xi}\boldsymbol{\psi} \rangle - \langle \boldsymbol{\varphi}, \boldsymbol{\psi}\boldsymbol{\xi} \rangle , \\ &= \left\langle \boldsymbol{\xi}^\top \boldsymbol{\varphi} - \boldsymbol{\varphi}\boldsymbol{\xi}^\top, \boldsymbol{\psi} \right\rangle . \end{aligned} \tag{2.15}$$

Therefore, we obtain $\mathsf{ad}^*_{\boldsymbol{\xi}}(\mathfrak{p}) = \mathsf{P}_{\mathfrak{g}^*}([\boldsymbol{\xi}^\top, \mathfrak{p}])$, where $\mathsf{P}_{\mathfrak{g}^*}$ is an orthogonal projector on $\mathfrak{g}^*$ [15, Def. 3.60]. On SE(3), we have $\mathsf{P}_{\mathfrak{g}^*}\left(\begin{bmatrix} \mathbf{A} & \mathbf{a} \\ \mathbf{b}^\top & c \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{2}(\mathbf{A} - \mathbf{A}^\top) & \mathbf{a} \\ \mathbf{0}^\top & 0 \end{bmatrix}$ [3] with $\mathbf{A} \in \mathbb{R}^{3\times 3}$, $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$, and $c \in \mathbb{R}$ leading to $\mathsf{ad}^*_{\boldsymbol{\xi}}(\mathfrak{p}) = \begin{bmatrix} \left(\hat{\mathbf{a}}\boldsymbol{\omega} + \frac{1}{2}\hat{\mathbf{b}}\mathbf{v}\right)^\wedge & \hat{\mathbf{b}}\boldsymbol{\omega} \\ \mathbf{0}^\top & 0 \end{bmatrix}$.

The dual map $\mathsf{T}^*_{\mathsf{e}}\mathsf{L}_{\mathsf{q}}(\boldsymbol{\eta})$ is defined as $\langle \boldsymbol{\xi}, \mathsf{T}^*_{\mathsf{e}}\mathsf{L}_{\mathsf{q}}(\boldsymbol{\eta}) \rangle = \langle \mathsf{T}_{\mathsf{e}}\mathsf{L}_{\mathsf{q}}(\boldsymbol{\xi}), \boldsymbol{\eta} \rangle$, where the matrix $\boldsymbol{\eta}$ is defined as $\boldsymbol{\eta} = \begin{bmatrix} \boldsymbol{\eta}_{\mathbf{R}} & \boldsymbol{\eta}_{\mathbf{p}} \\ \mathbf{0}^\top & 0 \end{bmatrix}$, with $\boldsymbol{\eta}_{\mathbf{R}} = \begin{bmatrix} \boldsymbol{\eta}_{\mathbf{r}_1} & \boldsymbol{\eta}_{\mathbf{r}_2} & \boldsymbol{\eta}_{\mathbf{r}_3} \end{bmatrix}^\top$. We have:

$$\begin{aligned} \langle \mathsf{T}_{\mathsf{e}}\mathsf{L}_{\mathsf{q}}(\boldsymbol{\xi}), \boldsymbol{\eta} \rangle &= \left\langle \begin{bmatrix} \dot{\mathbf{R}} & \dot{\mathbf{p}} \\ \mathbf{0}^\top & 0 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\eta}_{\mathbf{R}} & \boldsymbol{\eta}_{\mathbf{p}} \\ \mathbf{0}^\top & 0 \end{bmatrix} \right\rangle = \sum_{i=1}^{3} \dot{\mathbf{r}}_i^\top \boldsymbol{\eta}_{\mathbf{r}_i} + \left\langle \dot{\mathbf{p}}, \boldsymbol{\eta}_{\mathbf{p}} \right\rangle , \\ &= \sum_{i=1}^{3} (\hat{\mathbf{r}}_i \boldsymbol{\omega})^\top \boldsymbol{\eta}_{\mathbf{r}_i} + \left\langle \mathbf{R}\mathbf{v}, \boldsymbol{\eta}_{\mathbf{p}} \right\rangle = \mathbf{v}^\top \mathbf{R}^\top \boldsymbol{\eta}_{\mathbf{p}} - \sum_{i=1}^{3} \boldsymbol{\omega}^\top \hat{\mathbf{r}}_i \boldsymbol{\eta}_{\mathbf{r}_i}, \\ &= \left\langle \mathbf{v}, \mathbf{R}^\top \boldsymbol{\eta}_{\mathbf{p}} \right\rangle - \sum_{i=1}^{3} \boldsymbol{\omega}^\top (\mathbf{r}_i \times \boldsymbol{\eta}_{\mathbf{r}_i}), \\ &= \left\langle \boldsymbol{\omega}, -\sum_{i=1}^{3} \mathbf{r}_i \times \boldsymbol{\eta}_{\mathbf{r}_i} \right\rangle + \left\langle \mathbf{v}, \mathbf{R}^\top \boldsymbol{\eta}_{\mathbf{p}} \right\rangle . \end{aligned} \tag{2.16}$$

Letting $\mathsf{T}_{\mathbf{e}}^{*}\mathsf{L}_{\mathbf{q}}(\boldsymbol{\eta}) = \begin{bmatrix} \hat{\mathbf{a}} & \mathbf{b} \\ \mathbf{0}^{\top} & 0 \end{bmatrix}$, we have:

$$\begin{aligned} \langle \boldsymbol{\xi}, \mathsf{T}_{\mathbf{e}}^{*}\mathsf{L}_{\mathbf{q}}(\boldsymbol{\eta}) \rangle &= \left\langle \begin{bmatrix} \hat{\boldsymbol{\omega}} & \mathbf{v} \\ \mathbf{0}^{\top} & 0 \end{bmatrix}, \begin{bmatrix} \hat{\mathbf{a}} & \mathbf{b} \\ \mathbf{0}^{\top} & 0 \end{bmatrix} \right\rangle, \\ &= \langle \hat{\boldsymbol{\omega}}, \hat{\mathbf{a}} \rangle + \langle \mathbf{v}, \mathbf{b} \rangle, \\ &= 2 \langle \boldsymbol{\omega}, \mathbf{a} \rangle + \langle \mathbf{v}, \mathbf{b} \rangle. \end{aligned} \tag{2.17}$$

Comparing Eq. (2.16) and Eq. (2.17), we derive the closed-form solution of $\mathsf{T}_{\mathbf{e}}^{*}\mathsf{L}_{\mathbf{q}}(\boldsymbol{\eta})$ as:

$$\mathsf{T}_{\mathbf{e}}^{*}\mathsf{L}_{\mathbf{q}}(\boldsymbol{\eta}) = \begin{bmatrix} -\frac{1}{2}\left(\sum_{i=1}^{3} \mathbf{r}_i \times \boldsymbol{\eta}_{\mathbf{r}_i}\right)^{\wedge} & \mathbf{R}^{\top}\boldsymbol{\eta}_{\mathbf{p}} \\ \mathbf{0}^{\top} & 0 \end{bmatrix}. \tag{2.18}$$

Please refer to [6] for an excellent introduction to the use of $SE(3)$ in robot state estimation problems.

2.2 Hamiltonian Dynamics

There are three predominant formulations of classical mechanics [69] for describing the motion of macroscopic objects: Newtonian, Lagrangian, and Hamiltonian. Newtonian mechanics models the dynamics of mobile objects using forces and Cartesian coordinates according to Newton's laws of motion. Lagrangian and Hamiltonian mechanics use generalized coordinates and energy in their formulations, which simplifies the equations of motion and reveals conserved quantities and symmetries. In this section, we present a brief summary on Hamiltonian dynamics on a state space, from a simple vector space $\mathbb{R}^n$ to a Lie group. A more general port-Hamiltonian formulation is described as well, with an example of (port-)Hamiltonian dynamics on the $SE(3)$ manifolds.

2.2.1 Hamiltonian Dynamics on Vector Space

Both Lagrangian and Hamiltonian mechanics consider generalized coordinates $\mathbf{q} \in \mathbb{R}^n$ and velocity $\dot{\mathbf{q}} \in \mathbb{R}^n$, and defines a Lagrangian function $\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}})$ as the difference between kinetic

energy $\frac{1}{2}\dot{\mathbf{q}}^\top \mathbf{M}(\mathbf{q})\dot{\mathbf{q}}$ and the potential energy $V(\mathbf{q})$:

$$\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^\top \mathbf{M}(\mathbf{q})\dot{\mathbf{q}} - V(\mathbf{q}), \tag{2.19}$$

where the symmetric positive-definite matrix $\mathbf{M}(\mathbf{q}) \in \mathbb{S}^{n\times n}_{\succ 0}$ represents the generalized mass of the system. Starting from the Lagrangian formulation, Hamiltonian mechanics expresses the system dynamics in terms of the generalized coordinates $\mathbf{q} \in \mathbb{R}^n$ and generalized momenta $\mathfrak{p} \in \mathbb{R}^n$, defined as:

$$\mathfrak{p} = \frac{\partial \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}})}{\partial \dot{\mathbf{q}}} = \mathbf{M}(\mathbf{q})\dot{\mathbf{q}}. \tag{2.20}$$

Instead of the Lagrangian function, a Hamiltonian function $\mathcal{H}(\mathbf{q}, \mathfrak{p})$, representing the total energy of the dynamical system, is obtained by the Legendre transform of $\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}})$:

$$\mathcal{H}(\mathbf{q}, \mathfrak{p}) = \mathfrak{p}^\top \dot{\mathbf{q}} - \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\mathfrak{p}^\top \mathbf{M}(\mathbf{q})^{-1}\mathfrak{p} + V(\mathbf{q}). \tag{2.21}$$

The Hamiltonian characterizes the system dynamics according to the equations:

$$\dot{\mathbf{q}} = \frac{\partial \mathcal{H}}{\partial \mathfrak{p}}, \quad \dot{\mathfrak{p}} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}} + \mathbf{B}(\mathbf{q})\mathbf{u}, \tag{2.22}$$

where $\mathbf{u} \in \mathbb{R}^n$ is an external affine control input with coefficient matrix $\mathbf{B}(\mathbf{q}) \in \mathbb{R}^{n\times n}$. Without external input, i.e., $\mathbf{u} = \mathbf{0}$, (2.22) ensures that the total energy of the system is conserved, $\frac{d}{dt}\mathcal{H}(\mathbf{q}, \mathfrak{p}) = \frac{\partial \mathcal{H}}{\partial \mathbf{q}}\dot{\mathbf{q}} + \frac{\partial \mathcal{H}}{\partial \mathfrak{p}}\dot{\mathfrak{p}} = 0$.

2.2.2 Hamiltonian Dynamics on Matrix Lie Groups

In this section, we describe Hamilton's equations of motion on a matrix Lie group [116, 91]. Our neural network architecture design in Section 4.2 is based on Lie group Hamiltonian dynamics, which enables guarantees for both kinematic constraint satisfaction and energy conservation. Consider a system with generalized coordinates $\mathbf{q}$ in a matrix Lie group G and generalized velocity $\dot{\mathbf{q}} \in \mathsf{T}_{\mathbf{q}}\mathsf{G}$. The dynamics of the state $\mathbf{s} = (\mathbf{q}, \dot{\mathbf{q}}) \in \mathsf{TG}$ satisfy:

$$\dot{\mathbf{q}} = \mathsf{T}_{\mathsf{e}}\mathsf{L}_{\mathbf{q}}(\boldsymbol{\xi}) = \mathbf{q}\boldsymbol{\xi}, \tag{2.23}$$

where $\boldsymbol{\xi}$ is a element in the Lie algebra $\mathfrak{g}$.

The Lagrangian on a Lie group $\mathcal{L} : \mathsf{G} \times \mathfrak{g} \to \mathbb{R}$ is defined as the difference between the kinetic energy $\mathcal{T} : \mathsf{G} \times \mathfrak{g} \to \mathbb{R}$ and the potential energy $\mathcal{V} : \mathsf{G} \to \mathbb{R}$:

$$\mathcal{L}(\mathbf{q}, \boldsymbol{\xi}) = \mathcal{T}(\mathbf{q}, \boldsymbol{\xi}) - \mathcal{V}(\mathbf{q}). \tag{2.24}$$

The Hamiltonian is obtained using a Legendre transformation:

$$\mathcal{H}(\mathbf{q}, \mathfrak{p}) = \mathfrak{p} \cdot \boldsymbol{\xi} - \mathcal{L}(\mathbf{q}, \boldsymbol{\xi}), \tag{2.25}$$

where the momentum $\mathfrak{p}$ is defined as:

$$\mathfrak{p} = \frac{\partial \mathcal{L}(\mathbf{q}, \boldsymbol{\xi})}{\partial \boldsymbol{\xi}}. \tag{2.26}$$

The state $(\mathbf{q}, \mathfrak{p}) \in \mathsf{T}^*\mathsf{G}$ evolves according to the Hamiltonian dynamics [91] as:

$$\dot{\mathbf{q}} = \mathsf{T}_{\mathbf{e}}\mathsf{L}_{\mathbf{q}} \left(\frac{\partial \mathcal{H}(\mathbf{q}, \mathfrak{p})}{\partial \mathfrak{p}} \right), \tag{2.27a}$$

$$\dot{\mathfrak{p}} = \mathsf{ad}^*_{\boldsymbol{\xi}}(\mathfrak{p}) - \mathsf{T}^*_{\mathbf{e}}\mathsf{L}_{\mathbf{q}} \left(\frac{\partial \mathcal{H}(\mathbf{q}, \mathfrak{p})}{\partial \mathbf{q}} \right) + \mathbf{B}(\mathbf{q})\mathbf{u}. \tag{2.27b}$$

By comparing Eq. (2.23) and Eq. (2.27), we have:

$$\boldsymbol{\xi} = \frac{\partial \mathcal{H}(\mathbf{q}, \mathfrak{p})}{\partial \mathfrak{p}}. \tag{2.28}$$

Let $\boldsymbol{\eta} = \frac{\partial \mathcal{H}(\mathbf{q}, \mathfrak{p})}{\partial \mathbf{q}}$. When there is no control input, i.e., $\mathbf{u} = 0$, the conservation of energy is guaranteed as:

$$\frac{d\mathcal{H}(\mathbf{q}, \mathfrak{p})}{dt} = \langle \boldsymbol{\eta}, \dot{\mathbf{q}} \rangle + \langle \boldsymbol{\xi}, \dot{\mathfrak{p}} \rangle = \langle \boldsymbol{\eta}, \mathsf{T}_{\mathbf{e}}\mathsf{L}_{\mathbf{q}} (\boldsymbol{\xi}) \rangle - \langle \boldsymbol{\xi}, \mathsf{T}^*_{\mathbf{e}}\mathsf{L}_{\mathbf{q}} (\boldsymbol{\eta}) \rangle + \left\langle \boldsymbol{\xi}, \mathsf{ad}^*_{\boldsymbol{\xi}}(\mathfrak{p}) \right\rangle = 0, \tag{2.29}$$

because of Eq. (2.5) and, by definition,

$$\left\langle \boldsymbol{\xi}, \mathsf{ad}^*_{\boldsymbol{\xi}}(\mathfrak{p}) \right\rangle = \langle \mathsf{ad}_{\boldsymbol{\xi}}(\boldsymbol{\xi}), \mathfrak{p} \rangle = \langle [\boldsymbol{\xi}, \boldsymbol{\xi}], \mathfrak{p} \rangle = 0. \tag{2.30}$$

2.2.3 Port-Hamiltonian Dynamics

The notion of energy in dynamical systems is shared across multiple domains, including mechanical, electrical, and thermal. A port-Hamiltonian generalization [170] of Hamiltonian mechanics is used to model systems with energy-storing elements (e.g., kinetic and potential energy), energy-dissipating elements (e.g., friction or resistance), and external energy sources (e.g., control inputs), connected via energy ports. An input-state-output port-Hamiltonian system has the form:

$$\begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\mathbf{p}} \end{bmatrix} = (\mathcal{J}(\mathbf{q}, \mathbf{p}) - \mathcal{R}(\mathbf{q}, \mathbf{p})) \begin{bmatrix} \frac{\partial \mathcal{H}}{\partial \mathbf{q}} \\ \frac{\partial \mathcal{H}}{\partial \mathbf{p}} \end{bmatrix} + \mathcal{G}(\mathbf{q}, \mathbf{p})\mathbf{u}, \tag{2.31}$$

where $\mathcal{J}(\mathbf{q}, \mathbf{p})$ is a skew-symmetric interconnection matrix, representing the energy-storing elements, $\mathcal{R}(\mathbf{q}, \mathbf{p}) \succeq 0$ is a positive semi-definite dissipation matrix, representing the energy-dissipating elements, and $\mathcal{G}(\mathbf{q}, \mathbf{p})$ is an input matrix such that $\mathcal{G}(\mathbf{q}, \mathbf{p})\mathbf{u}$ represents the external energy sources. In the absence of energy-dissipating elements and external energy sources, the skew-symmetry of $\mathcal{J}(\mathbf{q}, \mathbf{p})$ guarantees the energy conservation of the system.

To model energy dissipating elements such as friction or drag forces, we reformulate the Hamiltonian dynamics on a matrix Lie group (2.27) in Port-Hamiltonian form (2.31). Such elements are often modeled [44] as a linear transformation $\mathbf{D}(\mathbf{q}, \mathbf{p}) \succeq 0$ of the velocity $\boldsymbol{\xi}$ and only affect the generalized momenta $\mathbf{p}$, i.e.,

$$\mathcal{R}(\mathbf{q}, \mathbf{p}) = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}(\mathbf{q}, \mathbf{p}) \end{bmatrix}. \tag{2.32}$$

The Hamiltonian dynamics on Lie groups (2.27) is a special case of (2.31), where the dissipation matrix is $\mathbf{D}(\mathbf{q}, \mathbf{p}) = \mathbf{0}$, the input matrix is $\mathcal{G}(\mathbf{q}, \mathbf{p}) = \begin{bmatrix} \mathbf{0}^\top & \mathbf{B}(\mathbf{q})^\top \end{bmatrix}^\top$ and the interconnection matrix $\mathcal{J}(\mathbf{q}, \mathbf{p})$ can be obtained by rearranging (2.27) with $\mathbf{u} = 0$ and is guaranteed to be skew-symmetric due to the energy conservation (2.29). The Hamiltonian dynamics on $\mathbb{R}^n$ (2.22) are

also a special case of port-Hamiltonian dynamics (2.31) with:

$$\mathcal{J}(\mathbf{q},\mathfrak{p}) = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{I} & \mathbf{0} \end{bmatrix}, \quad \mathcal{G}(\mathbf{q},\mathfrak{p}) = \begin{bmatrix} \mathbf{0} \\ \mathbf{B}(\mathbf{q}) \end{bmatrix}. \tag{2.33}$$

2.2.4 Example: Hamiltonian Dynamics on the SE(3) Manifold

In this section, we consider the generalized coordinate $\mathbf{q}$ of a mobile robot consisting of its position $\mathbf{p} \in \mathbb{R}^3$ and orientation $\mathbf{R} \in SO(3)$. Let $\mathbf{q} = (\mathbf{p}, \mathbf{R})$ be the generalized coordinates and $\boldsymbol{\zeta} = (\mathbf{v}, \boldsymbol{\omega}) \in \mathbb{R}^6$ be the generalized velocity, consisting of the body-frame linear velocity $\mathbf{v} \in \mathbb{R}^3$ and the body-frame angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$. The coordinate $\mathbf{q}$ evolves on the Lie group $SE(3)$ while the generalized velocity satisfies $\dot{\mathbf{q}} = \mathbf{q}\boldsymbol{\xi}$, where $\boldsymbol{\xi} = \hat{\boldsymbol{\zeta}}$ is a twist matrix in $\mathfrak{se}(3)$, as shown in Eq. (2.11).

The isomorphism between $\mathfrak{se}(3)$ and $\mathbb{R}^6$ via (2.11) simplifies the Hamiltonian (2.27) and its port-Hamiltonian formulation (2.31) as follows. The Lagrangian function on $SE(3)$ can be expressed in terms of $\mathbf{q}$ and $\boldsymbol{\zeta}$, instead of $\mathbf{q}$ and $\boldsymbol{\xi}$:

$$\mathcal{L}(\mathbf{q},\boldsymbol{\zeta}) = \frac{1}{2}\boldsymbol{\zeta}^\top \mathbf{M}(\mathbf{q})\boldsymbol{\zeta} - \mathcal{V}(\mathbf{q}). \tag{2.34}$$

The generalized mass matrix has a block-diagonal form when the body frame is attached to the center of mass [91]:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} \mathbf{M}_{\mathbf{v}}(\mathbf{q}) & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_{\boldsymbol{\omega}}(\mathbf{q}) \end{bmatrix} \in \mathbb{S}^{6\times 6}_{\succ 0}, \tag{2.35}$$

where $\mathbf{M}_{\mathbf{v}}(\mathbf{q}), \mathbf{M}_{\boldsymbol{\omega}}(\mathbf{q}) \in \mathbb{S}^{3\times 3}_{\succ 0}$. The generalized momenta are defined, as before, via the partial derivative of the Lagrangian with respect to the twist:

$$\mathfrak{p} = \begin{bmatrix} \mathfrak{p}_{\mathbf{v}} \\ \mathfrak{p}_{\boldsymbol{\omega}} \end{bmatrix} = \frac{\partial \mathcal{L}(\mathbf{q},\boldsymbol{\zeta})}{\partial \boldsymbol{\zeta}} = \mathbf{M}(\mathbf{q})\boldsymbol{\zeta} \in \mathbb{R}^6. \tag{2.36}$$

The Hamiltonian function of the system becomes:

$$\mathcal{H}(\mathfrak{q},\mathfrak{p}) = \mathfrak{p}\cdot\boldsymbol{\zeta} - \mathcal{L}(\mathfrak{q},\boldsymbol{\zeta}) = \frac{1}{2}\mathfrak{p}^\top \mathbf{M}^{-1}(\mathfrak{q})\mathfrak{p} + \mathcal{V}(\mathfrak{q}). \tag{2.37}$$

By vectorizing the generalized coordinates $\mathfrak{q} = [\mathbf{p}^\top \quad \mathbf{r}_1^\top \quad \mathbf{r}_2^\top \quad \mathbf{r}_3^\top]^\top$, the Hamiltonian dynamics on $SE(3)$ can be described in port-Hamiltonian form (2.31) [91, 47, 138] with interconnection matrix:

$$\mathcal{J}(\mathfrak{q},\mathfrak{p}) = \begin{bmatrix} \mathbf{0} & \mathfrak{q}^\times \\ -\mathfrak{q}^{\times\top} & \mathfrak{p}^\times \end{bmatrix}, \qquad \mathfrak{p}^\times = \begin{bmatrix} \mathbf{0} & \hat{\mathfrak{p}}_\mathbf{v} \\ \hat{\mathfrak{p}}_\mathbf{v} & \hat{\mathfrak{p}}_{\boldsymbol{\omega}} \end{bmatrix}, \tag{2.38}$$

and input matrix $\mathcal{G}(\mathfrak{q},\mathfrak{p}) = \begin{bmatrix} \mathbf{0}^\top & \mathbf{B}(\mathfrak{q})^\top \end{bmatrix}^\top$, where $\mathfrak{q}^\times = \begin{bmatrix} \mathbf{R}^\top & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{r}}_1^\top & \hat{\mathbf{r}}_2^\top & \hat{\mathbf{r}}_3^\top \end{bmatrix}^\top$. The port-Hamiltonian formulation allows us to model dissipation elements in the dynamics by the dissipation matrix:

$$\mathbf{D}(\mathfrak{q},\mathfrak{p}) = \begin{bmatrix} \mathbf{D}_\mathbf{v}(\mathfrak{q},\mathfrak{p}) & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_{\boldsymbol{\omega}}(\mathfrak{q},\mathfrak{p}) \end{bmatrix} \in \mathbb{S}^{6\times 6}_{\succ 0}, \tag{2.39}$$

where the components $\mathbf{D}_\mathbf{v}(\mathfrak{q},\mathfrak{p})$ and $\mathbf{D}_{\boldsymbol{\omega}}(\mathfrak{q},\mathfrak{p})$ correspond to $\mathfrak{p}_\mathbf{v}$ and $\mathfrak{p}_{\boldsymbol{\omega}}$, respectively. The equations of motions on the $SE(3)$ manifold are written in port-Hamiltonian form as:

$$\dot{\mathbf{p}} = \mathbf{R}\frac{\partial \mathcal{H}(\mathfrak{q},\mathfrak{p})}{\partial \mathfrak{p}_\mathbf{v}}, \tag{2.40a}$$

$$\dot{\mathbf{r}}_i = \mathbf{r}_i \times \frac{\partial \mathcal{H}(\mathfrak{q},\mathfrak{p})}{\partial \mathfrak{p}_{\boldsymbol{\omega}}}, \quad i = 1,2,3 \tag{2.40b}$$

$$\dot{\mathfrak{p}}_\mathbf{v} = \mathfrak{p}_\mathbf{v} \times \frac{\partial \mathcal{H}(\mathfrak{q},\mathfrak{p})}{\partial \mathfrak{p}_{\boldsymbol{\omega}}} - \mathbf{R}^\top \frac{\partial \mathcal{H}(\mathfrak{q},\mathfrak{p})}{\partial \mathbf{p}} \tag{2.40c}$$
$$- \mathbf{D}_\mathbf{v}(\mathfrak{q},\mathfrak{p})\frac{\partial \mathcal{H}(\mathfrak{q},\mathfrak{p})}{\partial \mathfrak{p}_\mathbf{v}} + \mathbf{b}_\mathbf{v}(\mathfrak{q})\mathbf{u},$$

$$\dot{\mathfrak{p}}_{\boldsymbol{\omega}} = \mathfrak{p}_{\boldsymbol{\omega}} \times \frac{\partial \mathcal{H}(\mathfrak{q},\mathfrak{p})}{\partial \mathfrak{p}_{\boldsymbol{\omega}}} + \mathfrak{p}_\mathbf{v} \times \frac{\partial \mathcal{H}(\mathfrak{q},\mathfrak{p})}{\partial \mathfrak{p}_\mathbf{v}} + \tag{2.40d}$$
$$\sum_{i=1}^{3} \mathbf{r}_i \times \frac{\partial \mathcal{H}(\mathfrak{q},\mathfrak{p})}{\partial \mathbf{r}_i} - \mathbf{D}_{\boldsymbol{\omega}}(\mathfrak{q},\mathfrak{p})\frac{\partial \mathcal{H}(\mathfrak{q},\mathfrak{p})}{\partial \mathfrak{p}_{\boldsymbol{\omega}}} + \mathbf{b}_{\boldsymbol{\omega}}(\mathfrak{q})\mathbf{u},$$

where the input matrix is $\mathbf{B}(\mathfrak{q}) = \begin{bmatrix} \mathbf{b}_\mathbf{v}(\mathfrak{q})^\top & \mathbf{b}_{\boldsymbol{\omega}}(\mathfrak{q})^\top \end{bmatrix}^\top$.

2.3 Neural Ordinary Differential Equation Networks

In this section, we briefly describe neural ordinary differential equation (ODE) networks [21], which approximate the continuous-time closed-loop dynamics $\dot{\mathbf{s}} = \mathbf{f}(\mathbf{s}, \boldsymbol{\pi}(\mathbf{s}))$ of a system for some unknown control policy $\mathbf{u} = \boldsymbol{\pi}(\mathbf{s})$ by a neural network $\bar{\mathbf{f}}_{\boldsymbol{\theta}}(\mathbf{s})$. The parameters of $\bar{\mathbf{f}}_{\boldsymbol{\theta}}(\mathbf{s})$ are trained using a dataset $\mathcal{D} = \{t^{(i)}_{0:N}, \mathbf{s}^{(i)}_{0:N}\}_i$ of state trajectory samples $\mathbf{x}^{(i)}_n = \mathbf{s}^{(i)}(t^{(i)}_n)$ via forward and backward passes through a differentiable ODE solver, where the backward passes provide the gradient of the loss function. Given an initial state $\mathbf{x}^{(i)}_0$ at time $t^{(i)}_0$, a forward pass returns predicted states at times $t^{(i)}_1, \ldots, t^{(i)}_N$:

$$\{\bar{\mathbf{s}}^{(i)}_1, \ldots, \bar{\mathbf{s}}^{(i)}_N\} = \text{ODESolver}(\mathbf{x}^{(i)}_0, \bar{\mathbf{f}}_{\boldsymbol{\theta}}, t^{(i)}_1, \ldots, t^{(i)}_N). \tag{2.41}$$

The gradient of a loss function, $\sum_{i=1}^{D} \sum_{j=1}^{N} \ell(\mathbf{x}^{(i)}_j, \bar{\mathbf{s}}^{(i)}_j)$, is back-propagated by solving another ODE with adjoint states. Specifically, let $\mathbf{a} = \frac{\partial \mathcal{L}}{\partial \bar{\mathbf{s}}}$ be the adjoint state and $\boldsymbol{\alpha} = (\bar{\mathbf{s}}, \mathbf{a}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}})$ be the augmented state. The augmented state dynamics are [21]:

$$\dot{\boldsymbol{\alpha}} = \bar{\mathbf{f}}_{\mathbf{s}} = (\bar{\mathbf{f}}_{\boldsymbol{\theta}}, -\mathbf{a}^\top \frac{\partial \bar{\mathbf{f}}_{\boldsymbol{\theta}}}{\partial \bar{\mathbf{s}}}, -\mathbf{a}^\top \frac{\partial \bar{\mathbf{f}}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}). \tag{2.42}$$

The predicted state $\bar{\mathbf{s}}$, the adjoint state $\mathbf{a}$, and the derivatives $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$ can be obtained by a single call to a reverse-time ODE solver starting from $\boldsymbol{\alpha}_N = \boldsymbol{\alpha}(t_N)$:

$$\boldsymbol{\alpha}_0 = \left(\bar{\mathbf{s}}_0, \mathbf{a}_0, \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right) = \text{ODESolver}(\boldsymbol{\alpha}_N, \bar{\mathbf{f}}_{\mathbf{s}}, t_N), \tag{2.43}$$

where at each time $t_k, k = 1, \ldots, N$, the adjoint state $\mathbf{a}_k$ at time t_k is reset to $\frac{\partial \mathcal{L}}{\partial \bar{\mathbf{s}}_k}$. The resulting derivative $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$ is used to update the parameters $\boldsymbol{\theta}$ using gradient descent.

For physical systems, Zhong et al. [184] extends the neural ODE by integrating the Hamiltonian dynamics on $\mathbb{R}^n$ into the neural network model $\bar{\mathbf{f}}_{\boldsymbol{\theta}}(\mathbf{s})$, and consider zero-order hold control input $\mathbf{u}$, leading to a neural ODE network with the following approximated dynamics:

$$\begin{bmatrix} \dot{\mathbf{s}} \\ \dot{\mathbf{u}} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{f}}_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{u}) \\ \mathbf{0} \end{bmatrix}. \tag{2.44}$$

In Chapter 4, we impose the Hamiltonian dynamics on Lie groups on the architecture of $\bar{\mathbf{f}}_{\boldsymbol{\theta}}$, and learn the parameters $\boldsymbol{\theta}$ in Eq. (2.44) from data using neural ODE networks.

Recently, neural ODE networks have been extended from the vector space $\mathbb{R}^n$ to manifolds such as Lie groups [45, 42, 105, 179]. While it is possible to encode Hamiltonian dynamics in such Lie group neural ODE networks, we leave this for future work due to the lack of publicly open-sourced software.

2.4 Machine Learning Classifiers

2.4.1 Kernel Perceptron

In this section, we provide a summary on kernel perceptron [13] and Fastron [30, 31] which is useful for our derivations in the next sections. The *kernel perceptron* model is used to classify a set of N labeled data points. For $l = 1, \ldots, N$, a data point $\mathbf{x}_l$ with label $y_l \in \{-1, 1\}$ is assigned a weight $\alpha_l \in \mathbb{R}$. Training determines a set of M^+ positive support vectors and their weights $\Lambda^+ = \{(\mathbf{x}_i, \alpha_i)\}$ and a set of M^- negative support vectors and their weights $\Lambda^- = \{(\mathbf{x}_j^-, \alpha_j^-)\}$. The decision boundary is represented by a score function,

$$F(\mathbf{x}) = \sum_{i=1}^{M^+} \alpha_i^+ k(\mathbf{x}_i^+, \mathbf{x}) - \sum_{j=1}^{M^-} \alpha_j^- k(\mathbf{x}_j^-, \mathbf{x}), \tag{2.45}$$

where $k(\cdot, \cdot)$ is a kernel function and $\alpha_j^-, \alpha_i^+ > 0$. The sign of $F(\mathbf{x})$ is used to predict the class of a test point $\mathbf{x}$.

Fastron [30, 31] is an efficient training algorithm for the kernel perceptron model. It prioritizes updating misclassified points based on their margins instead of random selection as in the original kernel perceptron training. Our previous work [30, 31] shows that if $\alpha_l = \xi y_l - (\sum_{i \neq l} \alpha_i^+ k(\mathbf{x}_i^+, \mathbf{x}_l) - \sum_{j \neq l} \alpha_j^- k(\mathbf{x}_j^-, \mathbf{x}_l))$ for some $\xi > 0$, then $\mathbf{x}_l$ is correctly classified with label y_l. Based on this fact, Fastron utilizes one-step weight correction

$$\Delta\alpha = \xi y_l - (\sum \alpha_i^+ k(\mathbf{x}_i^+, \mathbf{x}_l) - \sum \alpha_j^- k(\mathbf{x}_j^-, \mathbf{x}_l)),$$

where $\xi = \xi^+$ if $y_l = 1$ and $\xi = \xi^-$ if $y_l = -1$.

2.4.2 Relevance Vector Machine

A relevance vector machine [166] is a sparse Bayesian approach for classification. Given a training dataset of N binary-labeled samples $\mathcal{D} = (\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_l, y_l)\}_l$, where $y_l \in \{-1, 1\}$, an RVM model maintains a sparse set of relevance vectors $\mathbf{x}_m$ for $m = 1, \ldots, M$. The relevance vectors map a point $\mathbf{x}$ to a feature vector $\mathbf{\Phi}_\mathbf{x} = [k_1(\mathbf{x}), k_2(\mathbf{x}), \ldots, k_M(\mathbf{x})]^\top \in \mathbb{R}^M$ via a kernel function $k_m(\mathbf{x}) := k(\mathbf{x}, \mathbf{x}_m)$. The likelihood of label y at point $\mathbf{x}$ is modeled by squashing a linear feature function:

$$F(\mathbf{x}) := \mathbf{\Phi}_\mathbf{x}^\top \mathbf{w} + b, \tag{2.46}$$

with weights $\mathbf{w} \in \mathbb{R}^M$ and bias $b \in \mathbb{R}$ through a function $\sigma : \mathbb{R} \mapsto [0, 1]$:

$$\mathbb{P}(y = 1|\mathbf{x}, \mathbf{w}) = \sigma(F(\mathbf{x})), \mathbb{P}(y = -1|\mathbf{x}, \mathbf{w}) = 1 - \sigma(F(\mathbf{x})).$$

Note that Eq. (2.45) is a special case of (2.46) with $b = 0$. Examples of σ are the logistic function $\sigma(f) := \frac{1}{1+\exp(-f)}$ and the probit function $\sigma(f) := \int_{-\infty}^{f} \varphi(z)dz$, where $\varphi(z) := \frac{1}{\sqrt{2\pi}} \exp(-z^2/2)$ is the standard normal probability density. The data likelihood of the whole training set is:

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{l=1}^{N} \sigma(F(\mathbf{x}_l))^{\frac{1+y_l}{2}} \left(1 - \sigma(F(\mathbf{x}_l))\right)^{\frac{1-y_l}{2}}. \tag{2.47}$$

An RVM model imposes a Gaussian prior on each weight w_m with zero mean and precision ξ_m (i.e., variance $1/\xi_m$):

$$p(\mathbf{w}|\xi) = (2\pi)^{M/2} \prod_{m=1}^{M} \xi_m^{1/2} \exp\left(-\frac{\xi_m w_m^2}{2}\right). \tag{2.48}$$

The weight posterior is obtained via Bayes' rule:

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \xi) = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w}|\xi)}{p(\mathbf{y}|\mathbf{X}, \xi)}. \tag{2.49}$$

The precision $\boldsymbol{\xi}$ is determined via type-II maximum likelihood estimation, i.e., by maximizing

the marginal likelihood:

$$\mathcal{L}(\boldsymbol{\xi}) = \log p(\mathbf{y}|\mathbf{X}, \xi) = \log \int p(\mathbf{y}|\mathbf{X}, \mathbf{w}) p(\mathbf{w}|\xi) d\mathbf{w}. \tag{2.50}$$

Given a maximizer $\boldsymbol{\xi}$, the posterior $p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \xi)$ is generally intractable and approximated by a Gaussian distribution $p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ using Laplace approximation [114]. Training consists in determining $\boldsymbol{\xi}$, $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$.

Approximation of the weight posterior $p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \xi)$ is performed by fitting a Gaussian density function around its mode $\boldsymbol{\mu}$, the maximizer of

$$L(\mathbf{w}) := \log(p(\mathbf{y}|\mathbf{X}, \mathbf{w}) p(\mathbf{w}|\xi)). \tag{2.51}$$

Substituting (2.47) and (2.48) in (2.51), we can obtain the gradient and Hessian of $L(\mathbf{w})$ for the probit function σ:

$$\nabla L(\mathbf{w}) = \boldsymbol{\Phi}^\top \boldsymbol{\delta} - \mathbf{A}\mathbf{w}, \quad \nabla^2 L(\mathbf{w}) = -\boldsymbol{\Phi}^\top \mathbf{B} \boldsymbol{\Phi} - \mathbf{A}, \tag{2.52}$$

where $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ is the feature matrix with entries $\boldsymbol{\Phi}_{i,j} := k_j(\mathbf{x}_i)$, $\boldsymbol{\delta} \in \mathbb{R}^N$ is a vector with entries $\delta_l := \frac{\varphi(y_l F(\mathbf{x}_l))}{\sigma(y_l F(\mathbf{x}_l))} y_l$, $\mathbf{A} := \operatorname{diag}(\boldsymbol{\xi}) \in \mathbb{R}^{M \times M}$, $\mathbf{B} := \operatorname{diag}(\mathbf{D}\boldsymbol{\Phi}^\top \mathbf{w} + b\boldsymbol{\delta} + \mathbf{D}\boldsymbol{\delta}) \in \mathbb{R}^{N \times N}$, and $\mathbf{D} := \operatorname{diag}(\boldsymbol{\delta}) \in \mathbb{R}^{N \times N}$. The Hessian is negative semi-definite and, hence, $L(\mathbf{w})$ is concave. Setting $L(\mathbf{w}) = 0$, we obtain a Gaussian approximation $p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with:

$$\boldsymbol{\Sigma} = (\boldsymbol{\Phi}^\top \mathbf{B} \boldsymbol{\Phi} + \mathbf{A})^{-1}, \tag{2.53}$$

$$\boldsymbol{\mu} = \boldsymbol{\Sigma} \boldsymbol{\Phi}^\top \mathbf{B} \left(\boldsymbol{\Phi} \mu + \mathbf{B}^{-1} \boldsymbol{\delta} \right), \tag{2.54}$$

where $\boldsymbol{\mu}$ is defined implicitly and is obtained via first- or second-order ascent in practice [120]. Laplace approximation provides a closed-form approximated posterior, which enables efficient classifications of points, line segments and curves, as shown in Section 5.1. When the true posterior is multi-modal, Laplace approximation might not provide sufficient accuracy because it captures only one of the modes.

At test time, due to the Laplace approximation, the predictive distribution of a query point $\mathbf{x}$ becomes:

$$p(y|\mathbf{x}, \boldsymbol{\xi}) \approx \int p(y|\mathbf{x}, \mathbf{w}) p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{w}. \tag{2.55}$$

The usual formulation of RVM [166] uses a logistic function for σ, requiring additional approximations to the integral in (2.55). We emphasize that using a probit function, instead, enables a closed-form for the predictive distribution:

$$\begin{aligned} p(y|\mathbf{x}, \boldsymbol{\xi}) &\approx \int \sigma(y(\boldsymbol{\Phi}_{\mathbf{x}}^{\top} \mathbf{w} + b)) p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{w} \\ &= \sigma\left(\frac{y(\Phi_{\mathbf{x}}^{\top} \boldsymbol{\mu} + b)}{\sqrt{1 + \Phi_{\mathbf{x}}^{\top} \boldsymbol{\Sigma} \Phi_{\mathbf{x}}}} \right). \end{aligned} \tag{2.56}$$

This expression enables our later results on closed-form classification of curves in Section 5.1.

Acknowledgments

Chapter 2, in part, is a reprint of the material as it appears in T. Duong, M. Yip, N. Atanasov, "*Autonomous Navigation in Unknown Environments with Sparse Bayesian Kernel-based Occupancy Mapping*", IEEE Transactions on Robotics (T-RO), vol. 38, no. 6, pp. 3694-3712, 2022, in T. Duong, N. Das, M. Yip, N. Atanasov, "*Autonomous Navigation in Unknown Environments using Sparse Kernel-based Occupancy Mapping*", International Conference on Robotics and Automation (ICRA), pp. 9666-9672, 2020, in T. Duong, N. Atanasov, "*Hamiltonian-based Neural ODE Networks on the SE(3) Manifold For Dynamics Learning and Control*", Robotics: Science and Systems (RSS), 2021, in T. Duong, N. Atanasov, "*Physics-guided Learning-based Adaptive Control on the SE(3) Manifold*", Physical Reasoning and Inductive Biases for the Real World Workshop, 2021, and in T. Duong, N. Atanasov, "*Adaptive Control of SE(3) Hamiltonian Dynamics with Learned Disturbance Features*", IEEE Control Systems Letters (L-CSS), vol. 6, pp. 2773-2778, 2022. Chapter 2, in part, has been submitted for publication of the material as it may appear in T. Duong, A. Altawaitan, J. Stanley, N. Atanasov, "*Port-Hamiltonian-based Neural ODE Networks on Lie Groups For Robot Dynamics Learning and Control*", Transactions on Robotics, 2024. The dissertation author was the primary investigator and author of these papers.

Chapter 3

Learning Sparse Occupancy Map Representations

Autonomous navigation in robotics involves online localization, mapping, motion planning, and control in partially known environments perceived through streaming data from onboard sensors [61, 78]. This chapter focuses on the occupancy mapping problem and, specifically, on enabling large-scale, yet compact, representations for autonomous navigation. Occupancy mapping is a well established and widely studied problem in robotics and a variety of explicit and implicit map representations have been proposed. Explicit maps model the obstacle surfaces directly, e.g., via surfels [68, 87, 176, 173, 9], geometric primitives [81, 16, 124, 180, 152], or polygonal meshes [162, 135, 141]. Implicit maps model the obstacle surfaces as the level set of an occupancy [43, 82, 119, 70, 181, 171] or signed distance [28, 85, 76, 127, 63] or spatial function encoded via voxels [125, 127, 63] or octrees [70, 181, 171]. The goal of this chapter is to generate *sparse probabilistic* occupancy maps online, enabling large-scale environment modeling, map uncertainty quantification.

We first develop a kernel perceptron model for online occupancy mapping [39] in Section 3.2. The model uses support vectors and a kernel function to represent obstacle boundaries in configuration space. The number of support vectors scales with the complexity of the obstacle boundaries rather than the environment size. We develop an online training algorithm to update the support vectors incrementally as new range observations of the local surroundings are provided by the robot's sensors. Our kernel perceptron model, however, provides occupancy labels without a probability distribution, making the classification accuracy susceptible to measure-

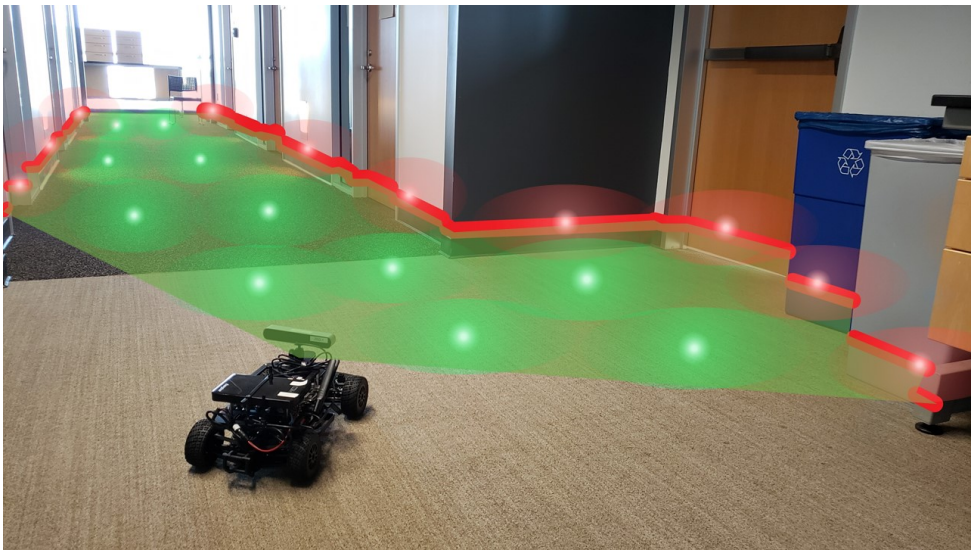


Figure 3.1. A ground robot equipped with a lidar (red) and our map representation as a sparse set of occupied (light red) and free (green) relevance vectors.

ment noise. Since unknown regions are frequently assumed free for motion planning purposes, the lack of probabilistic information also does not allow us to distinguish between well-observed and unseen regions. This is especially important in active exploration problems, where the robot autonomously chooses the unknown regions to explore.

To handle noisy measurements and probabilistically model well-observed and unknown regions, we introduce a sparse Bayesian formulation of the occupancy mapping problem based on relevance vector machine (RVM) inference that enables online sparse Bayesian kernel-based occupancy mapping [40]. Inspired by GP mapping techniques, we utilize a kernel function to capture occupancy correlations but focus on a compact representation of obstacle boundaries by building an RVM model, i.e. a sparse set of relevance vectors, incrementally from streaming local sensor data. Specifically, only a local subset of the relevance vectors is updated each time using our incremental RVM training algorithm.

Contributions. In this chapter, we introduce a sparse Bayesian kernel-based mapping method that:

- represents continuous-space probabilistic occupancy using a sparse set of relevance vectors stored in an R^*-tree data structure,
- allows online map updates from streaming partial observations using an incremental Rel-

evance Vector Machine training algorithm with the predictive distribution modeled by a probit function.

3.1 Sparse Probabilistic Occupancy Mapping Problem

We restate the mapping problem formulated in Problem 1 for readability. Consider a robot with state $\mathbf{s} \in \mathcal{S}$, consisting of the robot's position $\mathbf{p} \in [0,1]^d$ and other variables such as orientation, velocity, etc., navigating in an unknown environment (Figure 3.1). Let $\mathcal{O} \subset [0,1]^d$ be a closed set representing occupied space and let $\mathcal{F}$ be its complement, representing free space. Assume that the robot can be enclosed by a sphere of radius $r \in \mathbb{R}_{>0}$ centered at $\mathbf{p}$. In configuration space (C-space), the robot body becomes a point $\mathbf{p}$, while the obstacle space and free space are transformed as $\bar{\mathcal{O}} = \cup_{\mathbf{x}\in\mathcal{O}} \mathcal{B}(\mathbf{x}, r)$, where $\mathcal{B}(\mathbf{x}, r) = \{\mathbf{x}' \in [0,1]^d : \|\mathbf{x} - \mathbf{x}'\|_2 \leq r\}$, and $\bar{\mathcal{F}} = [0,1]^d \setminus \bar{\mathcal{O}}$. Let $\bar{\mathcal{S}}$ be the subset of the robot state space that corresponds to the collision-free robot positions $\bar{\mathcal{F}}$.

The robot is equipped with a sensor, such as a Lidar or depth camera, that provides distance measurements $\mathbf{z}_k$ at time t_k to the obstacle space $\mathcal{O}$ within its field of view. Our objective is to construct an occupancy map $m_k : [0,1]^d \to \{-1, 1\}$ of the C-space based on accumulated observations $\mathbf{z}_{0:k}$, where "-1" and "1" mean "free" and "occupied", respectively. As the robot is navigating, new sensor data are used to update the map as a function , $m_{k+1} = g(m_k, \mathbf{z}_k)$, of the previous estimate m_k and a newly received range observation $\mathbf{z}_k$. Assuming unobserved regions are free, we rely on m_k to plan a robot trajectory to a goal region $\mathcal{G} \subseteq \bar{\mathcal{S}}$. In this chapter, we develop a sparse Bayesian kernel-based map representation m_k, whose map updates $g(m_k, \mathbf{z}_k)$ are developed based on relevance vector machine incremental training.

3.2 Sparse Binary Kernel-based Occupancy Mapping

In this section, we present our sparse binary Kernel-based map (SKM) [39], which develops a sparse kernel perceptron model for online classification of occupied and free space in the environment. The model uses a set of support vectors and a kernel function to represent the obstacle boundaries in configuration space. The number of support vectors necessary for accurate classification scales with the complexity of the obstacle boundaries rather than the

environment size. Our approach extends the Fastron algorithm [30, 130], which efficiently trains a kernel perceptron model using a training dataset collected globally from the environment. We develop an online training procedure (Algorithm 1) that updates the support vectors incrementally as new range observations $\mathbf{z}_k$ of the local surroundings arrive. Given a training dataset $\mathcal{D} = \{(\mathbf{x}_l, y_l)\}$ generated from $\mathbf{z}_k$, Algorithm 1 prioritizes updating misclassified points' weight based on their margins (lines 6 and 7) and remove the redundant support vectors (line 8) without affecting the model. When the next local dataset arrives, it looks for new misclassified points and incrementally adds them to the set of support vectors. Algorithm 1 returns a set of M^+ positive support vectors and their weight $\Lambda^+ = \{(\mathbf{x}_i^+, \alpha_i^+)\}_i$ and a set of M^- negative support vectors and their weight $\Lambda^- = \{(\mathbf{x}_j^-, \alpha_j^-)\}_j$. The classifier decision boundary is characterized by a score function:

$$F(\mathbf{x}) = \sum_{i=1}^{M^+} \alpha_i^+ k(\mathbf{x}_i^+, \mathbf{x}) - \sum_{j=1}^{M^-} \alpha_j^- k(\mathbf{x}_j^-, \mathbf{x}), \tag{3.1}$$

where $k(\cdot, \cdot)$ is a kernel function and $\alpha_j^-, \alpha_i^+ > 0$. The occupancy of a query point $\mathbf{x}$ can be checked by evaluating the score function $F(\mathbf{x})$ in Eq. (3.1). Specifically, $m_k(\mathbf{x}) = -1$ if $F(\mathbf{x}) < 0$ and $m_k(\mathbf{x}) = 1$ if $F(\mathbf{x}) \geq 0$. The score calculation becomes slower when the number of support vectors increases. We improve on this by storing the support vectors in an R^*-tree data structure and efficiently query K^+ and K^- nearest positive and negative support vectors (line 1 in Algorithm 1) from the R^*-tree to approximate $F(\mathbf{x})$.

The sparse kernel-based model [39] is accurate, updates efficiently from streaming range data, and evaluates curves $\mathbf{p}(t)$ for collisions without sampling. However, the model does not provide occupancy probability, which is desirable in autonomous navigation applications for distinguishing between unknown and well-observed free regions and for identifying map areas with large uncertainty. This observation motivates us to develop a sparse probabilistic model for online occupancy classification and efficient collision checking.

3.3 Sparse Bayesian Kernel-based Occupancy Mapping

In this section, we develop an online probit relevance vector machine (RVM) training algorithm that builds a sparse probabilistic model for online occupancy mapping from streaming

Algorithm 1. Incremental Kernel Perceptron Training [39]

Input: Support vectors $\Lambda^+ = \{(\mathbf{x}_i^+, \alpha_i^+)\}_i$ and $\Lambda^- = \{(\mathbf{x}_j^-, \alpha_j^-)\}_j$ stored in an R^*-tree; Local dataset $\mathcal{D} = \{(\mathbf{x}_l, y_l)\}$; $\xi^+, \xi^- > 0$; N_{max}.
Output: Updated Λ^+, Λ^-.

Query K^+, K^- nearest negative and positive support vectors from an R^*-tree data structure.
for $(\mathbf{x}_l, y_l)$ in $\mathcal{D}$ **do**
    Calculate $F_l = \sum_{i=1}^{K^+} \alpha_i^+ k(\mathbf{x}_i^+, \mathbf{x}_l) - \sum_{j=1}^{K^-} \alpha_j^- k(\mathbf{x}_j^-, \mathbf{x}_l)$
for $t = 1$ to N_{max} **do**
    if $y_l F_l > 0 \quad \forall l$ **then return** Λ^+, Λ^-
    $m = \text{argmin}_l y_l F_l$
    WEIGHT CORRECTION$(F_m, y_m, \Lambda^+, \Lambda^-, \xi^+, \xi^-)$
    REDUNDANCY REMOVAL$(\Lambda^+, \Lambda^-, \mathcal{D})$
return Λ^+, Λ^-
function WEIGHT CORRECTION$(F_m, y_m, \Lambda^+, \Lambda^-, \xi^+, \xi^-)$
    $\xi = \xi^+$ if $y_m > 0$; and $\xi = \xi^-$, otherwise.
    Calculate $\Delta_\alpha = \xi y_m - F_m$.
    if $\exists(\mathbf{x}_m, \alpha_m) \in \Lambda^+ \cup \Lambda^-$ **then**
        Update weights: $\alpha_m \mathrel{+}= y_m \Delta\alpha$, $F_l \mathrel{+}= k(\mathbf{x}_l, \mathbf{x}_m) y_m \Delta\alpha, \forall l$
    else
        Calculate $\alpha_m = y_m \Delta\alpha$
        Add $(\mathbf{x}_m, \alpha_m)$ to Λ^+ if $y_m > 0$ and Λ^-, otherwise.
function REDUNDANCY REMOVAL$(\Lambda^+, \Lambda^-, \mathcal{D})$
    for $(\mathbf{x}_l, y_l) \in \mathcal{D}$ **do**
        if $\exists(\mathbf{x}_l, \alpha_l) \in \Lambda^+ \cup \Lambda^-$ and $y_l(F_l - \alpha_l^+) > 0$ **then**
            Remove $(\mathbf{x}_l, \alpha_l)$ from Λ^+ or Λ^-
            Update $F_n \mathrel{-}= k(\mathbf{x}_l, \mathbf{x}_n)\alpha_l^+, \forall(\mathbf{x}_n, \cdot) \in \mathcal{D}$

range observations.

3.3.1 Sequential Relevance Vector Machine Training

To the determine the precision ξ of the weight prior in (2.48), Tipping and Faul [166] proposed a sequential training algorithm that starts from an empty set of relevance vectors, i.e., $\xi_l = \infty$, and incrementally introduces new vectors to maximize the marginal likelihood in (2.50):

$$\mathcal{L}(\boldsymbol{\xi}) \approx -\frac{1}{2}\left(N \log 2\pi + \log\det \mathbf{C} + \hat{\mathbf{t}}^\top \mathbf{C}^{-1} \hat{\mathbf{t}}\right) \tag{3.2}$$

where $\hat{\mathbf{t}} := \mathbf{\Phi}\mu + \mathbf{B}^{-1}\boldsymbol{\delta}$ and $\mathbf{C} := \mathbf{B} + \mathbf{\Phi}\mathbf{A}^{-1}\mathbf{\Phi}^{\top}$. For each $(\mathbf{x}_l, y_l)$ in the training set $\mathcal{D}$, define $\theta_l = q_l^2 - s_l$ as follows:

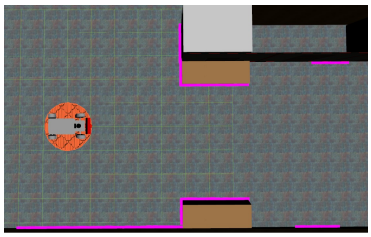
$$s_l := \begin{cases} \frac{\xi_l S_l}{\xi_l - S_l}, & \text{if } \xi_l < \infty \\ S_l, & \text{else} \end{cases} \quad q_l := \begin{cases} \frac{\xi_l Q_l}{\xi_l - S_l}, & \text{if } \xi_l < \infty \\ Q_l, & \text{else} \end{cases} \tag{3.3}$$

where $S_l = \mathbf{\Phi}_l^{\top}\mathbf{C}^{-1}\mathbf{\Phi}_l$, $Q_l = \mathbf{\Phi}_l^{\top}\mathbf{C}^{-1}\hat{\mathbf{t}}$, and $\mathbf{\Phi}_l$ is the l-th row of $\mathbf{\Phi}$. If $\theta_l > 0$, the point $\mathbf{x}_l$ is updated (if $\xi_l < \infty$) or added (if $\xi_l = \infty$) as a relevance vector with $\xi_l = \frac{s_l^2}{q_l^2 - s_l}$. If $\theta_l \leq 0$ and $\xi_l < \infty$, the point $\mathbf{x_l}$ is removed from the RVM model. These steps are shown in lines 8-12 of Algorithm 2.

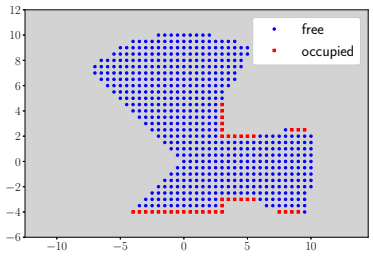
3.3.2 Online Mapping using Streaming Data

Existing techniques for RVM training assume that all data is available a priori. In this section, we develop an online RVM training algorithm (Algorithm 2) that updates the set of relevance vectors $\Lambda_k = \{\mathbf{x}_i^{(k)}, y_i^{(k)}, \xi_i^{(k)})\}_i$ incrementally using streaming data. Suppose that Λ_k has been obtained based on prior data $\mathcal{D}_0, \ldots, \mathcal{D}_k$. At time $k+1$, a new training set $\mathcal{D}_{k+1}$ is received. The training set generation depends on the application. We construct $\mathcal{D}_{k+1}$ using a lidar scan $\mathbf{z}_{k+1}$ of an unknown environment as detailed in Section 3.4. Relevance vectors are added or removed from Λ_k to correctly classify the latest changes, e.g., new or disappearing obstacles, in training set $\mathcal{D}_{k+1}$ without affecting the accuracy of the classification on the prior data and maintaining the sparsity of the model.

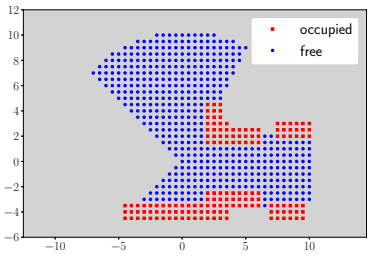
Algorithm 2 presents our online probit RVM training approach. The algorithm starts with the existing set of relevance vectors Λ_k and adds new relevance vectors based on the samples in $\mathcal{D}_{k+1}$ using the sequential training approach in Section 3.3.1. Instead of using the feature matrix $\mathbf{\Phi}$ (line 4) associated with all prior relevance vectors, we use a feature matrix approximation based on a local set Λ_{local} of K nearest relevance vectors (line 2). Section 3.6 provides a discussion on the computational improvements and assumptions of the score function approximation resulting from using Λ_{local} instead of Λ_k. For test time classification, we compute the mean $\boldsymbol{\mu}$ and covariance $\mathbf{\Sigma}$ of the Laplace approximation to the weight posterior according to Eq. (2.54) and (2.53). Laplace approximation requires all data $\mathcal{D} = \cup_{i=1}^{k+1}\mathcal{D}_i$, used for training up



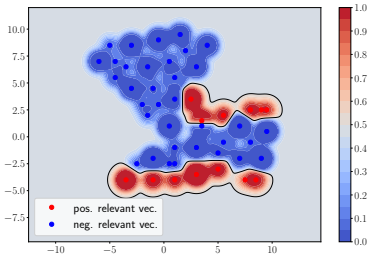
(a) Robot in an unknown environment with a laser scan.



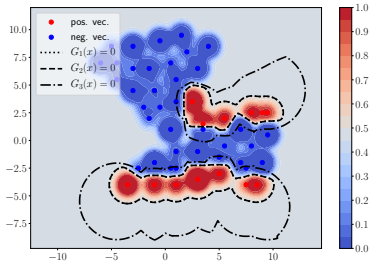
(b) Work-space samples from laser scans.



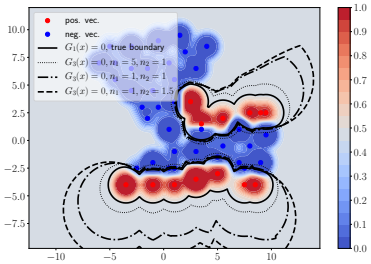
(c) Configuration-space samples.



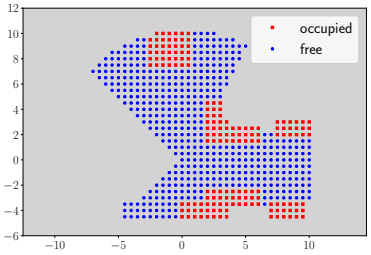
(d) Decision boundary with bias $b = -0.05$ and threshold $e = -0.01$, $\bar{e} = 0.494$.



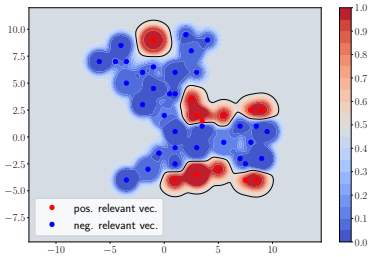
(e) Inflated boundaries generated by $G_1(\mathbf{x}), G_2(\mathbf{x}), G_3(\mathbf{x})$ with $n_1 = n_2 = 1$.



(f) Inflated boundary $G_3(\mathbf{x}) = 0$ with various n_1, n_2.



(g) Configuration-space samples from a second laser scan



(h) Our updated RVM model reflecting the latest changes.

Figure 3.2. Example of our mapping method for a ground robot in an unknown environment.

to time $k+1$ but only the local dataset $\mathcal{D}_{k+1}$ is available. Interestingly, the set Λ_{k+1} of relevance vectors itself globally and sparsely represents all the data used for training and, therefore, can be used for Laplace approximation (line 15). If additional computation for Laplace approximation is not feasible, one might directly store the weight mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ (line 15) over time. The memory requirements for either case are discussed in Section 3.6.

Figure 3.2a depicts a ground robot equipped with a lidar scanner whose goal is to build an occupancy map of the environment. Figure 3.2c plots the training set $\mathcal{D}_1$ generated from the first lidar scan $\mathbf{z}_1$, assuming the current set of relevance vectors Λ_k is empty. Figure 3.2d shows the trained RVM model as a sparse set of relevance vectors, serving as a sparse probabilistic

Algorithm 2. Online Probit RVM Training.

Input: Relevance vectors $\Lambda_k = \{(\mathbf{x}_i^{(k)}, y_i^{(k)}, \xi_i^{(k)})\}$; training set $\mathcal{D}_{k+1} = \{(\mathbf{x}_l, y_l)\}_l$; number of nearest relevance vectors to use K (optional)

Output: Relevance vectors $\Lambda_{k+1} = \{(\mathbf{x}_i^{(k+1)}, y_i^{(k+1)}, \xi_i^{(k+1)})\}$; weight posterior mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Initialize $\Lambda_{k+1} = \Lambda_k$.
if K is defined **then** $\Lambda_{local} = K$ nearest relevance vectors from Λ_k
else $\Lambda_{local} = \Lambda_k$.
$\boldsymbol{\Phi} = \textsc{FeatureMatrix}(\Lambda_{local}, \mathcal{D}_{k+1})$
$\xi_l = \infty$ for each $(\mathbf{x}_l, y_l)$ in $\mathcal{D}_{k+1}$
$\boldsymbol{\Sigma}, \mu = \textsc{LaplaceApproximation}(\Lambda_{local}, \mathcal{D}_{k+1})$.
while not converged and max number iterations not reached **do**
    Pick a candidate $(\mathbf{x}_m, y_m)$ from $\mathcal{D}_{k+1}$.
    Calculate $S_m, Q_m, s_m, q_m, \theta_m$.
    If $\theta_m > 0$ and $\xi_m = \infty$, add $(\mathbf{x}_m, y_m, \xi_m)$ to Λ_{local}.
    If $\theta_m \leq 0$ and $\xi_m < \infty$, remove $(\mathbf{x}_m, y_m, \xi_m)$ from Λ_{local}.
    If $\theta_m > 0$ and $\xi_m < \infty$, re-estimate $\xi_m = \frac{s_m^2}{q_m^2 - s_m}$ in Λ_{local}.
    $\boldsymbol{\Sigma}, \mu = \textsc{LaplaceApproximation}(\Lambda_{local}, \mathcal{D}_{k+1})$.
$\Lambda_{k+1} = \Lambda_{k+1} \cup \Lambda_{local}$.
$\boldsymbol{\Sigma}, \mu = \textsc{GlobalPosteriorApproximation}(\Lambda_{k+1})$
return Λ_{k+1}, $\boldsymbol{\Sigma}$, μ

function $\textsc{FeatureMatrix}\ (\Lambda, \mathcal{D})$
    Calculate $\boldsymbol{\Phi}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$ for all $\mathbf{x}_j \in \Lambda$ and all $\mathbf{x}_i \in \mathcal{D}$
    return $\boldsymbol{\Phi}$
function $\textsc{LaplaceApproximation}(\Lambda, \mathcal{D})$
    Calculate $\boldsymbol{\Sigma}$, μ for relevance vectors Λ using $\mathcal{D}$ (Eq. (2.53) and (2.54)).
    return $\boldsymbol{\Sigma}, \mu$.
function $\textsc{GlobalPosteriorApproximation}(\Lambda)$
    return $\textsc{LaplaceApproximation}(\Lambda, \Lambda)$.

occupancy map of the environment, incrementally updated via the streaming lidar scans. To illustrate map updates based on the latest depth measurements, we consider a second scan where part of the obstacles in the first scan disappears and a new obstacle appears in the second scan (Figure 3.2g). The RVM model, trained with the first laser scan, is updated with the second scan using our online RVM training algorithm (Algorithm 2), reflecting the obstacle changes in the environment as plotted in Figure 3.2h. A map representation is useful for autonomous navigation (Problem 3) only if it allows checking potential robot trajectories $\mathbf{s}(t)$, e.g. by sampling and checking points for collision described in the next section.

3.4 Online Mapping

We consider a robot placed in an unknown environment at time t_k as illustrated in Figure 3.1. It is equipped with a lidar scanner measuring distances to nearby obstacles. Samples generated from the lidar range scan $\mathbf{z}_k$ are shown in Figure 3.2b. Since the robot body is bounded by a sphere of radius r, each laser ray end point in configuration space becomes a ball-shaped obstacle, while the robot body becomes a point. To generate local training data, the occupied and free C-space areas observed by the lidar are sampled (e.g., on a regular grid). As shown in Figure 3.2c, this generates a set $\bar{\mathcal{D}}_k$ of points with label "1" (occupied) in the ball-shaped occupied areas and with label "-1" (free) between the robot position and each laser end point. To accelerate training, only the difference between two consecutive local datasets $\mathcal{D}_k = \bar{\mathcal{D}}_k \setminus \bar{\mathcal{D}}_{k-1}$ is used in our online RVM training algorithm (Algorithm 2). Storing the sets of relevance vectors Λ_k over time requires significantly less memory than storing the training data $\cup_k \mathcal{D}_k$. The occupancy of a query point $\mathbf{x}$ can be estimated from the relevance vectors by evaluating the function $G_1(\mathbf{x})$ in Eq. (3.5). Specifically, $m_k(\mathbf{x}) = -1$ if $G_1(\mathbf{x}) \leq 0$ and $m_k(\mathbf{x}) = 1$ if $G_1(\mathbf{x}) > 0$. Figure 3.2d illustrates the boundaries generated by Algorithm 2.

3.5 Efficient Relevance Vector Machine Inference

This section discusses classification using the predictive distribution in Eq. (2.56). This can be use for classification of general curves in robot motion planning by successively checking a dense set of points, sampled along the curve.

Consider a set Λ of M relevance vectors with prior weight precision $\boldsymbol{\xi}$ and mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ of the approximate weight posterior $p(\mathbf{w}|\mathbf{y}, \mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$. To classify a query point $\mathbf{x}$ using the RVM model, we place a threshold $\bar{e} = \sigma(e)$ on the probability $\mathbb{P}(y = 1|\mathbf{x}, \boldsymbol{\xi})$ (Def. 10). Figure 3.2d illustrates the decision boundary defined by Def. 10 with threshold $\bar{e} = \sigma(e) = 0.494$, i.e., $e = -0.01$.

Definition 10. Let $\bar{e} \in [0, 1]$ and $e := \sigma^{-1}(\bar{e})$. A point $\mathbf{x}$ is classified as "-1" (free) if

$$\mathbb{P}(y = 1|\mathbf{x}, \boldsymbol{\xi}) = \sigma\left(\frac{\Phi_{\mathbf{x}}^{\top}\boldsymbol{\mu} + b}{\sqrt{1 + \Phi_{\mathbf{x}}^{\top}\boldsymbol{\Sigma}\Phi_{\mathbf{x}}}}\right) \leq \bar{e}, \tag{3.4}$$

or, equivalently, if

$$G_1(\mathbf{x}) := \boldsymbol{\Phi}_{\mathbf{x}}^{\top}\boldsymbol{\mu} + b - e\sqrt{1 + \boldsymbol{\Phi}_{\mathbf{x}}^{\top}\boldsymbol{\Sigma}\boldsymbol{\Phi}_{\mathbf{x}}} \leq 0. \tag{3.5}$$

The condition in Eq. (3.5) can be verified for a given point but it is challenging to obtain an explicit expression in terms of $\mathbf{x}$. If, instead of a point $\mathbf{x}$, we consider a time-parameterized curve $\mathbf{p}(t)$, then Eq. (3.5) becomes a nonlinear programming feasibility problem in t. To avoid nonlinear programming, we develop a series of upper bounds for $G_1(\mathbf{x})$ that make the condition for classifying a point as free more conservative but with a simpler dependence on $\mathbf{x}$.

Proposition 1. *For a non-negative kernel function $k_m(\mathbf{x}) := k(\mathbf{x}, \mathbf{x}_m)$, a point $\mathbf{x}$ is classified as "-1" if*

$$G_2(\mathbf{x}) := \sum_{m=1}^{M} (\mu_m - e\mathbb{1}_{\{e\leq 0\}}\sqrt{\lambda_{\max}})k_m(\mathbf{x}) + b - e \leq 0, \tag{3.6}$$

where $\lambda_{\max} \geq 0$ is the largest eigenvalue of the covariance $\boldsymbol{\Sigma}$, μ_m is the mth element of the mean $\boldsymbol{\mu}$, and $\mathbb{1}_{\{e\leq 0\}}$ is an indicator function which equals 1 if $e \leq 0$ and 0, otherwise.

Proof. Please refer to Appendix B.1. □

The relaxed condition in Eq. (3.6) adjusts the weights of the relevance vectors by an amount of $\delta\mu = -e\mathbb{1}_{\{e\leq 0\}}\lambda_{max} \geq 0$. Intuitively, this increases the effect of the positive relevance vectors, leading to a more conservative condition than Def. 10. Prop. 1 also allows us to use only the largest eigenvalue λ_{max} of $\boldsymbol{\Sigma}$ for point classification, which is easier to obtain and store than the whole covariance matrix $\boldsymbol{\Sigma}$. Methods for computing λ_{max} are discussed in Section 3.6.1.

To simplify the notation, let $\nu_m := \mu_m - e\mathbb{1}_{\{e \leq 0\}}\lambda_{max}$ be the corrected relevance vector weights and split Λ into M^+ positive relevance vectors $\Lambda^+ = \{(\mathbf{x}_m^+, \nu_m^+)\}$ and M^- negative relevance vectors $\Lambda^- = \{(\mathbf{x}_m^-, \nu_m^-)\}$, where $\nu_m^+ = \nu_m$ if $\nu_m > 0$ and $\nu_m^- = -\nu_m$ if $\nu_m < 0$. Now, Eq. (3.6) can be re-written as:

$$G_2(\mathbf{x}) = \sum_{i=1}^{M^+} \nu_i^+ k(\mathbf{x}, \mathbf{x}_i^+) - \sum_{j=1}^{M^-} \nu_j^- k(\mathbf{x}, \mathbf{x}_j^-) + b - e \leq 0. \tag{3.7}$$

Hence, Prop. 1 allows us to make an important connection between sparse kernel classification with a 'hard' decision threshold (Section 3.2) and its Bayesian counterpart (Section 3.3.2). Specifically, after the relevance vector weight correction, Eq. (3.6) is equivalent to the kernel perceptron score in Eq. (2.45) except for the bias term $b-e$. Therefore, model inference results in this section and in Section 5.1 also hold for our sparse binary map representation in Section 3.2.

One of the motivations for developing a Bayesian map representation is to distinguish between observed and unobserved regions in the environment. Intuitively, as a query point $\mathbf{x}$ is chosen further away from "observed" regions, where training data has been obtained, its correlation with existing relevance vectors, measured by $k(\mathbf{x}, \mathbf{x}_m)$, decreases. To capture and exploit this property, we assume that the kernel has a common radial basis function structure that depends only on a quadratic norm $\|\mathbf{\Gamma}(\mathbf{x} - \mathbf{x}_m)\|$.

Assumption 1. Let $k(\mathbf{x}, \mathbf{x}_m) := \eta \exp\left(-\|\mathbf{\Gamma}(\mathbf{x} - \mathbf{x}_m)\|^2\right)$ with parameters $\eta > 0$ and $\mathbf{\Gamma} \in \mathbb{R}^{d \times d}$.

In our application, the kernel parameters η and $\mathbf{\Gamma}$ may be optimized offline via automatic relevance determination [122] using training data from known occupancy maps. Under this assumption, the feature vector $\mathbf{\Phi}_\mathbf{x}$ tends to 0 as $\mathbf{x}$ goes towards unobserved regions and the occupancy probability $\mathbb{P}(y = 1|\mathbf{x}, \boldsymbol{\xi})$ tends to $\sigma(b)$ in Eq. (3.4). Therefore, the value of $\sigma(b)$ represents the occupancy probability of points in the unknown regions. In other words, $\sigma(b)$ specifies how much we trust that unknown regions are occupied and should be a constant. For this reason, the bias b is fixed in our online RVM training algorithm. If we are optimistic about the unknown regions, the parameter b can be set to a large negative number, i.e. $\sigma(b) \approx 0$, and the decision boundary shrinks towards the occupied regions. If we want the robot to be cautious about the unknown regions, the parameter b can be set to a large positive number, i.e. $\sigma(b) \approx 1$,

and the decision boundary expands towards the unknown regions. A common assumption in motion planning [88] is to treat unknown regions as free in order to allow trajectory planning to goals in the unknown space. In the context of this dissertation, this means that the occupancy probability of points in unknown regions, $\sigma(b)$, should be lower than or equal to the decision threshold $\bar{e} = \sigma(e)$ in Def. 10.

Assumption 2. Assume that $e \geq b$ and, hence, $\bar{e} \geq \sigma(b)$.

For a general decision threshold, $e \geq b$, and a kernel function $k_m(\mathbf{x})$ satisfying Assumption 1, we develop an explicit condition for classifying a point $\mathbf{x}$ as free.

Proposition 2. *For integers $n_1, n_2 \geq 1$, define $\rho(a,b) := (n_1+n_2)\left(\frac{a}{n_1}\right)^{\frac{n_1}{n_1+n_2}}\left(\frac{b}{n_2}\right)^{\frac{n_2}{n_1+n_2}}$. A point $\mathbf{x}$ is classified as "-1" if*

$$G_3(\mathbf{x}) := \left(\sum_{i=1}^{M^+} \nu_i^+\right) k(\mathbf{x}, \mathbf{x}_*^+) - \rho(e-b, \nu_j^- k(\mathbf{x}, \mathbf{x}_j^-)) \leq 0, \tag{3.8}$$

where $\mathbf{x}_*^+$ is the closest positive relevance vector to $\mathbf{x}$ and $\mathbf{x}_j^-$ is any negative relevance vector.

Proof. Please refer to Appendix B.2. □

Figure 3.2e illustrates the exact RVM decision boundary from Eq. (3.5), $G_1(\mathbf{x}) = 0$, and the boundaries $G_2(\mathbf{x}) = 0$ and $G_3(\mathbf{x}) = 0$ resulting from the upper bounds in Prop. 1 and Prop. 2. Note that the boundary generated by $G_2(\mathbf{x})$ is very close to the true boundary from $G_1(\mathbf{x})$, empirically showing that the bound $G_2(\mathbf{x})$ is tight. The upper bound $G_3(\mathbf{x})$ provides a conservative "inflated boundary", whose accuracy can be controlled via the integers n_1, n_2 in Prop. 2. Note that $G_3(\mathbf{x})$ is inaccurate mainly in the unknown regions because the Arithmetic Mean-Geometric Mean inequality used in Prop. 2's proof (Appendix B.2) effectively replaces the kernel function $k(\mathbf{x}, \mathbf{x}_j^-)$ by a slower decaying one $k(\mathbf{x}, \mathbf{x}_j^-)^{\frac{n_2}{n_1+n_2}}$. This suits the intuition that unknown regions should be categorized as free more cautiously. Figure 3.2f shows that increasing the ratio n_2/n_1 makes the "inflated boundary" closer to the true decision boundary in the unknown regions but slightly looser in the well-observed regions and vice versa.

3.6 Computational and Storage Improvements

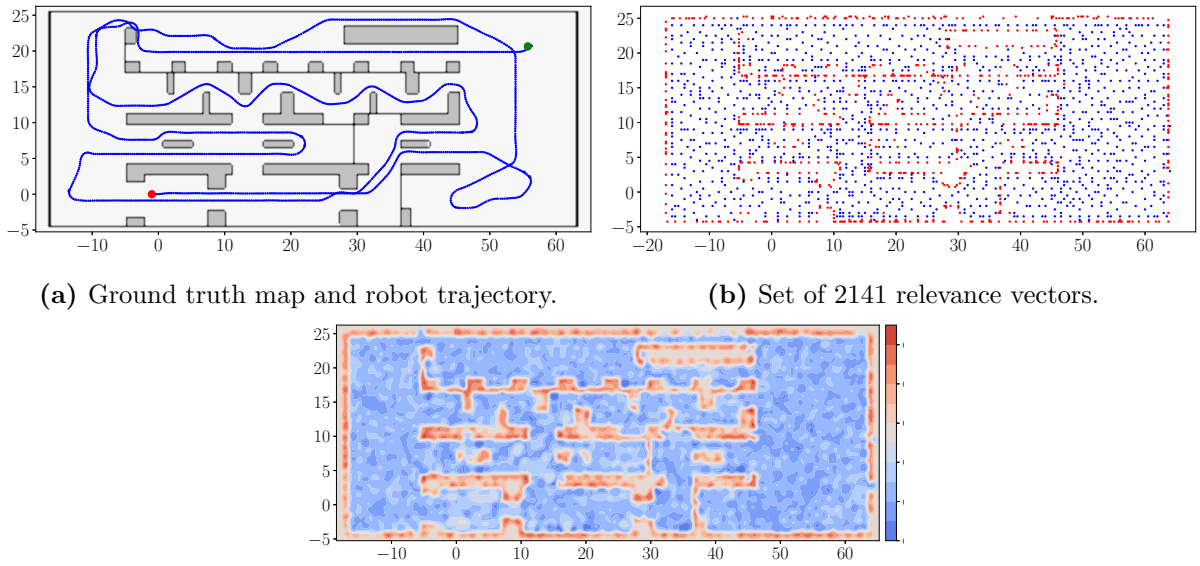
3.6.1 Computational Improvements

In the context of autonomous navigation, as a robot explores new regions of its environment, the number of relevance vectors required to represent the obstacle boundaries increases. Since the score function in Eq. (2.46) depends on all relevance vectors, the training time (Algorithm 2) and the classification time (Def. 10 for points, Algorithm 3 for lines, and Algorithm 4 for curves) increase as well. We propose an approximation to the score function $F(\mathbf{x})$ for the radial basis kernel in Assumption 1. Since $k(\mathbf{x}, \mathbf{x}_m)$ approaches zero rapidly as the distance between $\mathbf{x}$ and $\mathbf{x}_m$ increases, the value of $F(\mathbf{x})$ is not affected significantly by relevance vectors far from $\mathbf{x}$. We use R^*-tree data structures constructed from the relevance vectors Λ^+, Λ^- to allow efficient lookup of the nearest K^+ and K^- positive and negative relevance vectors. Approximating the score function $F(\mathbf{x})$ using the nearest K^+ and K^- relevance vectors improves its computational complexity from $O(M)$ to $O(\log M)$. Similarly, to classify a point $\mathbf{x}$, the M-dimensional feature vector $\mathbf{\Phi}_x$, may be approximated by a K-dimensional one using the K relevance vectors closest to $\mathbf{x}$. Classification of a line segment or a curve in Prop. 3 and 4 can be approximated by using the K^+ and K^- nearest positive and negative relevance vectors. The computational complexities of Eq. (5.4), (5.5), (5.6), and (5.7) improve from $O(M)$ and $O(M^2)$ to $O(\log M)$.

The line and curve classification algorithms depend on Prop. 1 which requires the largest eigenvalue λ_{max} of the weight posterior covariance matrix $\mathbf{\Sigma}$. Obtaining λ_{max} from $\mathbf{\Sigma}$ can be expensive as the number of relevance vectors grows. Under Assumption 1, the entries in the feature matrix $\mathbf{\Phi}$ for relevance vectors that are far from each other go to 0 quickly and can be set to zero, e.g., using a cut-off threshold for the kernel values or only keeping the kernel values for the K nearest relevance vectors. This leads to a sparse matrix $\mathbf{\Phi}$ and, in turn, the inverse covariance matrix $\mathbf{\Sigma}^{-1}$ in Eq. (2.53) is sparse and its smallest eigenvalue $1/\lambda_{max}$ can be approximated efficiently (e.g. [92]).

3.6.2 Storage Improvements

Algorithm 2 returns a set of M relevance vectors $\mathbf{x_m}$ with labels y_m and weight prior precision ξ_m. This set represents the RVM model parameters and its memory requirements



(a) Ground truth map and robot trajectory.

(b) Set of 2141 relevance vectors.

(c) The final SBKM map.

Figure 3.3. Sparse map representation (with $\eta = 1, \Gamma = \sqrt{\gamma} I, \gamma = 3.0$) built from local streaming laser scans along the robot trajectory.

are linear in M. However, the predictive distribution in Eq. (2.56) needs to be obtained via Laplace approximation (Eq. (2.54) and (2.53)) when the RVM model is used for classification. If additional computation for Laplace approximation is not feasible during test time, the weight posterior mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ may be stored also but $\boldsymbol{\Sigma}$ requires $O(M^2)$ storage. Fortunately, the approximate decision boundary $G_2(\mathbf{x}) = 0$ in Prop. 1 used for point, line, and curve classification only requires the largest eigenvalue λ_{max} of $\boldsymbol{\Sigma}$. Hence, only the value of λ_{max} needs to be stored in addition to the relevance vectors $\mathbf{x}_m$, labels y_m, and weight mean μ_m. In this case, line 15 in Algorithm 2 should return the weight posterior mean $\boldsymbol{\mu}$ and λ_{max} instead of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

3.7 Evaluation

This section presents an evaluation of our autonomous mapping and navigation method using a fully actuated robot (e.g., Eq. (5.1)) in a simulated warehouse environment (Section 3.7.1) and the Intel Research Lab dataset [72] (Section 3.7.2). We examined the obstacle boundary with respect to the bias parameter b and the threshold e in Section 3.7.3. We used a radial basis function (RBF) kernel with parameters $\eta = 1$ and $\Gamma = \sqrt{\gamma}\mathbf{I}$. The bias parameter b

Table 3.1. Comparison among our sparse Bayesian kernel-based map (SBKM), our sparse kernel-based map (SKM) [39], OctoMap (OM) [70], and sequential Bayesian Hilbert map (SBHM) [150].

Methods	Kernel param. γ	Threshold $\bar{e}$	Accuracy	Recall	Vectors/Nodes	Storage
SBKM	1.0	0.5	97.8%	97.9%	1115	9 kB†
SBKM	2.0	0.5	99.0%	99.3%	1642	13 kB†
SBKM	3.0	0.45	99.2%	99.7%	2141	17 kB†
SBKM	3.0	0.5	99.3%	99.4%	2141	17 kB†
SBKM	3.0	0.55	99.5%	98.7%	2141	17 kB†
SKM	3.0	-	99.9%	99.0%	2463	20 kB
SKM	2.0	-	99.8%	98.3%	2613	21 kB
SKM	1.0	-	99.8%	98.5%	3064	25 kB
OM	-	0.5	99.9%	99.7%	12432 non-leafs 34756 leafs	25 kB (bin.) 236 kB (full)
SBHM	1.0	0.5	97.0%	98.0%	1156	9 kB‡
SBHM	2.0	0.5	99.0%	99.4%	1676	13 kB‡
SBHM	3.0	0.45	98.6%	99.0%	2205	17 kB‡
SBHM	3.0	0.5	99.0%	98.6%	2205	17 kB‡
SBHM	3.0	0.55	99.5%	98.2%	2205	17 kB‡

is set to -0.05 in Section 3.7.1, and 0.0 in Section 3.7.2. Timing results are reported from an Intel i9 3.1 GHz CPU with 32GB RAM.

3.7.1 Comparison with Binary Map Representations

In this section, we compared the accuracy, the recall and storage requirements of our sparse Bayesian kernel-based map (SBKM) with those of the non-Bayesian sparse kernel-based map (SKM) from our preliminary work [39], the popular occupancy mapping algorithm OctoMap [70], and the sequential Bayesian Hilbert map (SBHM) [150]. Since SKM only provides binary maps, binary maps are used to calculate accuracy and recall. As the ground-truth map (Figure 3.3a) represents the work space instead of C-space, a point robot ($r = 0$) was used for an accurate comparison. Lidar scans were simulated along the robot trajectory shown in Figure 3.3a and used to build our sparse Bayesian kernel-based map (SBKM), the non-Bayesian sparse kernel-based map (SKM) [39], OctoMap, and sequential Bayesian Hilbert map (SBHM). An R^*-tree approximation of the score $F(\mathbf{x})$ was used with $K^+ + K^- = 200$ nearest support vectors around the robot location $\mathbf{p}_k$ for map updating and with $K^+ + K^- = 10$ nearest support

vectors for collision checking. OctoMap's resolution was set to 0.25 m to match that of the grid used to sample our training data from. As SBHM depends on a grid of hinge points to generate feature vectors, we chose the grid's resolution so that the number of hinge points is similar to our SBKM's number of relevance vectors for a fair comparison.

Table 3.1 compares the accuracy, the recall and the storage requirements of our SBKM maps, our SKM maps, SBHM maps [150] and OctoMap's binary and probabilistic maps. The SBKM map and its sparse set of relevance vectors are shown in Figure 3.3. To calculate map accuracy, we used different thresholds $\bar{e}$ to generate binary versions of our map and compare with the ground truth. The ground truth map was sampled on a grid with the same resolution 0.25 m and the accuracy was calculated as the number of correct predictions divided by the total number of samples. Note that the interior (gray regions in the ground-truth map) of the obstacles were considered occupied for our map - since it was surrounded by positive relevance vectors - but were considered free in SKM and OctoMap maps.

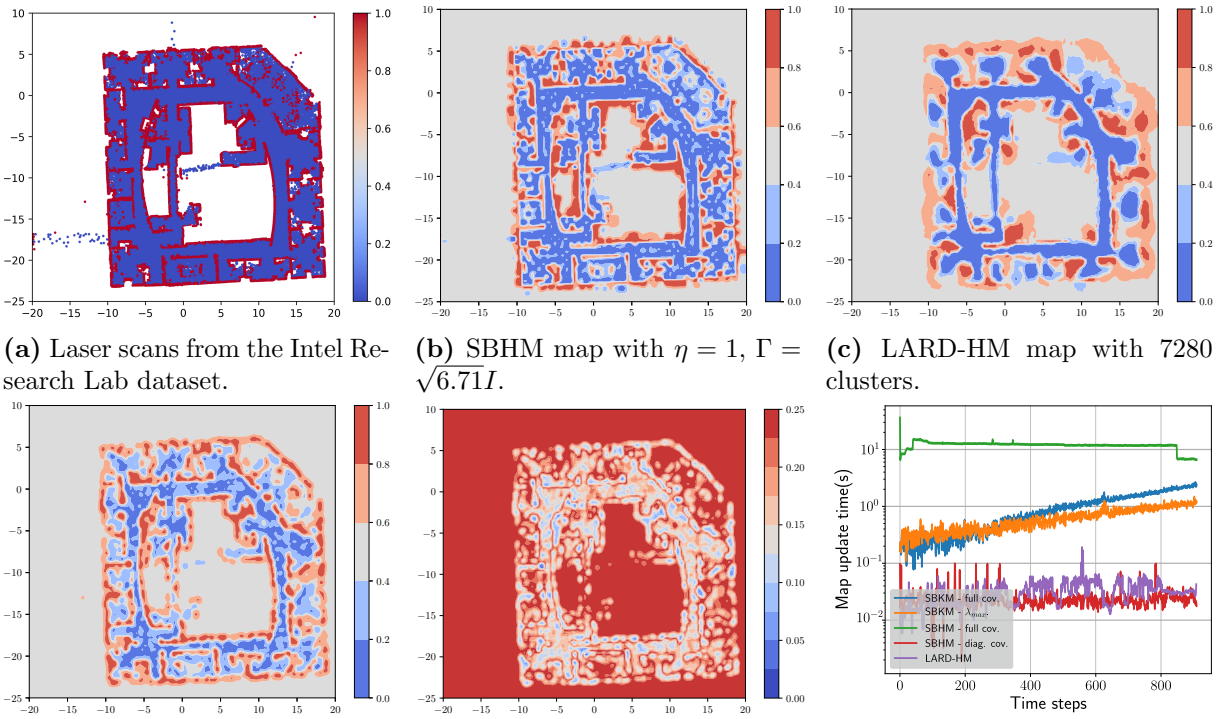
Table 3.1 shows that SBKM (with threshold $\bar{e} = 0.5$), SKM, OctoMap's binary map, and SBHM (with threshold $\bar{e} = 0.5$) led to a similar accuracy of $\sim 99\%$ ($\gamma = 2.0$ & 3.0) and $\sim 97\%$ ($\gamma = 1.0$) and a similar recall of $\sim 99\%$ ($\gamma = 2.0\&3.0$) and $\sim 98\%$ ($\gamma = 1.0$). SKM has $\sim 0.5\%$ higher accuracy than SBKM since the support vectors lie around the obstacle boundary, leading to sharper decision boundaries. As the decision threshold $\bar{e}$ decreases, the accuracy decreases, because more free cells are classified as "occupied", and the recall increases, because more occupied cells are classified as "occupied". When the parameter γ decreases, the support of the kernel expanded, leading to fewer relevance vectors, i.e., less storage but lower accuracy and recalls. This illustrates the trade-off between storage gains and accuracy when the details of the obstacles' boundaries can be reduced via a lower value of γ to achieve higher compression rate.

We compared the storage requirements for our SBKM and SKM representations and OctoMap. OctoMap's binary map required a compressed octree with 12432 non-leaf nodes with 2 bytes per node, leading to a storage requirement of ~ 25 kB. Its fully probabilistic map required to store 47188 leaf and non-leaf nodes with 5 bytes per node, leading to a storage requirement of ~ 236 kB. As the space consumption depends on the computer architecture and

how the relevance vector information is compressed, we provide only a rough estimate of storage requirements for our maps. For the SKM map, each support vector required 8 bytes, including an integer for the support vector's location on the underlying grid and a float for its weight. As a result, $\sim 20\ kB$ were needed to store the 2463 resulting support vectors for $\gamma = 3.0$. As discussed in Section 3.6.2, the SBKM map could be stored in two ways: 1) the relevance vectors' location, their label and their weight prior precision if Laplace approximation was allowed at test time; 2) the relevance vectors' location, their label, their weight mean and the largest eigenvalue λ_{max} of the covariance matrix Σ if Laplace approximation was not allowed at test time and our collision checking methods were used. The former stored an integer representing a relevance vector's location on the underlying grid and a float representing its weight prior's precision and its label (using the float sign). This required 8 bytes on a 32-bit architecture per relevance vector. Our SBKM map with $\Gamma = \sqrt{3.0}I$ contained 2141 relevance vectors, leading to storage requirements of $\sim 17\ kB$. The latter also needed 17 kB to store the relevance vectors' location, their weight's mean and label. Besides, an extra float (4 bytes) is needed to store λ_{max} leading to a similar total storage requirement of 17 kB. These requirements were 32% and 15% better than those of OctoMap and our (non-Bayesian) SKM, respectively. To achieve a sparse Bayesian map representation, more computation is needed, leading to slower map update for SBKM compared to SKM and OctoMap. Since SBHM and SBKM are both Bayesian online mapping methods and share similar settings, we compared their map update time in Section 3.7.2. As γ decreases, the number of relevance vectors of SBKM decreases while the number of support vectors of SKM increases. This is because the relevance vectors spread out in the environment while the support vectors are placed on both sides of the obstacle boundaries. Therefore, as γ decreases, a relevance vector represents more space leading to fewer relevance vectors in the SBKM models and more support vectors, maintaining a sharp decision boundary, for the SKM models.

3.7.2 Comparison with Probabilistic Map Representations

In this section, we compared our sparse Bayesian kernel-based map (SBKM) approach with other probabilistic occupancy mapping techniques: OctoMap [70], localized automatic relevance determination Hilbert map (LARD-HM) [58], and sequential Bayesian Hilbert map



(a) Laser scans from the Intel Research Lab dataset.

(b) SBHM map with $\eta = 1$, $\Gamma = \sqrt{6.71}I$.

(c) LARD-HM map with 7280 clusters.

(d) Our SBKM map with $\eta = 1$, $\Gamma = \sqrt{6.71}I$.

(e) Our SBKM variance with $\eta = 1$, $\Gamma = \sqrt{6.71}I$.

(f) Comparison of map update time per scan.

Figure 3.4. Our sparse Bayesian kernel-based map (SBKM) versus LARD-HM [58] and SBHM [150] on the Intel Research Lab dataset [72].

(SBHM) [150]. The Intel Research Lab dataset [72] was used with the four methods to build the map of the environment in an online manner. Both SBKM and SBHM are kernel-based Bayesian probabilistic mapping methods from streaming local observations. While SBHM achieves sparseness by calculating feature vectors based on a sparse set of hinged points, e.g., on a coarse grid, our SBKM method learns the hinged points, i.e. the relevance vectors, from data. We tried our best to match the parameters for a fair comparison, e.g. using the same kernel parameter $\Gamma = \sqrt{\gamma}\mathbf{I}$ with $\gamma = 6.71$ as provided by SBHM code [150]. Our online training data (Section 3.3.2) were generated from a grid with resolution 0.2 m. Meanwhile, the LARD-HM method determines the hinged points by clustering the training data points using k-means algorithms and calculates their kernel parameters by automatic relevance determination. To build a LARD-HM map from streaming depth measurements, we used the publicly available LARD-HM code in [58] to incrementally add the centroids of the clusters and their kernel parameters to the maps. OctoMap code [70] was used with its default parameters. Since we only considered probabilistic

Table 3.2. Comparison among our sparse SBKM map, SBHM map [150], LARD-HM map [58] and OctoMap [70] on the Intel Research Lab dataset [72].

Test data	Metrics	SBHM	SBHM	LARD-HM	LARD-HM	LARD-HM	SBKM	SBKM	OM
-	γ	6.71	6.71	-	-	-	6.71	6.71	-
-	$\mathbf{\Sigma}$	full	diag.	-	-	-	full	λ_{max} only	-
-	Feature dim.	5600	5600	3640	5460	7280	3492	3492	-
Uniformly sampled	AUC	0.98	0.98	0.90	0.96	0.97	0.96	0.95	0.95
Uniformly sampled	NLL	0.24	0.24	0.41	0.30	0.24	0.36	0.52	0.27
Near boundary	AUC	0.82	0.77	0.57	0.55	0.56	0.62	0.61	0.75
Near boundary	NLL	0.54	0.60	0.83	0.83	0.78	0.73	0.84	0.67
-	Update time/scan	11.8 s	0.03 s	0.03 s	0.03 s	0.03 s	0.76 s	0.43 s	0.01 s

occupancy maps in this section, the metrics for comparison were the area under the receiver operating characteristic curve (AUC) and the negative log-likelihood loss (NLL) of a point $\mathbf{x}$, defined as $NLL(y|\mathbf{x}, \boldsymbol{\xi}) = -\log p(y|\mathbf{x}, \boldsymbol{\xi})$, where $y \in \{-1, 1\}$ is the true label and $p(y|\mathbf{x}, \boldsymbol{\xi})$ is the predictive distribution in Eq. (2.56). The AUC score and NLL loss were calculated over two test sets: one sampled uniformly from the whole dataset to capture overall reconstruction accuracy and one sampled near the room area boundaries to capture detail reconstruction accuracy.

Table 3.2 presents the metrics for the four mapping methods. SBHM used a fixed grid of 5600 hinged points with resolution 0.5 m for feature vector calculation. Meanwhile, our approach incrementally learned a sparse set of 3492 relevance vectors from the training dataset, not requiring a set of fixed hinged points. Similarly, LARD-HM determined the hinged points from the data by incrementally adding the centroids of k_f free and k_o occupied clusters and their kernel parameters from each laser scan, where $k_f = k_o = 2, 3, 4$ leading to 3640 (similar to the SBKM's feature dimension), 5460 (similar to the SBHM's feature dimension) and 7280 hinged points in Table 3.2, respectively. Figure 3.4b, 3.4c and 3.4d show the final maps from the SBHM, LARD-HM approaches and our SBKM method, respectively. Figure 3.4e plots the our SBKM map's variance, distinguishing between known (low variance) and unknown (high variance) regions.

Our final map's AUC score and NLL loss were slightly worse than those of SBHM with full covariance matrix while maintaining $\sim 35\%$ fewer points to represent the environment and having faster map updates with less than 1 s per scan, on average, as shown in Table 3.2 and Figure 3.4f. Our training algorithm incrementally built the set of relevance vectors and only updated the weights of the local vectors due to the use of K nearest relevance vectors in Algorithm 2. Consequently, it did not have a fixed global set of points to optimize over as done by SBHM, leading to suboptimality in trade-off for sparseness. Note that both our map and the SBHM map estimated the mean $\boldsymbol{\mu}$ and the full covariance matrix $\boldsymbol{\Sigma}$ of the weights' posterior for test time. If our collision checking algorithms are used for planning, only the largest eigenvalue λ_{max} of $\boldsymbol{\Sigma}$ is needed and can be calculated efficiently using the sparsified inverse covariance matrix as shown in Section 3.6.2. In this case, Table 3.2 shows that our map update time was reduced by half to about ~ 0.43 s per scan (Table 3.2 and Figure 3.4f) while offering similar AUC

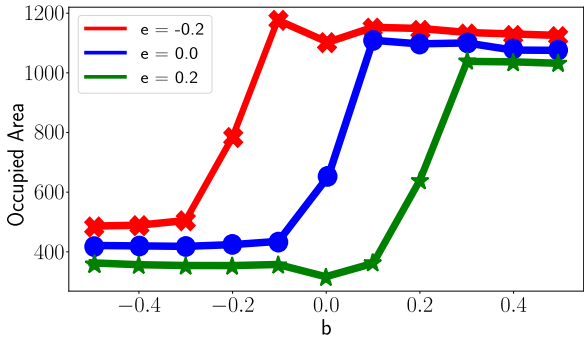
score to that of our SBKM map with full covariance matrix. The higher NLL loss was due to the upper bound used in Prop. 1 for point classification instead of the true occupancy probability. A variant of SBHM that uses diagonal covariance matrix updated the map 25 times faster than our SBKM method with full covariance matrix. While our SBKM time can be improved further using a diagonal covariance matrix, we leave this investigation for future work.

Our map's AUC score and NLL loss, calculated using test data sampled uniformly from the dataset, were better than those of LARD-HM with 3640 features (similar to our map's feature size 3492), comparable to those of LARD-HM with 5460 features, and worse than those of LARD-HM with 7280 features, shown in Figure 3.4c. Our map update time was 14 times (with λ_{max} only) and 25 times (with full covariance matrix) slower. The main reason for the slower speed is that our method uses Bayesian updates with a full posterior covariance matrix while LARD-HM is not Bayesian. The speed of our method can be accelerated by using a diagonal-only covariance matrix formulation or learning different kernel parameters from data using the key ideas in LARD-HM. With similar number of features, LARD-HM tended to preserve less details of the obstacle boundary compared to our method in the room areas of the Intel Lab dataset, as shown in Figure 3.4c and in Table 3.2 by the better AUC score and NLL loss of our map when the test set was sampled in the rooms near the boundary. This reflects the difference that our SBKM approach adds relevance vectors, which can be very close to the boundary, while LARD-HM maps chooses cluster centroids, which intuitively are farther from the boundary.

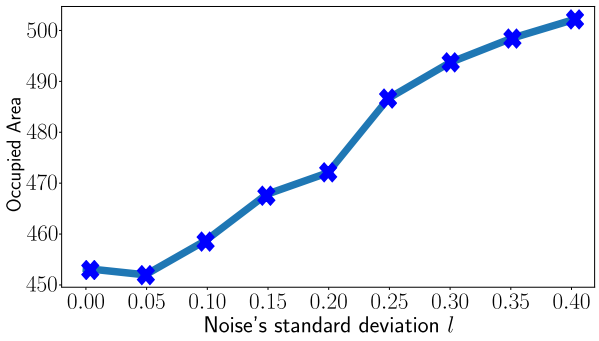
Octomap's AUC score, calculated from test points sampled uniformly from the complete dataset, was lower than ours as the default maximum (0.97) and minimum (0.12) values of the occupancy probability were used. Meanwhile, OctoMap's performance was better than ours in preserving boundary details in the room areas. A feature dimension is not reported for OctoMap since it is not a kernel-based method like SBHM, SBKM, and LARD-HM.

3.7.3 Decision Boundary's Conservativeness

As the decision boundary between free and occupied spaces affects the area for robot navigation, we examined its conservativeness with respect to the bias b, threshold e, and measurement noise variance l^2 parameters. Figure 3.5a plots the occupied area in our map, i.e.,



(a) The occupied area (with different values of e) versus the bias b.



(b) The occupied area (with $b = -0.01, e = 0.0$) versus the noise's standard deviation.

Figure 3.5. Occupied area versus the bias b and the threshold e (a) and versus noise level (b).

the area with occupancy probability greater than the threshold $e = 0$, built from the Intel Lab dataset [72] for different values of the bias b and threshold e. As expected, the occupied area increases, i.e., smaller navigable area, if e decreases and/or b increases. Note that our previous SKM model [39] does not offer similar tuning for the decision boundary because it does not provide parameters such as b and e.

The decision boundary's conservativeness should also be affected by the measurement noise since a robot should proceed carefully around the obstacle boundary if the depth measurements are noisy. To illustrate the conservativeness of the boundary generated by our approach against measurement noise, we added Gaussian noise with zero mean and variance l^2 to the laser endpoints in the dataset and trained our model. The occupied area versus the noise's standard deviation l is plotted in Figure 3.5b. As the noise level l increases, the occupied area increases, i.e., the navigable area decreases, making the robot more cautious around obstacles in the environment.

3.8 Summary

This chapter proposes a sparse Bayesian kernel-based mapping method for efficient online generation of large occupancy maps, supporting autonomous robot navigation in unknown environments. Our map representation, as a sparse set of relevance vectors learned from streaming range observations of the environment, is efficient to store. Our experiments demonstrate the potential of this model at generating compressed, yet accurate, probabilistic environment

models. Our results offer a promising venue for quantifying safety and uncertainty and enabling real-time long-term autonomous navigation in unpredictable environments.

Acknowledgements

Chapter 3, in part, is a reprint of the material as it appears in T. Duong, M. Yip, N. Atanasov, "*Autonomous Navigation in Unknown Environments with Sparse Bayesian Kernel-based Occupancy Mapping*", IEEE Transactions on Robotics (T-RO), vol. 38, no. 6, pp. 3694-3712, 2022, in T. Duong, N. Das, M. Yip, N. Atanasov, "*Autonomous Navigation in Unknown Environments using Sparse Kernel-based Occupancy Mapping*", International Conference on Robotics and Automation (ICRA), pp. 9666-9672, 2020. The dissertation author was the primary investigator and author of these papers.

Chapter 4

Learning Hamiltonian Dynamics on Lie Groups

Motion planning and optimal control algorithms are important components of any autonomous navigation framework and depend on the availability of accurate system dynamics models. Models obtained from first principles and calibrated over a small set of parameters via system identification [102] are widely used for unmanned ground vehicles (UGVs), unmanned aerial vehicles (UAVs), and unmanned underwater vehicles (UUVs). Such models may oversimplify or even incorrectly describe the underlying structure of the dynamical system, leading to bias and modeling errors that cannot be corrected by adjusting a few parameters. Data-driven techniques [178, 136, 24, 159, 144, 147, 104, 74] have emerged as a powerful approach to approximate system dynamics with an over-parameterized machine learning model, trained over a dataset of system state and control trajectories. Neural networks are expressive function approximation models, capable of identifying and generalizing interaction patterns from the training data. Training neural network models, however, typically requires large amounts of data and computation time, which may be impractical in mobile robotics applications. Recent works [110, 60, 27, 55, 23, 140, 106, 123] have considered a hybrid approach, where prior knowledge of the physics, governing the system dynamics, is used to assist the learning process. The dynamics of physical systems obey kinematic constraints and energy conservation laws. These laws are known to be universally true but a black-box machine learning model might struggle to infer them from the training data, causing poor generalization. Instead, prior knowledge may be encoded into the learning model, e.g., using a prior distribution [33], a graph-network forward kinematic

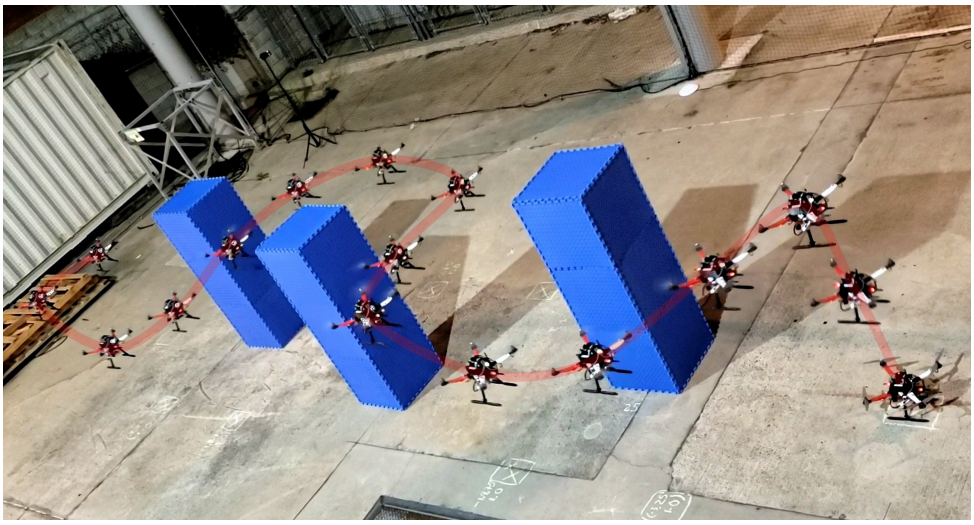


Figure 4.1. A quadrotor tracking a trajectory using our learned model and avoid obstacles.

model [145], symmetry [143, 174], Lagrangian mechanics [140, 111, 60, 27, 109, 110, 106] or Hamiltonian mechanics [55, 10, 23, 46, 184, 177, 123, 8, 7] guarantee that the laws of physics are satisfied by construction, regardless of the training data.

Sanchez-Gonzalez et al. [145] design graph neural networks to represent the kinematic structure of complex dynamical systems and demonstrate forward model learning and online planning via gradient-based trajectory optimization. Ruthotto et al. [143] propose a partial differential equation (PDE) interpretation of convolutional neural networks and derive new parabolic and hyperbolic ResNet architectures guided by PDE theory. Wang et al. [174] design symmetry equivariant neural network models, encoding rotation, scaling, and uniform motion, to learn physical dynamics that are robust to symmetry group distributional shifts.

Lagrangian-based methods [140, 111, 60, 27, 109, 110, 106] design neural network models for physical systems based on the Euler-Lagrange differential equations of motion [108, 69], in terms of generalized coordinates $\mathbf{q}$, their velocity $\dot{\mathbf{q}}$ and a Lagrangian function $\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}})$, defined as the difference between the kinetic and potential energies. The energy terms are modeled by neural networks, either separately [111, 109, 110] or together [27].

Hamiltonian-based methods [55, 10, 23, 46, 184, 177, 123] use a Hamiltonian formulation [108, 69] of the system dynamics, instead, in terms of generalized coordinates $\mathbf{q}$, generalized momenta $\mathfrak{p}$, and a Hamiltonian function, $\mathcal{H}(\mathbf{q}, \mathfrak{p})$, representing the total energy of the system.

Greydanus et al. [55] model the Hamiltonian as a neural network and update its parameters by minimizing the discrepancy between its symplectic gradients and the time derivatives of the states $(\mathbf{q}, \mathbf{p})$. This approach, however, requires that the state time derivatives are available in the training data set. Chen et al. [23], Zhong et al. [184] relax this assumption by using differentiable leapfrog integrators [93] and differentiable ODE solvers [21], respectively. The need for time derivatives of the states is eliminated by back-propagating a loss function measuring state discrepancy through the ODE solvers via the adjoint method. Our work extends the approach in [184, 185] by formulating the Hamiltonian dynamics over a matrix Lie group, which enforces kinematic constraints in the neural ODE network used to learn the dynamics. Toth et al. [168] and Mason et al. [117] show that, instead of from state trajectories, the Hamiltonian function can be learned from high-dimensional image observations. Finzi et al. [46] show that using Cartesian coordinates with explicit constraints improves both the accuracy and data efficiency for the Lagrangian- and Hamiltonian-based approaches. In a closely related work, Zhong et al. [185] showed that dissipating elements, such as friction or air drag, can be incorporated in a Hamiltonian-based neural ODE network by reformulating the system dynamics in port-Hamiltonian form [170]. The continuous-time equations of motions in Lagrangian or Hamiltonian dynamics can also be discretized using variational integrators [115] to learn discrete-time Lagrangian and Hamiltonian systems [67, 41, 20, 158] and provide long-term prediction for control methods such as model predictive control [14]. This approach eliminates the need to use an ODE solver to roll out the dynamics but its prediction accuracy depends on the discretization time step.

Lagrangian and Hamiltonian mechanics [108, 69] provide physical system descriptions that can be integrated into the structure of a neural network [55, 10, 23, 46, 184, 185, 177, 111, 109]. Meanwhile, many physical robot platforms are composed of rigid-body interconnections and their state evolution respects the structure of a Lie group [62], e.g., the position and orientation kinematics of a rigid body evolve on the $SE(3)$ Lie group [112]. Prior work, however, has only considered vector-valued states, when designing Lagrangian- or Hamiltonian-structured neural networks. This limits the applicability of these techniques as many common robot systems have states on a Lie group. For example, Hamiltonian equations of motion are available for

orientation but existing formulations rely predominantly on 3 dimensional vector parametrizations, such as Euler angles [113, 154], which suffer from singularities. The goal of this chapter is to incorporate both the kinematic structure and the energy conservation properties of physical systems with Lie group states into the structure of a dynamics learning model.

In our preliminary work [37], we design a neural ordinary differential equation (ODE) network [21], whose structure captures Hamiltonian dynamics over the $SE(3)$ manifold [91]. Our model guarantees by construction that long-term trajectory predictions satisfy $SE(3)$ constraints and conserve total energy. Inspired by [184, 185], we model kinetic energy and potential energy by separate neural networks, each governed by a set of Hamiltonian equations on $SE(3)$. The Hamiltonian formulation can be generalized to a Port-Hamiltonian one [170], enabling us to design an energy-based controller for trajectory tracking. Our preliminary work [37] is developed specifically for the $SE(3)$ manifold and does not consider dissipation elements that drain energy from the system, such as friction or drag forces. In this chapter, we generalize our port-Hamiltonian neural ODE network to embed general matrix Lie group constraints and introduce an energy dissipation term, represented by another neural network, to model friction and air drag in physical systems [36]. We verify our approach extensively with simulated robot systems. In summary, this chapter makes the following *contributions*.

- We design a neural ODE model that respects port-Hamiltonian dynamics over a matrix Lie group to enable data-driven learning of rigid-body system dynamics.
- We demonstrate our dynamics learning approach with various robot systems (a pendulum and a quadrotor).

4.1 Dynamics Learning Problem

In this section, we restate the problem of learning robot dynamics from data (Problem 2) but for robot states on a Lie group manifold. Consider a robot with state $\mathbf{s}$ consisting of generalized coordinates $\mathbf{q}$ evolving on a Lie group G and generalized velocity $\boldsymbol{\xi}$ on the Lie algebra $\mathfrak{g}$ of G. Let $\dot{\mathbf{s}} = \mathbf{f}(\mathbf{s}, \mathbf{u})$ characterize the robot dynamics with control input $\mathbf{u} \in \mathbb{R}^m$. For example, the state of rigid-body mobile robot, such as a UGV or UAV, may be modeled by its pose on

the $SE(3)$ group, consisting of position and orientation, and its twist on the $\mathfrak{se}(3)$ Lie algebra, consisting of linear and angular velocity. The control input of an Ackermann-drive UGV may include its linear acceleration and steering angle rate, and that of a quadrotor UAV may include the total thrust and moment generated by the propellers.

We assume that the function $\mathbf{f}$ specifying the robot dynamics is unknown and aim to approximate it using a dataset $\mathcal{D}$ of state and control trajectories. Specifically, let $\mathcal{D} = \{t^{(i)}_{0:N}, \mathbf{s}^{(i)}_{0:N}, \mathbf{u}^{(i)}\}_{i=1}^{D}$ consist of D state sequences $\mathbf{s}^{(i)}_{0:N}$, obtained by applying a constant control input $\mathbf{u}^{(i)}$ to the system with initial condition $\mathbf{s}^{(i)}_0$ at time $t^{(i)}_0$ and sampling its state $\mathbf{s}^{(i)}(t^{(i)}_n) =: \mathbf{s}^{(i)}_n$ at times $t^{(i)}_0 < t^{(i)}_1 < \ldots < t^{(i)}_N$. Using the dataset $\mathcal{D}$, we aim to find a function $\bar{\mathbf{f}}_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$ that approximates the true dynamics $\mathbf{f}$ well. To optimize $\boldsymbol{\theta}$, we roll out the approximate dynamics $\bar{\mathbf{f}}_{\boldsymbol{\theta}}$ with initial state $\mathbf{s}^{(i)}_0$ and constant control $\mathbf{u}^{(i)}$ and minimize the discrepancy between the computed state sequence $\bar{\mathbf{s}}^{(i)}_{1:N}$ and the true state sequence $\mathbf{s}^{(i)}_{1:N}$ in $\mathcal{D}$.

Problem 5. Given a dataset $\mathcal{D} = \{t^{(i)}_{0:N}, \mathbf{s}^{(i)}_{0:N}, \mathbf{u}^{(i)}\}_{i=1}^{D}$ and a function $\bar{\mathbf{f}}_{\boldsymbol{\theta}}$, find the parameters $\boldsymbol{\theta}$ that minimize:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \; & \sum_{i=1}^{D} \sum_{n=1}^{N} \ell(\mathbf{s}^{(i)}_n, \bar{\mathbf{s}}^{(i)}_n) \\ \text{s.t.} \; & \dot{\bar{\mathbf{s}}}^{(i)}(t) = \bar{\mathbf{f}}_{\boldsymbol{\theta}}(\bar{\mathbf{s}}^{(i)}(t), \mathbf{u}^{(i)}), \;\; \bar{\mathbf{s}}^{(i)}(t_0) = \mathbf{s}^{(i)}_0, \\ & \bar{\mathbf{s}}^{(i)}_n = \bar{\mathbf{s}}^{(i)}(t_n), \;\; \forall n = 1, \ldots, N, \;\; \forall i = 1, \ldots, D, \end{aligned} \tag{4.1}$$

where ℓ is a distance metric on the state space.

We consider robot kinematics on the Lie group G such that when there is no control input, $\mathbf{u} = \mathbf{0}$, the dynamics $\mathbf{f}(\mathbf{s}, \mathbf{u})$ respect the law of energy conservation. We embed these constraints in the structure of the parametric function $\bar{\mathbf{f}}_{\boldsymbol{\theta}}$.

4.2 Learning Hamiltonian Dynamics on Matrix Lie Groups

We consider a Hamiltonian system with unknown kinetic energy $\mathcal{T}(\mathbf{q})$, potential energy $\mathcal{V}(\mathbf{q})$, input matrix $\mathbf{B}(\mathbf{q})$, dissipation matrix $\mathbf{D}(\mathbf{q}, \mathfrak{p})$, and design a structured neural ODE network to learn these terms from state-control trajectories.

4.2.1 Data Collection

We collect a data set $\mathcal{D} = \{t^{(i)}_{0:N}, \mathbf{s}^{(i)}_{0:N}, \mathbf{u}^{(i)}\}_{i=1}^{D}$ consisting of state sequences $\mathbf{s}^{(i)}_{0:N}$, where $\mathbf{x}^{(i)}_n = [\mathbf{q}^{(i)\top}_n \quad \boldsymbol{\xi}^{(i)\top}_n]^\top$ for $n = 0, \ldots, N$. Such data are generated by applying a constant control input $\mathbf{u}^{(i)}$ to the system and sampling the state $\mathbf{s}^{(i)}_n = \mathbf{s}^{(i)}(t^{(i)}_n)$ at times $t^{(i)}_n$ for $n = 0, \ldots, N$. The generalized coordinates $\mathbf{q}$ and velocity $\boldsymbol{\xi}$ may be obtained from a state estimation algorithm, such as odometry algorithm for mobile robots [34, 118], or from a motion capture system. In physics-based simulation the data can be generated by applying random control inputs $\mathbf{u}^{(i)}$. In real-world applications, where safety is a concern, data may be collected by a human operator manually controlling the robot.

4.2.2 Model Architecture

Since robots are physical systems, their dynamics $\mathbf{f}(\mathbf{s}, \mathbf{u})$ satisfy the Hamiltonian formulation (Section 2.2.2). To learn the dynamics $\mathbf{f}(\mathbf{s}, \mathbf{u})$ from a trajectory dataset $\mathcal{D}$, we design a neural ODE network (Section 2.3), approximating the dynamics via a parametric function $\bar{\mathbf{f}}_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{u})$ based on Eq. (2.27).

To integrate the Hamiltonian equations into the structure of $\bar{\mathbf{f}}_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{u})$, we use four neural networks with parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\mathcal{T}}, \boldsymbol{\theta}_{\mathcal{V}}, \boldsymbol{\theta}_{\mathbf{D}}, \boldsymbol{\theta}_{\mathbf{B}})$ to approximate the kinetic energy by $\mathcal{T}_{\boldsymbol{\theta}}(\mathbf{q}, \boldsymbol{\xi})$, the potential energy by $\mathcal{V}_{\boldsymbol{\theta}}(\mathbf{q})$, the dissipation matrix by $\mathbf{D}_{\boldsymbol{\theta}}(\mathbf{q}, \mathfrak{p})$, and the input matrix by $\mathbf{B}_{\boldsymbol{\theta}}(\mathbf{q})$, respectively. Since the generalized momenta $\mathfrak{p}$ are not directly available in $\mathcal{D}$, the time derivative of the generalized velocity $\boldsymbol{\xi}$ is obtained from Eq. (2.28). The approximated dynamics function $\bar{\mathbf{f}}_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{u})$ is described with an internal state $\mathfrak{p}$ as follows:

$$\dot{\mathbf{q}} = \mathsf{T}_{\mathbf{e}}\mathsf{L}_{\mathbf{q}}\left(\frac{\partial \mathcal{H}_{\boldsymbol{\theta}}(\mathbf{q}, \mathfrak{p})}{\partial \mathfrak{p}}\right), \tag{4.2a}$$

$$\dot{\mathfrak{p}} = \mathsf{ad}^*_{\boldsymbol{\xi}}(\mathfrak{p}) - \mathbf{D}_{\boldsymbol{\theta}}(\mathbf{q}, \mathfrak{p})\frac{\partial \mathcal{H}_{\boldsymbol{\theta}}(\mathbf{q}, \mathfrak{p})}{\partial \mathfrak{p}} - \mathsf{T}^*_{\mathbf{e}}\mathsf{L}_{\mathbf{q}}\left(\frac{\partial \mathcal{H}_{\boldsymbol{\theta}}(\mathbf{q}, \mathfrak{p})}{\partial \mathbf{q}}\right) + \mathbf{B}_{\boldsymbol{\theta}}(\mathbf{q})\mathbf{u}, \tag{4.2b}$$

$$\dot{\boldsymbol{\xi}} = \frac{d}{dt}\frac{\partial \mathcal{H}_{\boldsymbol{\theta}}(\mathbf{q}, \mathfrak{p})}{\partial \mathfrak{p}}, \tag{4.2c}$$

where the Hamiltonian is $\mathcal{H}_{\boldsymbol{\theta}}(\mathbf{q}, \mathfrak{p}) = \mathfrak{p} \cdot \boldsymbol{\xi} - \mathcal{L}_{\boldsymbol{\theta}}(\mathbf{q}, \boldsymbol{\xi})$, and the Lagrangian is $\mathcal{L}_{\boldsymbol{\theta}}(\mathbf{q}, \boldsymbol{\xi}) = \mathcal{T}_{\boldsymbol{\theta}}(\mathbf{q}, \boldsymbol{\xi}) - \mathcal{V}_{\boldsymbol{\theta}}(\mathbf{q})$. The time derivative $\frac{d}{dt}\frac{\partial \mathcal{H}_{\boldsymbol{\theta}}(\mathbf{q}, \mathfrak{p})}{\partial \mathfrak{p}}$ is calculated using automatic differentiation, e.g. by Pytorch [132]. The approximated dynamics function $\bar{\mathbf{f}}_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{u})$ is implemented in a neural ODE

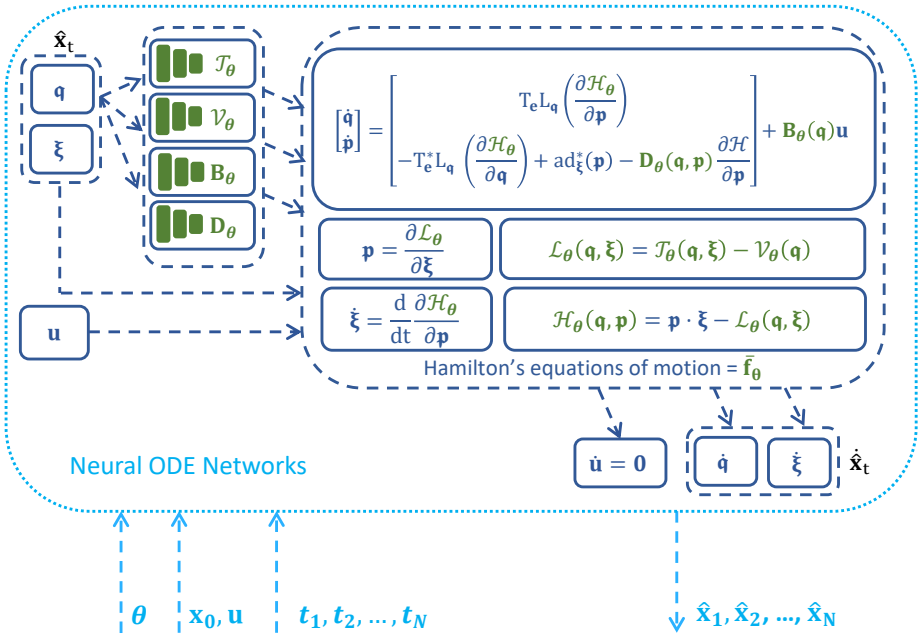


Figure 4.2. Architecture of port-Hamiltonian neural ODE network on a Lie group. The trainable terms are shown in green.

network architecture for training, shown in Figure 4.2.

4.2.3 Training Process

Let $\bar{\mathbf{s}}^{(i)}(t)$ denote the state trajectory predicted with control input $\mathbf{u}^{(i)}$ by the approximate dynamics $\bar{\mathbf{f}}_{\boldsymbol{\theta}}$ initialized at $\bar{\mathbf{s}}^{(i)}(t_0^{(i)}) = \mathbf{s}_0^{(i)}$. For sequence i, forward passes through the ODE solver in (2.41) return the predicted states $\bar{\mathbf{s}}_{0:N}^{(i)}$ at times $t_{0:N}^{(i)}$, where $\bar{\mathbf{s}}_n^{(i)} = [\bar{\mathbf{q}}_n^{(i)\top} \quad \bar{\boldsymbol{\xi}}_n^{(i)\top}]^\top$, for $n = 1, \ldots, N$. The predicted coordinates $\bar{\mathbf{q}}_n^{(i)}$ and the ground-truth ones $\mathbf{q}_n^{(i)}$ are used to calculate a loss on the Lie group manifold:

$$\mathcal{L}_{\mathbf{q}}(\boldsymbol{\theta}) = \sum_{i=1}^{D}\sum_{n=1}^{N}\left\|\log_{\mathsf{G}}^{\vee}\left(\bar{\mathbf{q}}_n^{(i)}\left(\mathbf{q}_n^{(i)}\right)^{-1}\right)\right\|_2^2. \tag{4.3}$$

We use the squared Euclidean norm to calculate losses for the generalized velocity terms:

$$\mathcal{L}_{\boldsymbol{\xi}}(\boldsymbol{\theta}) = \sum_{i=1}^{D}\sum_{n=1}^{N}\|\boldsymbol{\xi}_n^{(i)} - \bar{\boldsymbol{\xi}}_n^{(i)}\|_2^2. \tag{4.4}$$

The total loss $\mathcal{L}(\boldsymbol{\theta})$ is defined as:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\mathbf{q}}(\boldsymbol{\theta}) + \mathcal{L}_{\boldsymbol{\xi}}(\boldsymbol{\theta}). \tag{4.5}$$

The gradient of the total loss function $\mathcal{L}(\boldsymbol{\theta})$ is back-propagated by solving an ODE with adjoint states [21], described in Section 2.3.

4.2.4 Application to SE(3) Hamiltonian Dynamics Learning

This section applies our Lie group Hamiltonian dynamics learning approach to estimate mobile robot dynamics on the $SE(3)$ manifold (Section 2.2.4).

Neural ODE model architecture: When the Hamiltonian dynamics in (2.27) are defined on the $SE(3)$ manifold, the equations of motion become (2.40). The neural ODE network architecture in (4.2) is simplified as follows. We use five neural networks with parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\mathbf{v}}, \boldsymbol{\theta}_{\boldsymbol{\omega}}, \boldsymbol{\theta}_V, \boldsymbol{\theta}_{\mathbf{D}}, \boldsymbol{\theta}_{\mathbf{B}})$ to approximate the blocks $\mathbf{M}^{-1}_{\mathbf{v};\boldsymbol{\theta}}(\mathbf{q})$, $\mathbf{M}^{-1}_{\boldsymbol{\omega};\boldsymbol{\theta}}(\mathbf{q})$ of the inverse generalized mass in (2.35), the potential energy $\mathcal{V}_{\boldsymbol{\theta}}(\mathbf{q})$, the dissipation matrix $\mathbf{D}_{\boldsymbol{\theta}}(\mathbf{q}, \mathbf{p})$, and the input matrix $\mathbf{B}_{\boldsymbol{\theta}}(\mathbf{q})$, respectively. The approximated kinetic energy is calculated as $\mathcal{T}_{\boldsymbol{\theta}}(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\mathbf{p}^\top \mathbf{M}^{-1}_{\boldsymbol{\theta}}(\mathbf{q})\mathbf{p}$, where $\mathbf{M}_{\boldsymbol{\theta}}(\mathbf{q}) = \mathrm{diag}(\mathbf{M}_{\mathbf{v};\boldsymbol{\theta}}(\mathbf{q}), \mathbf{M}_{\boldsymbol{\omega};\boldsymbol{\theta}}(\mathbf{q}))$.

Neural network design: In many applications, nominal information is available about the generalized mass matrices $\mathbf{M}^{-1}_{\mathbf{v};\boldsymbol{\theta}}(\mathbf{q})$, $\mathbf{M}^{-1}_{\boldsymbol{\omega};\boldsymbol{\theta}}(\mathbf{q})$, the potential energy $\mathcal{V}_{\boldsymbol{\theta}}(\mathbf{q})$, the dissipation matrix $\mathbf{D}_{\boldsymbol{\theta}}(\mathbf{q}, \mathbf{p})$, and the input matrix $\mathbf{B}_{\boldsymbol{\theta}}(\mathbf{q})$, and can be included in the neural network design.

Let $\mathbf{M}^{-1}_{\mathbf{v}0}(\mathbf{q})$, $\mathbf{M}^{-1}_{\boldsymbol{\omega}0}(\mathbf{q})$, and $\mathbf{D}_0(\mathbf{q}, \mathbf{p})$ be the nominal values of the generalized mass matrices $\mathbf{M}^{-1}_{\mathbf{v};\boldsymbol{\theta}}(\mathbf{q})$, $\mathbf{M}^{-1}_{\boldsymbol{\omega};\boldsymbol{\theta}}(\mathbf{q})$ and the dissipation matrix $\mathbf{D}_{\boldsymbol{\theta}}(\mathbf{q}, \mathbf{p})$ with Cholesky decomposition:

$$\begin{aligned}
\mathbf{M}^{-1}_{\mathbf{v}0}(\mathbf{q}) &= \mathbf{L}_{\mathbf{v}0}(\mathbf{q})\mathbf{L}^\top_{\mathbf{v}0}(\mathbf{q}), \\
\mathbf{M}^{-1}_{\boldsymbol{\omega}0}(\mathbf{q}) &= \mathbf{L}_{\boldsymbol{\omega}0}(\mathbf{q})\mathbf{L}^\top_{\boldsymbol{\omega}0}(\mathbf{q}), \\
\mathbf{D}_0(\mathbf{q}) &= \mathbf{L}_{\mathbf{D}0}(\mathbf{q})\mathbf{L}^\top_{\mathbf{D}0}(\mathbf{q}).
\end{aligned} \tag{4.6}$$

The learned terms $\mathbf{M}^{-1}_{\mathbf{v};\boldsymbol{\theta}}(\mathbf{q})$, $\mathbf{M}^{-1}_{\boldsymbol{\omega};\boldsymbol{\theta}}(\mathbf{q})$, and $\mathbf{D}_{\boldsymbol{\theta}}(\mathbf{q},\mathfrak{p})$ are obtained using Cholesky decomposition:

$$\begin{aligned}
\mathbf{M}^{-1}_{\mathbf{v};\boldsymbol{\theta}}(\mathbf{q}) &= \left(\mathbf{L}_{\mathbf{v}0}(\mathbf{q}) + \mathbf{L}_{\mathbf{v}}(\mathbf{q})\right)\left(\mathbf{L}_{\mathbf{v}0}(\mathbf{q}) + \mathbf{L}_{\mathbf{v}}(\mathbf{q})\right)^{\top} + \varepsilon_{\mathbf{v}}\mathbf{I}, \\
\mathbf{M}^{-1}_{\boldsymbol{\omega};\boldsymbol{\theta}}(\mathbf{q}) &= \left(\mathbf{L}_{\boldsymbol{\omega}0}(\mathbf{q}) + \mathbf{L}_{\boldsymbol{\omega}}(\mathbf{q})\right)\left(\mathbf{L}_{\boldsymbol{\omega}0}(\mathbf{q}) + \mathbf{L}_{\boldsymbol{\omega}}(\mathbf{q})\right)^{\top} + \varepsilon_{\boldsymbol{\omega}}\mathbf{I}, \\
\mathbf{D}_{\boldsymbol{\theta}}(\mathbf{q},\mathfrak{p}) &= \left(\mathbf{L}_{\mathbf{D}0}(\mathbf{q},\mathfrak{p}) + \mathbf{L}_{\mathbf{D}}(\mathbf{q},\mathfrak{p})\right)\left(\mathbf{L}_{\mathbf{D}0}(\mathbf{q},\mathfrak{p}) + \mathbf{L}_{D}(\mathbf{q},\mathfrak{p})\right)^{\top}
\end{aligned} \tag{4.7}$$

where $\mathbf{L}_{\mathbf{v}}(\mathbf{q})$, $\mathbf{L}_{\boldsymbol{\omega}}(\mathbf{q})$, and $\mathbf{L}_{\mathbf{D}}(\mathbf{q},\mathfrak{p})$ are lower-triangular matrices implemented as three neural networks with parameters $\boldsymbol{\theta}_{\mathbf{v}}$, $\boldsymbol{\theta}_{\boldsymbol{\omega}}$, and $\boldsymbol{\theta}_{\mathbf{D}}$ respectively, and $\varepsilon_{\mathbf{v}}, \varepsilon_{\boldsymbol{\omega}} > 0$.

The potential energy $\mathcal{V}(\mathbf{q})$ and the input matrix $\mathbf{B}(\mathbf{q})$ are implemented with nominal values $\mathcal{V}_0(\mathbf{q})$ and $\mathbf{B}_0(\mathbf{q})$ as follows:

$$\begin{aligned}
\mathcal{V}_{\boldsymbol{\theta}}(\mathbf{q}) &= \mathcal{V}_0(\mathbf{q}) + \mathrm{L}_{\mathcal{V}}(\mathbf{q}), \\
\mathbf{B}_{\boldsymbol{\theta}}(\mathbf{q}) &= \mathbf{B}_0(\mathbf{q}) + \mathbf{L}_{\mathbf{B}}(\mathbf{q}),
\end{aligned} \tag{4.8}$$

where $\mathrm{L}_{\mathcal{V}}(\mathbf{q})$ and $\mathbf{L}_{\mathbf{B}}(\mathbf{q})$ are two neural networks with parameters $\boldsymbol{\theta}_{\mathcal{V}}$ and $\boldsymbol{\theta}_{\mathbf{B}}$, respectively.

Loss function: The orientation loss is calculated as:

$$\mathcal{L}_{\mathbf{R}}(\boldsymbol{\theta}) = \sum_{i=1}^{D}\sum_{n=1}^{N}\left\|\log^{\vee}_{SO(3)}(\bar{\mathbf{R}}_n^{(i)}\mathbf{R}_n^{(i)\top})\right\|_2^2, \tag{4.9}$$

We use the squared Euclidean norm to calculate losses for the position and generalized velocity terms:

$$\begin{aligned}
\mathcal{L}_{\mathbf{p}}(\boldsymbol{\theta}) &= \sum_{i=1}^{D}\sum_{n=1}^{N}\|\mathbf{p}_n^{(i)} - \bar{\mathbf{p}}_n^{(i)}\|_2^2, \\
\mathcal{L}_{\boldsymbol{\zeta}}(\boldsymbol{\theta}) &= \sum_{i=1}^{D}\sum_{n=1}^{N}\|\boldsymbol{\zeta}_n^{(i)} - \bar{\boldsymbol{\zeta}}_n^{(i)}\|_2^2.
\end{aligned} \tag{4.10}$$

The total loss $\mathcal{L}(\boldsymbol{\theta})$ is defined as:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\mathbf{R}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathbf{p}}(\boldsymbol{\theta}) + \mathcal{L}_{\boldsymbol{\zeta}}(\boldsymbol{\theta}). \tag{4.11}$$

4.3 Disturbance Model Learning Problem

When disturbances and system changes generate new out-of-distribution data, it is often too slow to re-train the dynamics model in Section 4.1 to support real-time adaptation to environment changes. In this section, we consider the problem of learning disturbance features from state-control trajectories that can be used to estimate the disturbance online in an adaptive control paradigm [38]. Let the system state $\mathbf{s}$ be defined as $\mathbf{s} = (\mathbf{q}, \mathfrak{p})$, where $\mathbf{q} \in \mathsf{Q}$, and its evolution is governed by the system dynamics:

$$\dot{\mathbf{s}} = \mathbf{f}(\mathbf{s}, \mathbf{u}, \mathbf{d}), \tag{4.12}$$

where $\mathbf{u}$ is the control input and $\mathbf{d}$ is a disturbance signal. The disturbance $\mathbf{d}$ is modeled as a linear combination of nonlinear features $\mathbf{W}(\mathbf{s}) \in \mathbb{R}^{6\times p}$:

$$\mathbf{d}(t) = \mathbf{W}(\mathbf{s}(t))\mathbf{a}^*, \tag{4.13}$$

where $\mathbf{a}^* \in \mathbb{R}^p$ are unknown feature weights.

A mechanical system obeys Hamilton's equations of motion [91], as shown in Eq. (2.31). The Hamiltonian, $\mathcal{H}(\mathbf{q}, \mathfrak{p}) = \mathcal{T}(\mathbf{q}, \mathfrak{p}) + \mathcal{V}(\mathbf{q})$, captures the total energy of the system as the sum of the kinetic energy $\mathcal{T}(\mathbf{q}, \mathfrak{p})$ and the potential energy $\mathcal{V}(\mathbf{q})$. The dynamics in (4.12) are determined by the Hamiltonian formulation [37, 47]:

$$\dot{\mathbf{q}} = \mathsf{T}_{\mathbf{e}}\mathsf{L}_{\mathbf{q}} \left(\frac{\partial \mathcal{H}(\mathbf{q}, \mathfrak{p})}{\partial \mathfrak{p}} \right), \tag{4.14a}$$

$$\dot{\mathfrak{p}} = \mathsf{ad}^*_{\boldsymbol{\xi}}(\mathfrak{p}) - \mathsf{T}^*_{\mathbf{e}}\mathsf{L}_{\mathbf{q}} \left(\frac{\partial \mathcal{H}(\mathbf{q}, \mathfrak{p})}{\partial \mathbf{q}} \right) + \mathbf{B}(\mathbf{q})\mathbf{u} + \mathbf{d}, \tag{4.14b}$$

where the disturbance $\mathbf{d}$ appears as an external force applied to the system.

Consider a collection $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_M\}$ of system state transitions $\mathcal{D}_j$, each obtained under a different unknown disturbance realization $\mathbf{a}^*_j$, for $j = 1, \ldots, M$. Each $\mathcal{D}_j = \{\mathbf{s}_0^{(ij)}, \mathbf{u}^{(ij)}, \mathbf{s}_f^{(ij)}, \tau^{(ij)}\}_{i=1}^{D_j}$ consists of D_j state transitions, each obtained by applying a constant control input $\mathbf{u}^{(ij)}$ to the system with initial condition $\mathbf{s}_0^{(ij)}$ and sampling the state

$\mathbf{s}_f^{(ij)} := \mathbf{s}^{(ij)}(\tau^{(ij)})$ at time $\tau^{(ij)}$. Our objective is to approximate the disturbance model in (4.13) by $\bar{\mathbf{d}}_{\boldsymbol{\theta}}(t) = \mathbf{W}_{\boldsymbol{\theta}}(\mathbf{s}(t))\mathbf{a}_j$, where $\boldsymbol{\theta}$ parameterizes the shared disturbance features and the parameters $\{\mathbf{a}_j\}_{j=1}^M$ model each disturbance realization. To optimize $\boldsymbol{\theta}$, $\{\mathbf{a}_j\}$, we predict the dynamics evolution starting from state $\mathbf{s}_0^{(ij)}$ with control $\mathbf{u}^{(ij)}$ and minimize the distance between the predicted state $\bar{\mathbf{s}}_f^{(ij)}$ and the true state $\mathbf{s}_f^{(ij)}$ from $\mathcal{D}_j$, for $j = 1, \ldots, M$. Since the approximated disturbance $\bar{\mathbf{d}}_{\boldsymbol{\theta}}$ does not change if the features $\mathbf{W}_{\boldsymbol{\theta}}$ and the coefficients $\mathbf{a}_j$ are scaled by constants γ and $1/\gamma$, respectively, we add the norms of the neural network outputs, $\mathbf{W}_{\boldsymbol{\theta}}(\mathbf{s}_0^{(ij)})$ and $\{\mathbf{a}_j\}_{j=1}^M$, to the objective function as regularization terms.

Problem 6. Given $\mathcal{D} = \left\{\{\mathbf{s}_0^{(ij)}, \mathbf{u}^{(ij)}, \mathbf{s}_f^{(ij)}, \tau^{(ij)}\}_{i=1}^{D_j}\right\}_{j=1}^{M}$, find disturbance feature parameters $\boldsymbol{\theta}$, and coefficients $\{\mathbf{a}_j\}_{j=1}^M$ that minimize:

$$
\begin{aligned}
\min_{\boldsymbol{\theta}, \{\mathbf{a}_j\}} \; & \sum_{j=1}^{M}\sum_{i=1}^{D_j} \ell(\mathbf{s}_f^{(ij)}, \bar{\mathbf{s}}_f^{(ij)}) + \lambda_{\boldsymbol{\theta}} \sum_{j=1}^{M}\sum_{i=1}^{D_j} \|\mathbf{W}_{\boldsymbol{\theta}}(\mathbf{s}_0^{(ij)})\|^2 + \lambda_{\mathbf{a}} \sum_{j=1}^{M} \|\mathbf{a}_j\|^2 \\
\text{s.t. } & \dot{\bar{\mathbf{s}}}^{(ij)}(t) = \mathbf{f}(\bar{\mathbf{s}}^{(ij)}(t), \mathbf{u}^{(ij)}, \bar{\mathbf{d}}_{\boldsymbol{\theta}}^{(ij)}(t)), \\
& \bar{\mathbf{d}}_{\boldsymbol{\theta}}^{(ij)}(t) = \mathbf{W}_{\boldsymbol{\theta}}(\bar{\mathbf{s}}^{(ij)}(t))\mathbf{a}_j, \\
& \bar{\mathbf{s}}^{(ij)}(0) = \mathbf{s}_0^{(ij)}, \;\; \bar{\mathbf{s}}_f^{(ij)} = \bar{\mathbf{s}}^{(ij)}(\tau^{(ij)}), \\
& \forall i = 1, \ldots, D_j, \;\; \forall j = 1, \ldots, M,
\end{aligned}
\tag{4.15}
$$

where ℓ is a distance metric on the state space.

After the offline disturbance feature identification in Problem 6, we design an adaptive controller $\mathbf{u} = \boldsymbol{\pi}(\mathbf{s}, \mathbf{s}^*, \mathbf{a}; \boldsymbol{\theta})$ in Chapter 5 that tracks a desired state trajectory $\mathbf{s}^*(t)$, compensating for the disturbance, estimated online using the learned disturbance model $\mathbf{W}_{\boldsymbol{\theta}}(\mathbf{s})$ via an adaptation law $\dot{\mathbf{a}} = \boldsymbol{\rho}(\mathbf{s}, \mathbf{s}^*, \mathbf{a}; \boldsymbol{\theta})$.

4.4 Hamiltonian-based Disturbance Feature Learning

To address Problem 6, we use a neural ODE network [21] whose structure respects Hamilton's equations in (2.27) with known generalized mass $\mathbf{M}(\mathbf{q})$, potential energy $\mathcal{V}(\mathbf{q})$ and the input gain $\mathbf{g}(\mathbf{q})$. As the energy-dissipating components $\mathbf{D}(\mathbf{q}, \mathbf{p})$ can be considered as a part of the disturbance sources, we omit the energy dissipation $\mathbf{D}(\mathbf{q}, \mathbf{p})$ for simplicity. We introduce

a disturbance model, $\mathbf{d} = \mathbf{W}_{\boldsymbol{\theta}}(\mathbf{q}, \mathfrak{p})\mathbf{a}$, where $\mathbf{W}_{\boldsymbol{\theta}}(\mathbf{q}, \mathfrak{p})$ is a neural network, and estimate its parameters $\boldsymbol{\theta}$ from disturbance-corrupted data. The training data $\mathcal{D}_j = \{\mathbf{s}_0^{(ij)}, \mathbf{u}^{(ij)}, \mathbf{s}_f^{(ij)}, \tau^{(ij)}\}_{i=1}^{D_j}$ may be obtained using an odometry algorithm [118] or a motion capture system. The data collection can be performed using an existing baseline controller or a human operator manually controlling the system under different disturbance conditions (e.g., wind, ground effect, etc. for an aerial robot).

We define the geometric distance metric ℓ in Problem 5 as a sum of position, orientation, and momentum errors:

$$\ell(\mathbf{s}, \bar{\mathbf{s}}) = \ell_{\mathbf{q}}(\mathbf{s}, \bar{\mathbf{s}}) + \ell_{\mathfrak{p}}(\mathbf{s}, \bar{\mathbf{s}}), \tag{4.16}$$

where the loss function $\ell_{\mathbf{q}}(\mathbf{s}, \bar{\mathbf{s}}) = \left\| \log_{\mathsf{G}}^{\vee} \left(\bar{\mathbf{q}}_n^{(i)} \left(\mathbf{q}_n^{(i)} \right)^{-1} \right) \right\|_2^2$, $\ell_{\mathfrak{p}}(\mathbf{s}, \bar{\mathbf{s}}) = \|\mathfrak{p} - \bar{\mathfrak{p}}\|_2^2$.

Let $\mathcal{L}(\boldsymbol{\theta}, \{\mathbf{a}_j\}; \mathcal{D})$ be the total loss in Problem 6. To calculate the loss, for each dataset $\mathcal{D}_j$ with disturbance $\bar{\mathbf{d}}_{\boldsymbol{\theta}}^{(ij)}(t) = \mathbf{W}_{\boldsymbol{\theta}}(\bar{\mathbf{s}}^{(ij)}(t))\mathbf{a}_j$, we solve an ODE:

$$\dot{\bar{\mathbf{s}}}^{(ij)} = \mathbf{f}(\bar{\mathbf{s}}^{(ij)}, \mathbf{u}^{(ij)}, \bar{\mathbf{d}}_{\boldsymbol{\theta}}^{(ij)}), \quad \bar{\mathbf{s}}^{(ij)}(0) = \mathbf{s}_0^{(ij)}, \tag{4.17}$$

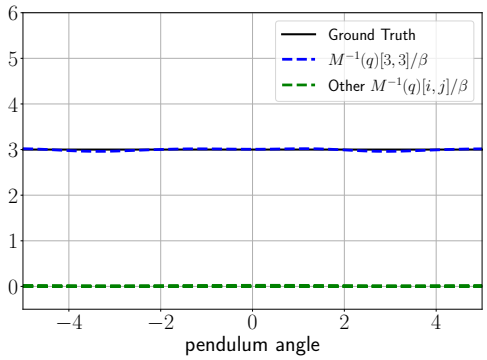
using an ODE solver. This generates a predicted state $\bar{\mathbf{s}}_f^{(ij)}$ at time $\tau^{(ij)}$ for each $i = 1, \ldots, D_j$ and $j = 1, \ldots, M$:

$$\bar{\mathbf{s}}_f^{(ij)} = \text{ODESolver}\left(\mathbf{s}_0^{(ij)}, \mathbf{f}, \tau^{(ij)}; \boldsymbol{\theta}\right), \tag{4.18}$$

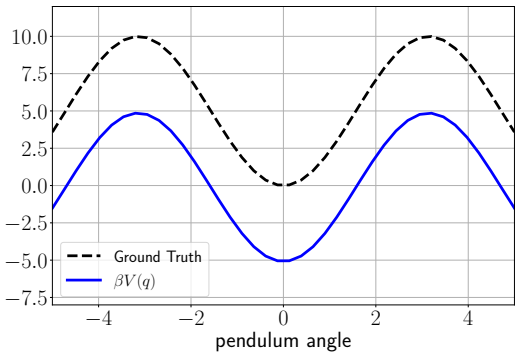
sufficient to compute $\mathcal{L}(\boldsymbol{\theta}, \{\mathbf{a}_j\}; \mathcal{D})$. The parameters $\boldsymbol{\theta}$ and $\{\mathbf{a}_j\}$ are updated using gradient descent by back-propagating the loss through the neural ODE solver using adjoint methods [21], as described in Section 2.3.

4.5 Evaluation

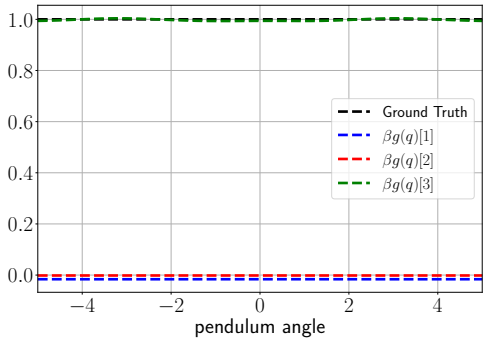
We verify the effectiveness of our port-Hamiltonian neural ODE network for dynamics learning and control on matrix Lie groups using a simulated pendulum and a simulated Crazyflie quadrotor platform, whose states evolve on the $SE(3)$ manifold. The implementation details for the experiments are provided in Appendix C.



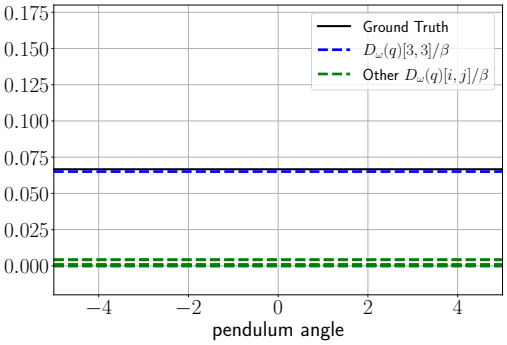
(a) $\mathbf{M}^{-1}(q)/\beta$ versus φ.



(b) $\beta V(q)$ versus φ.



(c) $\beta\mathbf{g}(q)$ versus φ.



(d) $\mathbf{D}_\omega(q)/\beta$ versus φ.

Figure 4.3. Evaluation of our $SO(3)$ Hamiltonian neural ODE network on a pendulum system with scale factor $\beta = 1.33$.

4.5.1 Pendulum

We consider a pendulum with the following dynamics:

$$\ddot{\varphi} = -15 \sin \varphi + 3u - 0.2\dot{\varphi}, \tag{4.19}$$

where φ is the angle of the pendulum with respect to its vertically downward position and u is a scalar control input. The ground-truth mass, potential energy, friction coefficient and the input gain are: $m = 1/3, V(\varphi) = 5(1 - \cos \varphi), D(\varphi) = 0.2/3$, and $g(\varphi) = 1$, respectively. We collected data of the form $\{(\cos \varphi, \sin \varphi, \dot{\varphi})\}$ from an OpenAI Gym environment, provided by [184], with the dynamics in (4.19). To illustrate our manifold-constrained neural ODE learning, we viewed

φ as a yaw angle and convert $(\cos\varphi, \sin\varphi)$ into a rotation matrix:

$$\mathbf{R} = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.20}$$

We used $\boldsymbol{\omega} = [0, 0, \dot{\varphi}]$ for angular velocity and remove position $\mathbf{p}$ and linear velocity $\mathbf{v}$ from the Hamiltonian dynamics in (2.40), restricting the system to the $SO(3)$ manifold with generalized coordinates $\mathbf{q} = [\mathbf{r}_1^\top \quad \mathbf{r}_2^\top \quad \mathbf{r}_3^\top]^\top \in \mathbb{R}^9$.

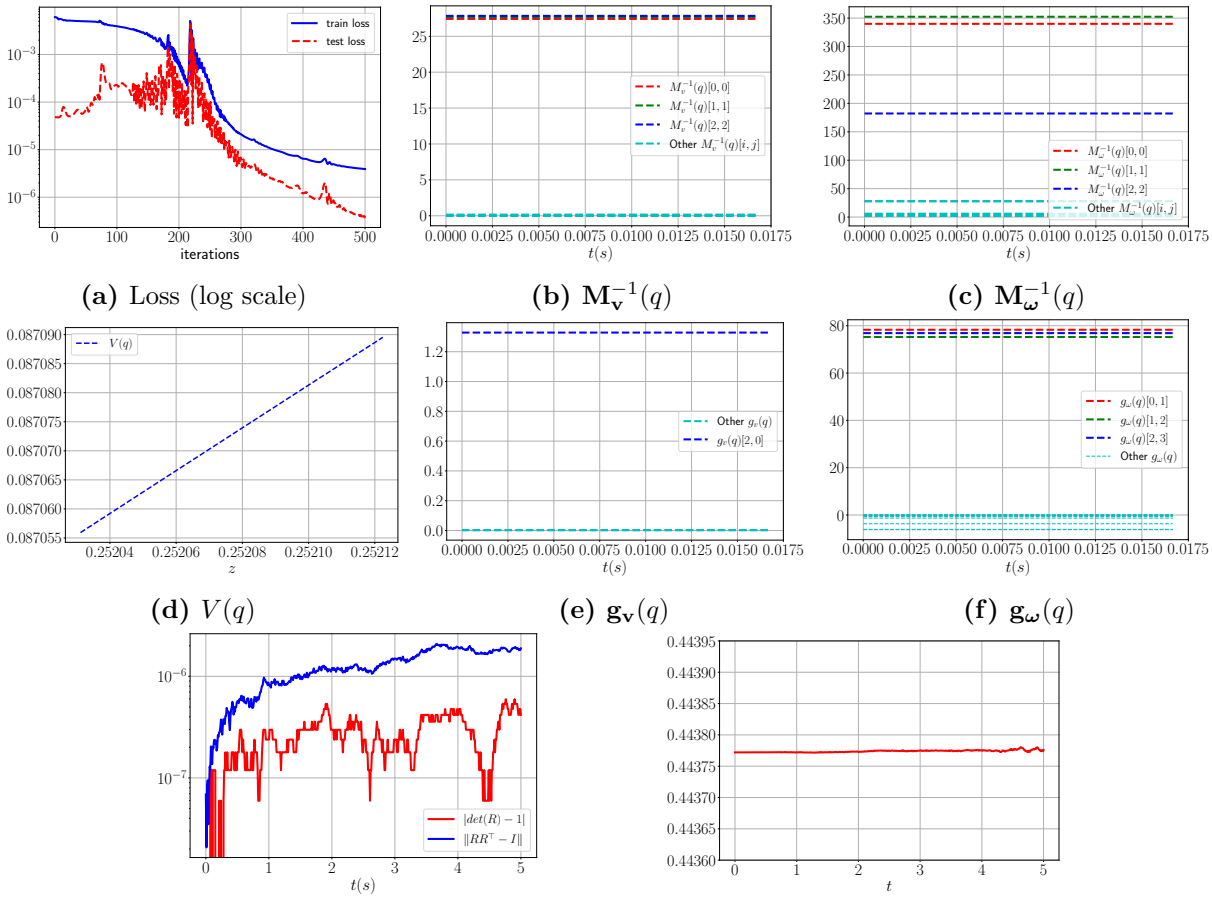
As described in Section 4.2.4, control inputs $\mathbf{u}^{(i)}$ were sampled randomly and applied to the pendulum for five time intervals of $0.05s$, forming a dataset $\mathcal{D} = \left\{ t_{0:N}^{(i)}, \mathbf{q}_{0:N}^{(i)}, \boldsymbol{\omega}_{0:N}^{(i)}, \mathbf{u}^{(i)} \right\}_{i=1}^{D}$ with $N = 5$ and $D = 5120$. We trained the $SO(3)$ Hamiltonian neural ODE network as described in Section 4.2.4 for 5000 iterations without any nominal model, i.e., $\mathbf{M}_{\boldsymbol{\omega}0}^{-1}(\mathbf{q}) = \mathbf{0}$, $\mathbf{D}_{\boldsymbol{\omega}0}(\mathbf{q}, \mathbf{p}) = \mathbf{0}$, $\mathcal{V}_0(\mathbf{q}) = \mathbf{0}$ and $\mathbf{g}_0(\mathbf{q}) = \mathbf{0}$.

As noted in [184], since the generalized momenta $\mathbf{p}$ are not available in the dataset, the dynamics of $\mathbf{q}$ in (4.19) do not change if $\mathbf{p}$ is scaled by a factor $\beta > 0$. This is also true in our formulation as scaling $\mathbf{p}$ leaves the dynamics of $\mathbf{q}$ in (2.40) unchanged. To emphasize this scale invariance, let $\mathbf{M}_\beta(\mathbf{q}) = \beta\mathbf{M}(\mathbf{q})$, $\mathcal{V}_\beta(\mathbf{q}) = \beta\mathcal{V}(\mathbf{q})$, $\mathbf{D}_\beta(\mathbf{q}, \mathbf{p}) = \beta\mathbf{D}(\mathbf{q}, \mathbf{p})$ $\mathbf{g}_\beta(\mathbf{q}) = \beta\mathbf{g}(\mathbf{q})$, and:

$$\begin{aligned} \mathbf{p}_\beta &= \mathbf{M}_\beta(\mathbf{q})\boldsymbol{\omega} = \beta\mathbf{p}, \\ \dot{\mathbf{p}}_\beta &= \beta\dot{\mathbf{p}}, \\ \mathcal{H}_\beta(\mathbf{q}, \mathbf{p}) &= \frac{1}{2}\mathbf{p}_\beta^\top \mathbf{M}_\beta^{-1}(\mathbf{q})\mathbf{p}_\beta + \mathcal{V}_\beta(\mathbf{q}) = \beta\mathcal{H}(\mathbf{q}, \mathbf{p}), \\ \frac{\partial \mathcal{H}_\beta(\mathbf{q}, \mathbf{p})}{\partial \mathbf{p}_\beta} &= \mathbf{M}_\beta^{-1}(\mathbf{q})\mathbf{p}_\beta = \frac{\partial \mathcal{H}(\mathbf{q}, \mathbf{p})}{\partial \mathbf{p}}, \end{aligned} \tag{4.21}$$

guaranteeing that the equations of motions (2.40) still hold.

Figure 4.3 shows the training and testing behavior of our $SO(3)$ Hamiltonian ODE network. Figure 4.3a and 4.3c show that the $\left[\mathbf{M}(\mathbf{q})^{-1}\right]_{3,3}$ entry of the mass inverse and the $[\mathbf{g}(\mathbf{q})]_3$ entry of the input matrix are close to their correct values of 3 and 1, respectively, while the other entries are close to zero. Figure 4.3b indicates a constant gap between the learned and the ground-truth potential energy, which can be explained by the relativity of potential energy.



(a) Loss (log scale) **(b)** $\mathbf{M}_{\mathbf{v}}^{-1}(q)$ **(c)** $\mathbf{M}_{\boldsymbol{\omega}}^{-1}(q)$

(d) $V(q)$ **(e)** $\mathbf{g}_{\mathbf{v}}(q)$ **(f)** $\mathbf{g}_{\boldsymbol{\omega}}(q)$

(g) SO(3) constraints. **(h)** Total energy.

Figure 4.4. Evaluation of the $SE(3)$ Hamiltonian neural ODE network on an under-actuated Crazyflie quadrotor in the PyBullet simulator [131].

4.5.2 Crazyflie Quadrotor

In this section, we demonstrate that our $SE(3)$ dynamics learning and control approach can achieve trajectory tracking for an under-actuated system. We consider a Crazyflie quadrotor, shown in Figure 5.6a, simulated in the physics-based simulator PyBullet [131]. The control input $\mathbf{u} = [f, \boldsymbol{\tau}]$ includes a thrust $f \in \mathbb{R}_{\geq 0}$ and a torque vector $\boldsymbol{\tau} \in \mathbb{R}^3$ generated by the 4 rotors. The generalized coordinates and velocity are $\mathbf{q} = [\mathbf{p}^\top \quad \mathbf{r}_1^\top \quad \mathbf{r}_2^\top \quad \mathbf{r}_3^\top]^\top \in SE(3)$ and $\boldsymbol{\zeta} = [\mathbf{v}^\top \quad \boldsymbol{\omega}^\top]^\top \in \mathbb{R}^6$ as before. As we do not consider energy dissipation such as drag effects in the Pybullet simulator, we omit the dissipation matrix $\mathbf{D}(\mathbf{q}, \mathbf{p})$ in the model design in Section 4.2.4.

The quadrotor was controlled from a random starting point to 18 different desired poses using a PID controller provided by [131], providing 18 2.5-second trajectories. The trajectories

were used to generate a dataset $\mathcal{D} = \{t_{0:N}^{(i)}, \mathbf{q}_{0:N}^{(i)}, \boldsymbol{\zeta}_{0:N}^{(i)}, \mathbf{u}^{(i)})\}_{i=1}^{D}$ with $N = 5$ and $D = 1080$. The $SE(3)$ Hamiltonian ODE network was trained, as described in Section 4.2.4, for 500 iterations without any nominal model, i.e., $\mathbf{M}_{\mathbf{v}0}^{-1}(\mathbf{q}) = \mathbf{0}$, $\mathbf{M}_{\boldsymbol{\omega}0}^{-1}(\mathbf{q}) = \mathbf{0}$, $\mathbf{D}_{\mathbf{v}0}(\mathbf{q}, \mathbf{p}) = \mathbf{0}$, $\mathbf{D}_{\boldsymbol{\omega}0}(\mathbf{q}, \mathbf{p}) = \mathbf{0}$, $V_0(\mathbf{q}) = \mathbf{0}$ and $\mathbf{g}_0(\mathbf{q}) = \mathbf{0}$. As the data from the Pybullet simulator was not affected by drag effect, we skip the dissipation matrix in the dynamics model. Meanwhile, the trained model for the real experiments in Chapter 5.4.5 includes energy dissipation to account for the drag forces on the real systems.

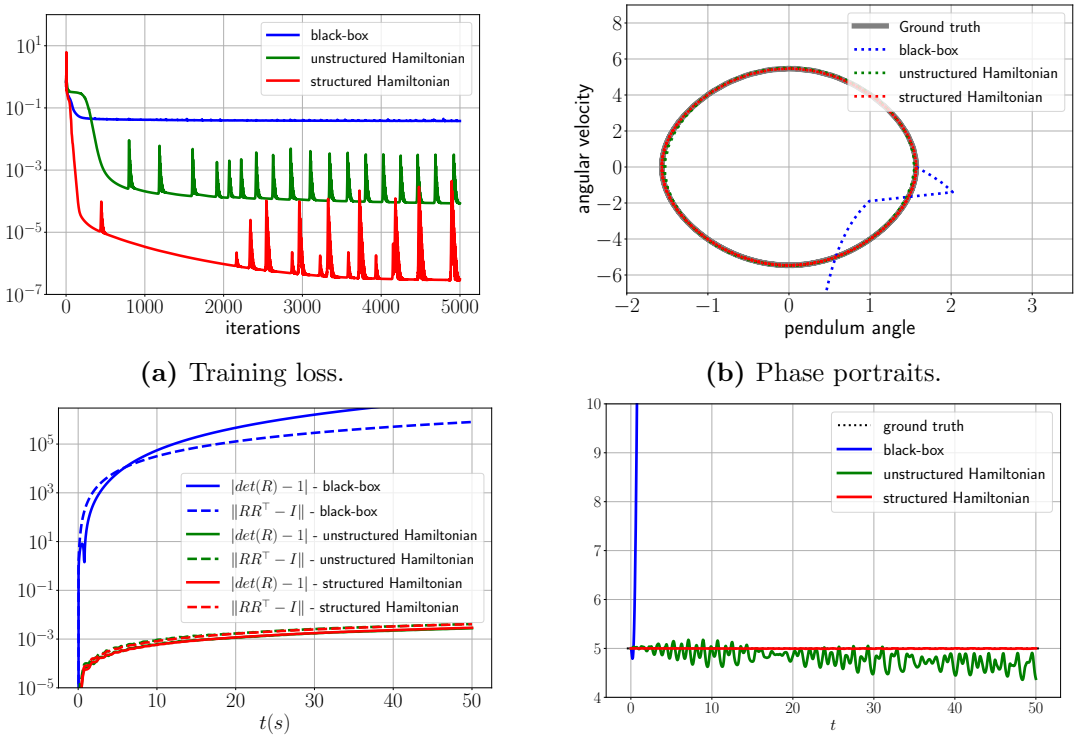
Our training and test results are shown in Figure 4.4. The learned generalized mass and inertia converged to constant diagonal matrices:

$$\mathbf{M}_1^{-1}(\mathbf{q}) \approx 27.5\mathbf{I}, \quad \mathbf{M}_2^{-1}(\mathbf{q}) \approx \text{diag}([351, 340, 181]).$$

The input matrix $\mathbf{g}_{\mathbf{v}}(\mathbf{q})$ converged to a constant matrix whose entry $\Big[\mathbf{g}_{\mathbf{v}}(\mathbf{q})\Big]_{2,0} \approx 1.33$ while other entries were closed to 0, consistent with the fact that the thrust only affects the linear velocity along the z axis in the body-fixed frame. The input matrix $\mathbf{g}_{\boldsymbol{\omega}}(\mathbf{q})$ converged to $\sim 76\mathbf{I}$ as the torques affects all components of the angular velocity $\boldsymbol{\omega}$. The potential energy $V(\mathbf{q})$ was linear in the height z, agreed with the gravitational potential.

4.5.3 Comparison to Unstructured Neural ODE Models

In this section, we show the benefits of our neural ODE architecture by comparing 1) our *structured Hamiltonian model*, 2) a *black-box model*, i.e., the approximated dynamics $\bar{\mathbf{f}}_{\boldsymbol{\theta}}$ is represented by a multilayer perceptron network, and 3) an *unstructured Hamiltonian model*, i.e., the Hamiltonian function is represented by a multilayer perceptron network instead of using the structure in Eq. (2.37), in terms of training convergence rates, guarantees of energy conservation principle, and Lie group constraint satisfaction. To verify energy conservation, we rolled out the learned dynamics and calculated the Hamiltonian via (2.37) along the predicted trajectories for (1) the black-box model using ground-truth mass and potential energy with the predicted states; (2) the unstructured Hamiltonian model using the output of the multilayer perceptron Hamiltonian network; and (3) the structured Hamiltonian model using the learned mass and potential energy networks. We check the $SO(3)$ constraints by verifying that two quantities



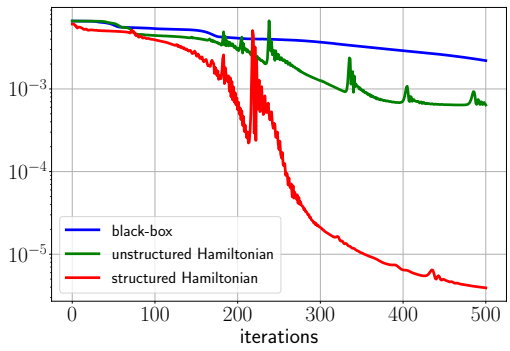
(a) Training loss. **(b)** Phase portraits.

(c) SO(3) constraints. **(d)** Total energy.

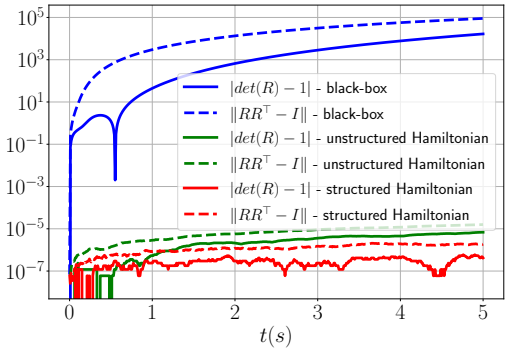
Figure 4.5. Comparison of different network architectures to learn pendulum dynamics: 1) black-box; 2) unstructured Hamiltonian; 3) structured Hamiltonian.

$|\det \mathbf{R} - 1|$ and $\|\mathbf{R}\mathbf{R}^\top - \mathbf{I}\|$ remain small along the predicted trajectories.

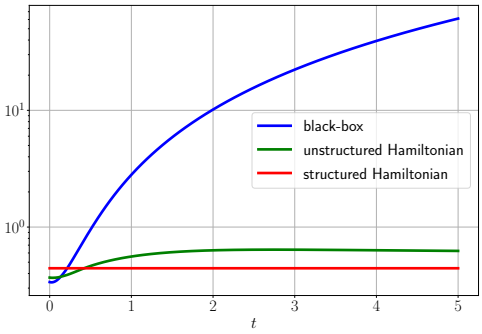
We first use a pendulum as described in Section 4.5.1 without energy dissipation. The models are trained for 5000 iterations from 512 0.2-second state-control trajectories and rolled out for a significantly longer horizon of 50 seconds. Figure 4.5 plots the training loss, the phase portraits, the SO(3) constraints and the total energy (Hamiltonian) of the learned models for a pendulum system. As the Hamiltonian structure is imposed in the neural ODE network architecture, our model is able to converge faster with lower loss (Figure 4.5a) and preserves the phase portraits for state predictions (Figure 4.5b). Figure 4.5c shows that the $SO(3)$ constraints are satisfied by our structured and unstructured Hamiltonian models as their values of $|\det \mathbf{R} - 1|$ and $\|\mathbf{R}\mathbf{R}^\top - \mathbf{I}\|$ remain small along a 50-second trajectory rollout initialized at $\phi = \pi/2$. The constant Hamiltonian in Figure 4.5d of our structured Hamiltonian verifies that our model obeys the energy conservation law with no control input and no energy dissipation. The Hamiltonian of the black-box model increases along the trajectory while that of the unstructured Hamiltonian



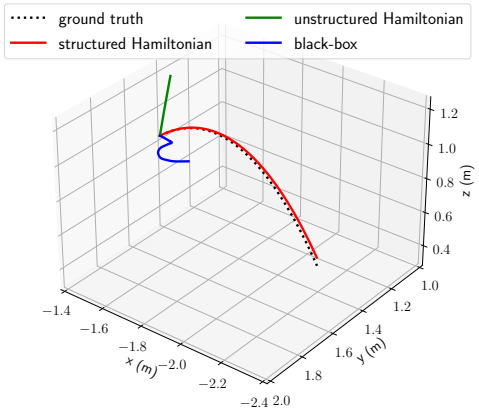
(a) Training loss.



(b) SO(3) constraints.



(c) Total energy.



(d) Predicted trajectories.

Figure 4.6. Comparison of different network architecture to learn quadrotor dynamics: 1) black-box; 2) unstructured Hamiltonian; 3) structured Hamiltonian.

model fluctuates and slightly decreases over time.

We also tested the models using the simulated Crazyflie quadrotor with the same dataset $\mathcal{D}$ of 18 trajectories as described in Section 4.5.2. The $SE(3)$ port-Hamiltonian ODE network was trained, as described in Section 4.2.4, for 500 iterations. Our structured Hamiltonian model converges faster with significantly lower loss as seen in Figure 4.6a. We verified that the predicted orientation trajectories from our learned models satisfy the $SO(3)$ constraints. Figure 4.6b shows two near-zero quantities $|\det \mathbf{R} - 1|$ and $\|\mathbf{R}\mathbf{R}^\top - \mathbf{I}\|$, obtained by rolling out our learned dynamics for 5 seconds, while the learned black-box model violates the constraints after a very short time. Figure 4.6c shows a constant total energy along the predicted trajectory from our structured Hamiltonian model without control input and dissipation networks, verifying that the learned model obeys the law of energy conservation. Figure 4.6d shows that our structured Hamiltonian

model provides better trajectory predictions compared to the other methods.

4.6 Summary

This chapter proposes a neural ODE network design for robot dynamics learning that captures Lie group kinematics, e.g. the $SE(3)$ manifold, and Hamiltonian dynamics constraints by construction. The learning design is not system-specific and thus can be applied to different types of robots, such as mobile robots whose state evolves on the $SE(3)$ manifold. This technique has the potential to enable robots to quickly adapt their models online, in response to changing operational conditions or structural damage, and, yet, maintain stability and autonomous operation. Future work will focus on extending our formulation to allow learning the kinematic and dynamic structure of multi-rigid body systems, contact dynamics and provide safe and stable adaptive control in the presence of noise and disturbances.

Acknowledgements

Chapter 4, in part, is a reprint of the material as it appears in T. Duong, N. Atanasov, "*Hamiltonian-based Neural ODE Networks on the SE(3) Manifold For Dynamics Learning and Control*", Robotics: Science and Systems (RSS), 2021, in T. Duong, N. Atanasov, "*Physics-guided Learning-based Adaptive Control on the SE(3) Manifold*", Physical Reasoning and Inductive Biases for the Real World Workshop, 2021, and in T. Duong, N. Atanasov, "*Adaptive Control of SE(3) Hamiltonian Dynamics with Learned Disturbance Features*", IEEE Control Systems Letters (L-CSS), vol. 6, pp. 2773-2778, 2022. Chapter 4, in part, has been submitted for publication of the material as it may appear in T. Duong, A. Altawaitan, J. Stanley, N. Atanasov, "*Port-Hamiltonian-based Neural ODE Networks on Lie Groups For Robot Dynamics Learning and Control*", Transactions on Robotics, 2024. The dissertation author was the primary investigator and author of these papers.

Chapter 5

Autonomous Navigation with Learned Robot Dynamics and Sparse Map Representations

Finally, this chapter presents a complete online mapping, planning, and control for autonomous navigation tasks, using our sparse occupancy maps described in Chapter 3 and our learned dynamics model in Chapter 4. The robot starts by observing the environment via depth measurements from its sensors such as lidars. It runs our sequential mapping algorithm (Algorithm 1 or Algorithm 2 in Chapter 3) on the depth measurements to build a sparse (binary or probabilistic, respectively) occupancy map. The sparse kernel-based structure of the occupancy maps allows us to derive efficient collision checking algorithms for robot trajectories such as line segments or general curves in Section 5.1. The collision checking algorithms can be easily integrated into motion planner such as A^* [142] or RRT^* [83], which returns a desired trajectory for the robot to follow. In Section 5.2, we develop control policies to track the trajectory, with and without the presence of disturbance and dynamics changes, for the learned Hamiltonian dynamics in Chapter 4. The autonomous navigation algorithm is summarized in Section 5.3.

5.1 Motion Planning With Sparse Occupancy Maps

In this section, motivated by the collision checking approach in [11], we derive our own efficient collision checking algorithms for our map representations in Chapter 3. We develop an "inflated boundary" of the obstacle boundary that enables closed-form conditions for checking line segments and ellipsoids for collision. These key conditions allow us to check potential robot

trajectories for motion planning purposes, e.g., using common motion planners such as A^* [142] or RRT^* [83].

For example, if we use a first-order fully actuated robot, $\dot{\mathbf{p}} = \mathbf{v}$, where the state $\mathbf{s}$ is the robot position $\mathbf{p} \in [0, 1]^3$, with piecewise-constant velocity $\mathbf{v}(t) \equiv \mathbf{v}_k \in \mathcal{V}$ for $t \in [t_k,\ t_{k+1})$, the robot trajectories are piecewise-linear:

$$\mathbf{p}(t) = \mathbf{p}_k + (t - t_k)\mathbf{v}_k, \qquad t \in [t_k,\ t_{k+1}), \tag{5.1}$$

where $\mathbf{p}_k := \mathbf{p}(t_k)$. In this case, the classification algorithm for line segments (Algorithm 3) in Section 5.1.1 is used during motion planning.

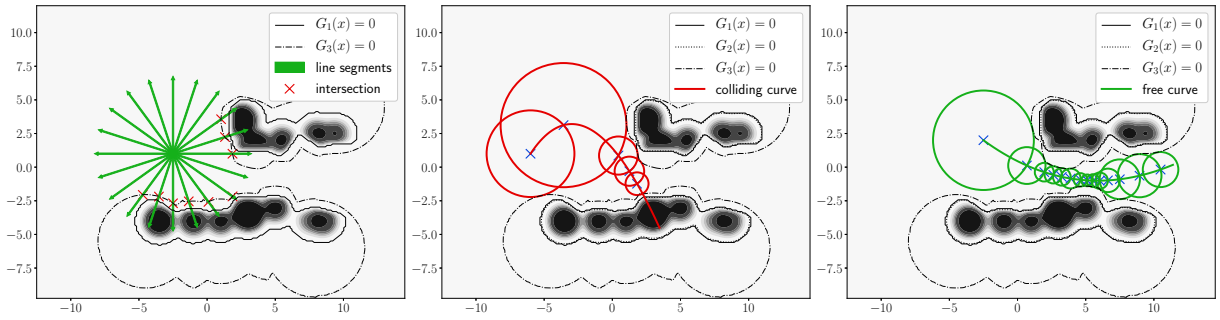
In the real experiments with ground robots in Section 5.4.4, we consider a ground wheeled Ackermann-drive robot with dynamics model:

$$\dot{\mathbf{p}} = v \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}, \qquad \dot{\theta} = \frac{v}{\ell} \tan \phi, \tag{5.2}$$

where the state $\mathbf{s}$ consists of the position $\mathbf{p} \in \mathbb{R}^2$ and orientation $\theta \in \mathbb{R}$, the control input $\mathbf{u}$ consists of the linear velocity $v \in \mathbb{R}$ and the steering angle $\phi \in \mathbb{R}$, and ℓ is the distance between the front and back wheels. The nonlinear car dynamics can be transformed into a 2nd-order fully actuated system $\ddot{\mathbf{p}} = \mathbf{a}$ via feedback linearization [32, 48]. Using piecewise-constant acceleration $\mathbf{a}(t) \equiv \mathbf{a}_k \in \mathcal{A}$ for $t \in [t_k,\ t_{k+1})$ leads to piecewise-polynomial trajectories:

$$\mathbf{p}(t) = \mathbf{p}_k + (t - t_k)v_k \begin{bmatrix} \cos(\theta_k) \\ \sin(\theta_k) \end{bmatrix} + \frac{(t - t_k)^2}{2}\mathbf{a}_k, \tag{5.3}$$

where $\mathbf{p}_k := \mathbf{p}(t_k)$, $\theta_k := \theta(t_k)$, $v_k := v(t_k)$. For the simulated and real quadrotor platforms in Section 5.4.2, 5.4.3, and 5.4.5, the robot trajectories can be modeled as piecewise-polynomials due to their differential flatness [99]. For these robot platforms, the classification algorithm for curves (Algorithm 4) can used to get successor nodes in an A^* or RRT^* motion planning algorithm.



(a) Checking line segments. **(b)** Checking a colliding curve. **(c)** Checking a free curve.

Figure 5.1. Illustration of our classification algorithms for the trained RVM model in Figure 3.2 with $b = -0.05, e = -0.01$, and $n_1 = n_2 = 1$.

5.1.1 Checking Line Segments

Consider a linear trajectory described by a ray $\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v}$, $t \geq 0$ such that $\mathbf{p}_0$ is obstacle-free according to Prop. 2, i.e., $G_3(\mathbf{p}_0) \leq 0$, and $\mathbf{v}$ is a constant. To check if $\mathbf{p}(t)$ collides with the inflated boundary $G_3(\mathbf{x}) = 0$, we find a time t_u such that any point $\mathbf{p}(t)$ is classified free for $t \in [0, t_u)$.

Proposition 3. *Consider a ray $\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v}$, $t \geq 0$. Let $\mathbf{x}_i^+$ and $\mathbf{x}_j^-$ be arbitrary positive and negative relevance vectors. Then, any point $\mathbf{p}(t)$ with $t \in [0, t_u) \subseteq [0, t_u^*)$ is free for:*

$$t_u := \min_{i=1,\ldots,M^+} \tau(\mathbf{p}_0, \mathbf{x}_i^+, \mathbf{x}_j^-) \tag{5.4}$$

$$t_u^* := \min_{i=1,\ldots,M^+} \max_{j=1,\ldots,M^-} \tau(\mathbf{p}_0, \mathbf{x}_i^+, \mathbf{x}_j^-), \tag{5.5}$$

where $\tau(\mathbf{p}_0, \mathbf{x}_i^+, \mathbf{x}_j^-)$

$$= \begin{cases} +\infty, & \text{if } V(t, \mathbf{x}_i^+, \mathbf{x}_j^-) \text{ has less than 2 roots} \\ +\infty, & \text{if } V(t, \mathbf{x}_i^+, \mathbf{x}_j^-) \text{ has 2 roots } t_1 < t_2 \leq 0 \\ t_1 & \text{if } V(t, \mathbf{x}_i^+, \mathbf{x}_j^-) \text{ has 2 roots } 0 \leq t_1 < t_2 \\ 0 & \text{if } V(t, \mathbf{x}_i^+, \mathbf{x}_j^-) \text{ has 2 roots } t_1 \leq 0 \leq t_2 \end{cases}.$$

Algorithm 3. Collision Checking for Line Segments

Input: Line segment $(\mathbf{p}_A, \mathbf{p}_B)$; relevance vectors $\Lambda = \{(\mathbf{x}_i, y_i, \boldsymbol{\xi}_i)\}$; weight posterior mean $\boldsymbol{\mu}$ and max covariance eigenvalue λ_{max}

1: $\mathbf{v}_A = \mathbf{p}_B - \mathbf{p}_A$, $\mathbf{v}_B = \mathbf{p}_A - \mathbf{p}_B$
2: Calculate t_{uA} and t_{uB} using Eq. (5.4) or Eq. (5.5).
3: **if** $t_{uA} + t_{uB} > 1$ **then return** True (Free)
4: **else return** False (Colliding)

and $V(t, \mathbf{x}_i^+, \mathbf{x}_j^-) = at^2 + b(\mathbf{x}_i^+, \mathbf{x}_j^-)t + c(\mathbf{x}_i^+, \mathbf{x}_j^-)$ *with*

$$\begin{aligned} a &:= -n_1\|\boldsymbol{\Gamma}\mathbf{v}\|^2, \\ b(\mathbf{x}_i^+, \mathbf{x}_j^-) &:= -2\mathbf{v}^\top\boldsymbol{\Gamma}^\top\boldsymbol{\Gamma}(n_1\mathbf{p}_0 - (n_1+n_2)\mathbf{x}_i^+ + n_2\mathbf{x}_j^-), \\ c(\mathbf{x}_i^+, \mathbf{x}_j^-) &:= -(n_1+n_2)\|\boldsymbol{\Gamma}(\mathbf{p}_0 - \mathbf{x}_i^+)\|^2 + n_2\|\boldsymbol{\Gamma}(\mathbf{p}_0 - \mathbf{x}_j^-)\|^2 \\ &\quad -(n_1+n_2)\log\frac{\rho(e-b, \nu_j^-)}{\eta^{\frac{n_1}{n_1+n_2}}\sum_{i=1}^{M^+}\nu_i^+}. \end{aligned}$$

Proof. Please refer to Appendix B.3. □

For a line segment $(\mathbf{p}_A, \mathbf{p}_B)$, all points on the segment can be expressed as $\mathbf{p}(t_A) = \mathbf{p}_A + t_A\mathbf{v}_A$, $\mathbf{v}_A = \mathbf{p}_B - \mathbf{p}_A$, $0 \le t_A \le 1$ or $\mathbf{p}(t_B) = \mathbf{p}_B + t_B\mathbf{v}_B$, $\mathbf{v}_B = \mathbf{p}_A - \mathbf{p}_B$, $0 \le t_B \le 1$. Using the upper bound t_{uA} on t_A provided by Eq. (5.4) or Eq. (5.5), we find the free region on $(\mathbf{p}_A, \mathbf{p}_B)$ starting from $\mathbf{p}_A$. Likewise, we calculate t_{uB} which specifies the free region from $\mathbf{p}_B$. If $t_{uA} + t_{uB} > 1$, the entire line segment is free, otherwise the segment is considered colliding. The proposed approach is summarized in Algorithm 3 and illustrated in Figure 5.1a.

5.1.2 Checking Curves

Instead of a constant velocity $\mathbf{v}$ representing the direction of motion, we can define a general curve $\mathbf{p}(t)$ by considering a time-varying term $\mathbf{v}(t)$. We extend the collision checking conditions in Prop. 3 by finding an ellipsoid $\mathcal{E}(\mathbf{p}_0, r) := \{\mathbf{x} : \|\boldsymbol{\Gamma}(\mathbf{x} - \mathbf{p}_0)\| \le r\}$ around $\mathbf{p}_0$ whose interior is free of obstacles, where Γ is the kernel parameter defined in Assumption 1. This specific form of the ellipsoid leads a closed-conditions as shown in the Prop. 4.

Proposition 4. *Let* $\mathbf{p}_0$ *be such that* $G_3(\mathbf{p}_0) < 0$ *and let* $\mathbf{x}_i^+$ *and* $\mathbf{x}_j^-$ *be arbitrary positive and*

negative support vectors. The interior of the ellipsoids $\mathcal{E}(\mathbf{p}_0, r_u) \subseteq \mathcal{E}(\mathbf{p}_0, r_u^*)$ *is free for:*

$$r_u = \min_{i=1,\ldots,M^+} r(\mathbf{p}_0, \mathbf{x}_i^+, \mathbf{x}_j^-) \tag{5.6}$$

$$r_u^* = \min_{i=1,\ldots,M^+} \max_{j=1,\ldots,M^-} r(\mathbf{p}_0, \mathbf{x}_i^+, \mathbf{x}_j^-). \tag{5.7}$$

where $r(\mathbf{p}_0, \mathbf{x}_i^+, \mathbf{x}_j^-)$

$$= \begin{cases} +\infty, & \text{if } \bar{V}(t, \mathbf{x}_i^+, \mathbf{x}_j^-) \text{ has less than 2 roots} \\ +\infty, & \text{if } \bar{V}(t, \mathbf{x}_i^+, \mathbf{x}_j^-) \text{ has 2 roots } t_1 < t_2 \leq 0 \\ t_1 & \text{if } \bar{V}(t, \mathbf{x}_i^+, \mathbf{x}_j^-) \text{ has 2 roots } 0 \leq t_1 < t_2 \\ 0 & \text{if } \bar{V}(t, \mathbf{x}_i^+, \mathbf{x}_j^-) \text{ has 2 roots } t_1 \leq 0 \leq t_2 \end{cases},$$

and $\bar{V}(t, \mathbf{x}_i^+, \mathbf{x}_j^-) = \bar{a}t^2 + \bar{b}(\mathbf{x}_i^+, \mathbf{x}_j^-)t + \bar{c}(\mathbf{x}_i^+, \mathbf{x}_j^-)$ *with*

$$\begin{aligned} \bar{a} &:= -n_1, \\ \bar{b}(\mathbf{x}_i^+, \mathbf{x}_j^-) &:= 2\|\mathbf{\Gamma}(n_1\mathbf{p}_0 - (n_1+n_2)\mathbf{x}_i^+ + n_2\mathbf{x}_j^-)\|, \\ \bar{c}(\mathbf{x}_i^+, \mathbf{x}_j^-) &:= c(\mathbf{x}_i^+, \mathbf{x}_j^-). \end{aligned}$$

Proof. Please refer to Appendix B.4. □

Consider a general time-parameterized curve $\mathbf{p}(t)$, $t \in [0, t_f]$ from $\mathbf{p}_0 := \mathbf{p}(0)$ to $\mathbf{p}_f := \mathbf{p}(t_f)$. Prop. 4 shows that all points inside the ellipsoid $\mathcal{E}(\mathbf{p}_0, r)$ are free for $r = r_u \leq r_u^*$. If we can find the smallest positive t_1 such that

$$\|\mathbf{\Gamma}(\mathbf{p}(t_1) - \mathbf{p}_0)\| = r, \tag{5.8}$$

then all points on the curve $\mathbf{p}(t)$ for $t \in [0, t_1)$ are free. This is equivalent to finding the smallest positive solution of Eq. (5.8). We perform curve classification by iteratively covering the curve by free ellipsoids. If the value of r is smaller than a threshold ε, the curve is considered colliding. Otherwise, it is considered free. The classification process for curves is shown in Algorithm 4 and illustrated in Figure 5.1b and 5.1c for the trained RVM model in Figure 3.2 for a colliding

Algorithm 4. Collision Checking for Curves

Input: Curve $\mathbf{p}(t)$, $t \in [0, t_f]$; threshold ε; relevance vectors $\Lambda = \{(\mathbf{x}_i, y_i, \boldsymbol{\xi}_i)\}$; weight posterior mean $\boldsymbol{\mu}$ and max covariance eigenvalue λ_{max}

while True **do**

Calculate r_k using Eq. (5.6) or Eq. (5.7).

if $r_k < \varepsilon$ **then return** False (Colliding)

Solve $\|\boldsymbol{\Gamma}(\mathbf{p}(t) - \mathbf{p}(t_k))\| = r_k$ for $t_{k+1} \geq t_k$

if $t_{k+1} \geq t_f$ **then return** True (Free)

curve and a free curve, respectively.

In Prop. 3 and 4, calculating t_u and r_u takes $O(M)$ time, while the computational complexity of calculating t_u^* and r_u^* are $O(M^2)$, where $M = M^+ + M^-$. If the line segments or curves are limited to the neighborhood of the starting point $\mathbf{p}_0$, the bound t_u and r_u can reasonably approximate t_u^* and r_u^*, respectively, if $\mathbf{x}_j^-$ is chosen as the negative support vector, closest to $\mathbf{p}_0$. Calculation of t_u and r_u in Prop. 3 and 4 is efficient in the sense that it has the same complexity as classifying a point, yet it can classify an entire line segment for $t \in [0, t_u)$ and an entire ellipsoid $\mathcal{E}(\mathbf{p}_0, r_u)$, respectively.

In the next section, we will describe how to integrate the proposed collision checking algorithms for line segments (Algorithm 3) and curves (Algorithm 4) into motion planners.

5.1.3 Integration with Motion Planning Algorithms

Our collision checking algorithms for line segments (Algorithm 3) and curves (Algorithm 4) are compatible with common motion planning algorithms such as A^* [142] or RRT^* [83] to generate robot trajectories in the free space, e.g. in GetSuccessors subroutine for A^* and ObstacleFree subroutine for RRT^*, as shown in Algorithm 5.

The use of the "inflated boundary" $G_3(\mathbf{x}) = 0$ from Prop. 2 for collision checking might block the motion planning task if it is not tight enough in certain regions of the environment (e.g., unobserved regions as discussed in Section 3.5). For such regions, a different ratio of n_2/n_1 can be used in Prop. 2 to achieve a tighter bound $G_3(\mathbf{x})$. Increasing the decision threshold $\bar{e}$ (Def. 10) can also improve the accuracy of $G_3(\mathbf{x})$ if a trade-off with robot safety is allowed. Another resort is to use sampling-based collision checking, selecting points along the curve $\mathbf{p}(t)$ and using Def. 10.

Algorithm 5. GetSuccessors and ObstacleFree subroutines in A^*[142] and RRT^* [83], respectively

Input: Current position $\mathbf{p}_k$; set of relevance vectors $\Lambda = \{(\mathbf{x}_i, y_i, \boldsymbol{\xi}_i)\}$ with posterior weight mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$; set $\mathcal{N}(\mathbf{p}_k)$ of potential reference trajectories $\mathbf{p}(t - t_k)$ with $\mathbf{p}(t_k) = \mathbf{p}_k$.
Output: Set of collision-free trajectories S.

```
S ← ∅;.
for p′ in N(p_k) do
    if p′ is a line and CheckLine(p_k, p′, Λ) then                ▷ Algorithm 3
        S ← S ∪ {p′}
    if p′ is a curve and CheckCurve(p_k, p′, Λ) then              ▷ Algorithm 4
        S ← S ∪ {p′}
return S
```

5.2 Trajectory Tracking with Learned Hamiltonian Dynamics

In this section, we aim to design a control approach that achieves stabilization or trajectory tracking for different rigid-body robots, such as UGVs, UAVs, or UUVs, using the learned models in Chapter 4 without requiring prior knowledge of its parameters.

The Hamiltonian formulation and its port-Hamiltonian generalization [170], presented in Chapter 4, are built around the notion of system energy and, hence, are naturally related to control techniques for stabilization aiming to minimize the total energy. Since the minimum point of the Hamiltonian might not correspond to a desired regulation point, the control design needs to inject additional energy to ensure that the minimum of the total energy is at the desired equilibrium. For fully-actuated (port-)Hamiltonian systems, it is sufficient to shape the potential energy only using an energy-shaping and damping-injection (ES-DI) controller [170]. For underactuated systems, both the kinetic and potential energies needs to be shaped, e.g., via interconnection and damping assignment passivity-based control (IDA-PBC) [170, 128, 1, 25]. Wang and Goldsmith [175] extend the IDA-PBC controller from stabilization to trajectory tracking. Closely related to our controller design, Souza et. al. [160] apply this technique to design a controller for an underactuated quadrotor robot but use Euler angles as the orientation representation. Port-Hamiltonian structure and energy-based control design are also used to learn distributed control policies from state-control trajectories [50, 52, 148]. We connect Hamiltonian-dynamics learning in Chapter 4 with the idea of IDA-PBC control to allow

stabilization of any rigid-body robot without relying on its model parameters a priori. We design a trajectory-tracking controller for underactuated systems, e.g., quadrotor robots, based on the IDA-PBC approach and show how to construct desired pose and momentum trajectories given only desired position and yaw. We demonstrate the tight integration of dynamics learning and control to achieve closed-loop trajectory tracking with underactuated quadrotor robots.

To handle dynamics changes and disturbances, we develop data-driven adaptive control for rigid-body systems, such as unmanned ground vehicles (UGVs), unmanned aerial vehicles (UAVs), or unmanned underwater vehicles (UUVs), that satisfy Hamilton's equations of motion on position and orientation manifold $SE(3)$. While adaptation laws have been developed to work with non-parametric uncertainties in the related work, we consider linearly parameterized disturbances, i.e. linear combinations of *unknown* nonlinear features. While recent techniques for disturbance feature learning and data-driven adaptive control are restricted to systems whose states evolve in Euclidean space, a unique aspect of our adaptive control design is the consideration of geometric tracking errors on the $SE(3)$ manifold. Compared to existing $SE(3)$ geometric adaptive controllers specifically designed for quadrotors with a known disturbance model [53, 12], we develop a general adaptation law that can be used for any rigid-body robot, such as a UGV, UAV, or UUV, and learn disturbance features from trajectory data instead of assuming a known model. Specifically, given the learned nonlinear disturbance features in Chapter 4, we develop a geometric adaptation law to estimate the disturbances online and compensate them by a nonlinear energy-shaping tracking controller.

In summary, this section provides control designs to solve Problem 4 using the learned dynamics with and without the presence of disturbances or dynamics changes from new operational conditions:

- a unified energy-based control policy for learned port-Hamiltonian dynamics on a Lie group that achieves trajectory tracking if permissible by the system's degree of underactuation.
- an adaptive control policy with disturbance compensation online based on the learned disturbance features and the geometric tracking errors.

5.2.1 Control Design for Hamiltonian Dynamics on Lie Groups

Consider a desired regulation point $(\mathbf{q}^*, \mathbf{p}^*) \in \mathsf{T}^*\mathsf{G}$ that the system should be stabilized to. The Hamiltonian function $\mathcal{H}(\mathbf{q}, \mathbf{p})$, representing the total energy of the system, generally does not have a minimum at $(\mathbf{q}^*, \mathbf{p}^*)$. An IDA-PBC controller [170, 128, 175] is designed to inject additional energy $\mathcal{H}_a(\mathbf{q}, \mathbf{p})$ such that the desired total energy:

$$\mathcal{H}_d(\mathbf{q}, \mathbf{p}) = \mathcal{H}(\mathbf{q}, \mathbf{p}) + \mathcal{H}_a(\mathbf{q}, \mathbf{p}) \tag{5.9}$$

achieves its minimum at $(\mathbf{q}^*, \mathbf{p}^*)$. In other words, the closed-loop system obtained by applying the controller to the port-Hamiltonian dynamics in (2.31) should have the form:

$$\begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\mathbf{p}} \end{bmatrix} = (\mathcal{J}_d(\mathbf{q}, \mathbf{p}) - \mathcal{R}_d(\mathbf{q}, \mathbf{p})) \begin{bmatrix} \frac{\partial \mathcal{H}_d}{\partial \mathbf{q}} \\ \frac{\partial \mathcal{H}_d}{\partial \mathbf{p}} \end{bmatrix}. \tag{5.10}$$

to ensure that $(\mathbf{q}^*, \mathbf{p}^*)$ is an equilibrium. The control input $\mathbf{u}$ should be chosen so that (2.31) and (5.10) are equal. This matching equation design does not directly apply to trajectory tracking, especially for underactuated systems [160, 175].

Consider a desired trajectory $(\mathbf{q}^*(t), \mathbf{p}^*(t))$ that the system should track, where $\mathbf{q}^*(t)$ is the desired generalized coordinate and $\mathbf{p}^*(t)$ is the desired generalized momentum. Let $(\mathbf{q}_e(t), \mathbf{p}_e(t))$ denote the error in the generalized coordinates and momentum, respectively, where the coordinate error $\mathbf{q}_e = (\mathbf{q}^*)^{-1}\mathbf{q} \in \mathsf{G}$ and the momentum $\mathbf{p}_e = \mathbf{p} - \mathbf{p}^* \in \mathsf{T}^*_{\mathbf{q}_e}\mathsf{G}$. For trajectory tracking, the desired total energy $\mathcal{H}_d(\mathbf{q}_e, \mathbf{p}_e)$ is defined in terms of the error state, with desired closed-loop dynamics:

$$\begin{bmatrix} \dot{\mathbf{q}}_e \\ \dot{\mathbf{p}}_e \end{bmatrix} = (\mathcal{J}_d(\mathbf{q}_e, \mathbf{p}_e) - \mathcal{R}_d(\mathbf{q}_e, \mathbf{p}_e)) \begin{bmatrix} \frac{\partial \mathcal{H}_d}{\partial \mathbf{q}_e} \\ \frac{\partial \mathcal{H}_d}{\partial \mathbf{p}_e} \end{bmatrix}. \tag{5.11}$$

Matching (5.11) with (2.31) leads to the following requirement for the control input:

$$\begin{aligned}\mathcal{G}(\mathbf{q},\mathfrak{p})\mathbf{u} =&(\mathcal{J}_d(\mathbf{q}_e,\mathfrak{p}_e)-\mathcal{R}_d(\mathbf{q}_e,\mathfrak{p}_e))\begin{bmatrix}\frac{\partial \mathcal{H}_d}{\partial \mathbf{q}_e}\\ \frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e}\end{bmatrix}\\ &-(\mathcal{J}(\mathbf{q},\mathfrak{p})-\mathcal{R}(\mathbf{q},\mathfrak{p}))\begin{bmatrix}\frac{\partial \mathcal{H}}{\partial \mathbf{q}}\\ \frac{\partial \mathcal{H}}{\partial \mathfrak{p}}\end{bmatrix}+\begin{bmatrix}\dot{\mathbf{q}}\\ \dot{\mathfrak{p}}\end{bmatrix}-\begin{bmatrix}\dot{\mathbf{q}}_e\\ \dot{\mathfrak{p}}_e\end{bmatrix}.\end{aligned} \tag{5.12}$$

The control input can be obtained from (5.12) as the sum $\mathbf{u}=\mathbf{u}_{ES}+\mathbf{u}_{DI}$ of an energy-shaping component $\mathbf{u}_{ES}$ and a damping-injection component $\mathbf{u}_{DI}$:

$$\mathbf{u}_{ES}=\mathcal{G}^{\dagger}(\mathbf{q},\mathfrak{p})\left((\mathcal{J}_d(\mathbf{q}_e,\mathfrak{p}_e))\begin{bmatrix}\frac{\partial \mathcal{H}_d}{\partial \mathbf{q}_e}\\ \frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e}\end{bmatrix}-\begin{bmatrix}\dot{\mathbf{q}}_e\\ \dot{\mathfrak{p}}_e\end{bmatrix}-(\mathcal{J}(\mathbf{q},\mathfrak{p})-\mathcal{R}(\mathbf{q},\mathfrak{p}))\begin{bmatrix}\frac{\partial \mathcal{H}}{\partial \mathbf{q}}\\ \frac{\partial \mathcal{H}}{\partial \mathfrak{p}}\end{bmatrix}+\begin{bmatrix}\dot{\mathbf{q}}\\ \dot{\mathfrak{p}}\end{bmatrix}\right), \tag{5.13a}$$

$$\mathbf{u}_{DI}=-\mathcal{G}^{\dagger}(\mathbf{q},\mathfrak{p})\mathcal{R}_d(\mathbf{q}_e,\mathfrak{p}_e)\begin{bmatrix}\frac{\partial \mathcal{H}_d}{\partial \mathbf{q}_e}\\ \frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e}\end{bmatrix}, \tag{5.13b}$$

where $\mathcal{G}^{\dagger}(\mathbf{q},\mathfrak{p})=\left(\mathcal{G}^{\top}(\mathbf{q},\mathfrak{p})\mathcal{G}(\mathbf{q},\mathfrak{p})\right)^{-1}\mathcal{G}^{\top}(\mathbf{q},\mathfrak{p})$ is the pseudo-inverse of $\mathcal{G}(\mathbf{q},\mathfrak{p})$. The control input $\mathbf{u}_{ES}$ exists as long as the desired interconnection matrix $\mathcal{J}_d$, dissipation matrix $\mathcal{R}_d$, and total energy $\mathcal{H}_d$ satisfy the following matching condition for all $(\mathbf{q},\mathfrak{p})\in\mathsf{T}^*\mathsf{G}$ and $(\mathbf{q}_e,\mathfrak{p}_e)\in\mathsf{T}^*\mathsf{G}$:

$$\begin{aligned}\mathcal{G}^{\dagger}(\mathbf{q},\mathfrak{p})\Bigg((\mathcal{J}_d(\mathbf{q}_e,\mathfrak{p}_e)-\mathcal{R}_d(\mathbf{q}_e,\mathfrak{p}_e))\begin{bmatrix}\frac{\partial \mathcal{H}_d}{\partial \mathbf{q}_e}\\ \frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e}\end{bmatrix}\\ -(\mathcal{J}(\mathbf{q},\mathfrak{p})-\mathcal{R}(\mathbf{q},\mathfrak{p}))\begin{bmatrix}\frac{\partial \mathcal{H}}{\partial \mathbf{q}}\\ \frac{\partial \mathcal{H}}{\partial \mathfrak{p}}\end{bmatrix}+\begin{bmatrix}\dot{\mathbf{q}}\\ \dot{\mathfrak{p}}\end{bmatrix}-\begin{bmatrix}\dot{\mathbf{q}}_e\\ \dot{\mathfrak{p}}_e\end{bmatrix}\Bigg)=0.\end{aligned} \tag{5.14}$$

where $\mathcal{G}^{\perp}(\mathbf{q},\mathfrak{p})$ is a maximal-rank left annihilator of $\mathcal{G}(\mathbf{q},\mathfrak{p})$, i.e., $\mathcal{G}^{\perp}(\mathbf{q},\mathfrak{p})\mathcal{G}(\mathbf{q},\mathfrak{p})=\mathbf{0}$.

5.2.2 Control Design for Hamiltonian Dynamics on the SE(3) Manifold

Consider a desired state trajectory $\mathbf{s}^*(t)=(\mathbf{q}^*(t),\boldsymbol{\zeta}^*(t))$ that the system should track where $\mathbf{q}^*(t)\in SE(3)$ is the desired pose and $\boldsymbol{\zeta}^*(t)=\begin{bmatrix}\mathbf{v}^*(t)^{\top} & \boldsymbol{\omega}^*(t)^{\top}\end{bmatrix}^{\top}$ is the desired gen-

eralized velocity expressed in the desired frame. Let $\mathfrak{p}^* = \mathbf{M}\begin{bmatrix}\mathbf{R}^\top\mathbf{R}^*\mathbf{v}^* \\ \mathbf{R}^\top\mathbf{R}^*\boldsymbol{\omega}^*\end{bmatrix}$ denote the desired momentum, defined based on (2.36) with the desired velocity expressed in the body frame. Let $\mathbf{p}_e = \mathbf{p} - \mathbf{p}^*$ and $\mathbf{R}_e = \mathbf{R}^{*\top}\mathbf{R} = \begin{bmatrix}\mathbf{r}_{e1} & \mathbf{r}_{e2} & \mathbf{r}_{e3}\end{bmatrix}^\top$ be the position error and rotation error between the current orientation $\mathbf{R}$ and the desired one $\mathbf{R}^*$, respectively. The vectorized error $\mathfrak{q}_e$ in the generalized coordinates is:

$$\mathfrak{q}_e = \begin{bmatrix}\mathbf{p}_e^{*\top} & \mathbf{r}_{e1}^\top & \mathbf{r}_{e2}^\top & \mathbf{r}_{e3}^\top\end{bmatrix}^\top. \tag{5.15}$$

The error in the generalized momenta is $\mathfrak{p}_e = \mathfrak{p} - \mathfrak{p}^*$, described in the body frame. The desired total energy is defined in terms of the error state as:

$$\mathcal{H}_d(\mathfrak{q}_e, \mathfrak{p}_e) = \frac{1}{2}\mathfrak{p}_e^\top \mathbf{M}_d^{-1}(\mathfrak{q}_e)\mathfrak{p}_e + V_d(\mathfrak{q}_e), \tag{5.16}$$

where $\mathbf{M}_d(\mathfrak{q}_e)$ and $V_d(\mathfrak{q}_e)$ are the desired generalized mass and potential energy. Choosing the following desired inter-connection matrix and dissipation matrix:

$$\mathcal{J}_d(\mathfrak{q}_e, \mathfrak{p}_e) = \begin{bmatrix}\mathbf{0} & \mathbf{J}_1 \\ -\mathbf{J}_1^\top & \mathbf{J}_2\end{bmatrix}, \quad \mathcal{R}_d(\mathfrak{q}_e, \mathfrak{p}_e) = \begin{bmatrix}\mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{K_d}\end{bmatrix}, \tag{5.17}$$

and plugging $\mathcal{J}(\mathfrak{q}, \mathfrak{p})$ and $\mathcal{R}(\mathfrak{q}, \mathfrak{p})$ from (2.38) into the matching equations in (5.12), leads to:

$$\mathbf{0} = \mathbf{J}_1\frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e} - \mathfrak{q}^\times\frac{\partial \mathcal{H}}{\partial \mathfrak{p}} + \dot{\mathfrak{q}} - \dot{\mathfrak{q}}_e, \tag{5.18}$$

$$\begin{aligned}\mathbf{B}(\mathfrak{q})\mathbf{u} = {} & \mathfrak{q}^{\times\top}\frac{\partial \mathcal{H}}{\partial \mathfrak{q}} - \mathbf{J}_1^\top\frac{\partial \mathcal{H}_d}{\partial \mathfrak{q}_e} + \mathbf{J}_2\frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e} - \mathfrak{p}^\times\frac{\partial \mathcal{H}}{\partial \mathfrak{p}} \\ & -\mathbf{K_d}\frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e} + \mathbf{D}(\mathfrak{q}, \mathfrak{p})\frac{\partial \mathcal{H}}{\partial \mathfrak{p}} + \dot{\mathfrak{p}} - \dot{\mathfrak{p}}_e.\end{aligned} \tag{5.19}$$

Assuming $\mathbf{M}_d(\mathbf{q}_e) = \mathbf{M}(\mathbf{q})$, (5.18) is satisfied if we choose $\mathbf{J}_1 = \begin{bmatrix} \mathbf{R}^\top & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{r}}_{e1}^\top & \hat{\mathbf{r}}_{e2}^\top & \hat{\mathbf{r}}_{e3}^\top \end{bmatrix}^\top$. Indeed, we have $\dot{\mathbf{q}} = \mathbf{q}^\times \frac{\partial \mathcal{H}}{\partial \mathfrak{p}}$ (from (11) and (19)) and

$$\frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e} = \mathbf{M}_d^{-1}\mathfrak{p}_e = \begin{bmatrix} \mathbf{v}_e \\ \boldsymbol{\omega}_e \end{bmatrix} = \begin{bmatrix} \mathbf{v} - \mathbf{R}^\top \mathbf{R}^* \mathbf{v}^* \\ \boldsymbol{\omega} - \mathbf{R}^\top \mathbf{R}^* \boldsymbol{\omega}^* \end{bmatrix}.$$

The error dynamics becomes:

$$\dot{\mathbf{q}}_e = \begin{bmatrix} \dot{\mathbf{p}} - \dot{\mathbf{p}}^* \\ \dot{\mathbf{r}}_{e1} \\ \dot{\mathbf{r}}_{e2} \\ \dot{\mathbf{r}}_{e3} \end{bmatrix} = \begin{bmatrix} \mathbf{R}^\top & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{r}}_{e1}^\top & \hat{\mathbf{r}}_{e2}^\top & \hat{\mathbf{r}}_{e3}^\top \end{bmatrix}^\top \begin{bmatrix} \mathbf{v}_e \\ \boldsymbol{\omega}_e \end{bmatrix} = \mathbf{J}_1 \frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e}, \tag{5.20}$$

since $\dot{\mathbf{R}}_e = \frac{d}{dt}(\mathbf{R}_e) = \mathbf{R}_e \widehat{\boldsymbol{\omega}}_e$ as shown in [90, Section III-A] and $\dot{\mathbf{p}} - \dot{\mathbf{p}}^* = \mathbf{R}\mathbf{v} - \mathbf{R}^*\mathbf{v}^* = \mathbf{R}\mathbf{v}_e$.

The desired control input can be obtained from (5.19) as $\mathbf{u} = \mathbf{u}_{ES} + \mathbf{u}_{DI}$ with:

$$\mathbf{u}_{ES} = \mathbf{B}^\dagger(\mathbf{q}) \left(\mathbf{q}^{\times\top} \frac{\partial \mathcal{H}}{\partial \mathbf{q}} - \mathbf{J}_1^\top \frac{\partial \mathcal{H}_d}{\partial \mathbf{q}_e} + \mathbf{J}_2 \frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e} - \mathfrak{p}^\times \frac{\partial \mathcal{H}}{\partial \mathfrak{p}} + \mathbf{D}(\mathbf{q}, \mathfrak{p}) \frac{\partial \mathcal{H}}{\partial \mathfrak{p}} + \dot{\mathfrak{p}} - \dot{\mathfrak{p}}_e \right), \tag{5.21a}$$

$$\mathbf{u}_{DI} = -\mathbf{B}^\dagger(\mathbf{q}) \mathbf{K_d} \frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e}, \tag{5.21b}$$

where $\mathbf{B}^\dagger(\mathbf{q}) = \left(\mathbf{B}^\top(\mathbf{q})\mathbf{B}(\mathbf{q})\right)^{-1} \mathbf{B}^\top(\mathbf{q})$ is the pseudo-inverse of $\mathbf{B}(\mathbf{q})$. The matching condition (5.14) becomes:

$$\mathbf{B}^\perp(\mathbf{q}) \left(\mathbf{q}^{\times\top} \frac{\partial \mathcal{H}}{\partial \mathbf{q}} - \mathbf{J}_1^\top \frac{\partial \mathcal{H}_d}{\partial \mathbf{q}_e} + \mathbf{J}_2 \frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e} - \mathfrak{p}^\times \frac{\partial \mathcal{H}}{\partial \mathfrak{p}} + \dot{\mathfrak{p}} - \dot{\mathfrak{p}}_e \right) = 0. \tag{5.22}$$

In this chapter, we reshape the open-loop Hamiltonian $\mathcal{H}(\mathbf{q}, \mathfrak{p})$ into the following desired total energy $\mathcal{H}_d(\mathbf{q}_e, \mathfrak{p}_e)$, minimized along the desired trajectory:

$$\mathcal{H}_d(\mathbf{q}_e, \mathfrak{p}_e) = \frac{1}{2}(\mathbf{p}-\mathbf{p}^*)^\top \mathbf{K_p}(\mathbf{p}-\mathbf{p}^*) + \frac{1}{2}\operatorname{tr}(\mathbf{K_R}(\mathbf{I}-\mathbf{R}^{*\top}\mathbf{R})) + \frac{1}{2}(\mathfrak{p}-\mathfrak{p}^*)^\top \mathbf{M}^{-1}(\mathbf{q})(\mathfrak{p}-\mathfrak{p}^*), \tag{5.23}$$

where $\mathbf{K_p}, \mathbf{K_R} \succ 0$ are positive-definite matrices.

For an $SE(3)$ rigid-body system with constant generalized mass matrix $\mathbf{M}_d = \mathbf{M}$ and $\mathbf{J}_2 = 0$, which is a common choice, the energy-shaping term in (5.21a) and the damping-injection term in (5.21b) simplify as:

$$\begin{aligned}
\mathbf{u}_{ES}(\mathfrak{q}, \mathfrak{p}) &= \mathbf{B}^{\dagger}(\mathfrak{q}) \left(\mathfrak{q}^{\times\top} \frac{\partial V}{\partial \mathfrak{q}} - \left(\mathfrak{p}^{\times} - \mathbf{D}(\mathfrak{q}, \mathfrak{p}) \right) \mathbf{M}^{-1} \mathfrak{p} - \mathbf{e}(\mathfrak{q}, \mathfrak{q}^*) + \dot{\mathfrak{p}}^* \right), \\
\mathbf{u}_{DI}(\mathfrak{q}, \mathfrak{p}) &= -\mathbf{B}^{\dagger}(\mathfrak{q}) \mathbf{K_d} \mathbf{M}^{-1} (\mathfrak{p} - \mathfrak{p}^*),
\end{aligned} \tag{5.24}$$

where the generalized coordinate error between $\mathfrak{q}$ and $\mathfrak{q}^*$ is:

$$\mathbf{e}(\mathfrak{q}, \mathfrak{q}^*) := \mathbf{J}_1^{\top} \frac{\partial V_d}{\partial \mathfrak{q}_e} = \begin{bmatrix} \mathbf{R}^{\top} \mathbf{K_p} (\mathbf{p} - \mathbf{p}^*) \\ \frac{1}{2} \left(\mathbf{K_R} \mathbf{R}^{*\top} \mathbf{R} - \mathbf{R}^{\top} \mathbf{R}^* \mathbf{K_R}^{\top} \right)^{\vee} \end{bmatrix}, \tag{5.25}$$

and the derivative of the desired momentum is:

$$\dot{\mathfrak{p}}^* = \mathbf{M} \begin{bmatrix} \mathbf{R}^{\top} \ddot{\mathbf{p}}^* - \hat{\boldsymbol{\omega}} \mathbf{R}^{\top} \dot{\mathbf{p}}^* \\ \mathbf{R}^{\top} \mathbf{R}^* \dot{\boldsymbol{\omega}}^* - \hat{\boldsymbol{\omega}}_e \mathbf{R}^{\top} \mathbf{R}^* \boldsymbol{\omega}^* \end{bmatrix}. \tag{5.26}$$

By expanding the terms in (5.24), we have:

$$\mathfrak{p}^{\times} \mathbf{M}^{-1} \mathfrak{p} = \mathfrak{p}^{\times} \boldsymbol{\zeta} = \begin{bmatrix} \hat{\mathbf{p}}_{\mathbf{v}} \boldsymbol{\omega} \\ \hat{\mathbf{p}}_{\boldsymbol{\omega}} \boldsymbol{\omega} + \hat{\mathbf{p}}_{\mathbf{v}} \mathbf{v} \end{bmatrix}, \tag{5.27}$$

$$\mathbf{M}^{-1} (\mathfrak{p} - \mathfrak{p}^*) = \begin{bmatrix} \mathbf{v} - \mathbf{R}^{\top} \dot{\mathbf{p}}^* \\ \boldsymbol{\omega} - \mathbf{R}^{\top} \mathbf{R}^* \boldsymbol{\omega}^* \end{bmatrix}, \tag{5.28}$$

$$\mathfrak{q}^{\times\top} \frac{\partial \mathcal{V}}{\partial \mathfrak{q}} = \begin{bmatrix} \mathbf{R}^{\top} \frac{\partial \mathcal{V}(\mathfrak{q})}{\partial \mathbf{p}} \\ \sum_{i=1}^{3} \hat{\mathbf{r}}_i \frac{\partial \mathcal{V}(\mathfrak{q})}{\partial \mathbf{r}_i} \end{bmatrix}. \tag{5.29}$$

Theorem 1. *Consider a port-Hamiltonian system on the $SE(3)$ manifold with dynamics (2.40). Assume that the matching condition (5.22) is satisfied, the desired momentum's derivative $\dot{\mathfrak{p}}^*$ is bounded, and the matrices $\mathbf{K_p}$, $\mathbf{K_R}$, and $\mathbf{K_d}$ are positive-definite. The control policy in (5.21) leads to closed-loop error dynamics in (5.11), (5.17). The tracking error $(\mathfrak{q}_e, \mathfrak{p}_e) = ((\mathbf{p}_e, \mathbf{R}_e), \mathfrak{p}_e)$*

asymptotically stabilizes to $((\mathbf{0}, \mathbf{I}), \mathbf{0})$ *with Lyapunov function given by the desired Hamiltonian* $\mathcal{H}_d(\mathbf{q}_e, \mathfrak{p}_e)$ *in* (5.23).

Proof. Since the matching condition is satisfied and the desired momentum's derivative $\dot{\mathfrak{p}}^*$ is bounded, the control policy in (5.21), (5.21b) exists and achieves the desired closed-loop error dynamics:

$$\begin{bmatrix} \dot{\mathbf{q}}_e \\ \dot{\mathfrak{p}}_e \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{J}_1 \\ -\mathbf{J}_1^\top & \mathbf{J}_2 - \mathbf{K}_d \end{bmatrix} \begin{bmatrix} \frac{\partial \mathcal{H}_d}{\partial \mathbf{q}_e} \\ \frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e} \end{bmatrix}. \tag{5.30}$$

We have $\mathrm{tr}\left(\mathbf{I} - \mathbf{R}^{*\top}\mathbf{R}\right) \geq 0$, as all entries in $\mathbf{R}_e \in SO(3)$ are less than 1. Since $\mathbf{M}$, $\mathbf{K_p}$ and $\mathbf{K_R}$ are positive-definite matrices, the desired Hamiltonian $\mathcal{H}_d$ is positive-definite, and achieves minimum value 0 only at $\mathbf{q}_e = (\mathbf{0}, \mathbf{I})$ and $\mathfrak{p}_e = \mathbf{0}$, i.e., no position, rotation and momentum errors. The time derivative of $\mathcal{H}_d(\mathbf{q}_e, \mathfrak{p}_e)$ can be computed as:

$$\begin{aligned} \dot{\mathcal{H}}_d(\mathbf{q}_e, \mathfrak{p}_e) &= \frac{\partial \mathcal{H}_d}{\partial \mathbf{q}_e}^\top \dot{\mathbf{q}}_e + \frac{\partial \mathcal{H}_d}{\partial \mathfrak{p}_e}^\top \dot{\mathfrak{p}}_e \\ &= -\mathfrak{p}_e^\top \mathbf{M}^{-1}(\mathbf{q}) \mathbf{K}_d \mathbf{M}^{-1}(\mathbf{q}) \mathfrak{p}_e. \end{aligned} \tag{5.31}$$

As $\mathbf{K}_d$ and $\mathbf{M}(\mathbf{q})$ are positive-definite, we have $\dot{\mathcal{H}}_d(\mathbf{q}_e, \mathfrak{p}_e) \leq 0$ for all $(\mathbf{q}_e, \mathfrak{p}_e)$ and equality holds at $((\mathbf{0}, \mathbf{I}), \mathbf{0})$. By LaSalle's invariance principle [86], the tracking errors $(\mathbf{q}_e, \mathfrak{p}_e)$ asymptotically converge to $((\mathbf{0}, \mathbf{I}), \mathbf{0})$. □

Without requiring a priori knowledge of the system parameters, the control design in (5.21) offers a unified control approach for $SE(3)$ Hamiltonian systems that achieves trajectory tracking, if permissible by the system's degree of underactuation. Thus, our control design solves Problem 4 for rigid-body robot systems, such as UGVs, UAVs, and UUVs, with tracking performance guaranteed by Theorem 1.

5.2.3 Adaptive Control with Learned Disturbance Model on the SE(3) Manifold

Given the learned disturbance model $\mathbf{W}_{\boldsymbol{\theta}}(\mathbf{s})$ in Chapter 4 and a desired trajectory $\mathbf{s}^*(t)$, we develop a trajectory tracking controller $\mathbf{u} = \boldsymbol{\pi}(\mathbf{s}, \mathbf{s}^*, \mathbf{a}; \boldsymbol{\theta})$ that compensates for disturbances and an adaptation law $\dot{\mathbf{a}} = \boldsymbol{\rho}(\mathbf{s}, \mathbf{s}^*, \mathbf{a}; \boldsymbol{\theta})$ that estimates the disturbance realization online. Our

tracking controller for the Hamiltonian dynamics on $SE(3)$ in (2.40) is developed using interconnection and damping assignment passivity-based control (IDA-PBC) [170]. Consider a desired pose-velocity trajectory $(\boldsymbol{\mathfrak{q}}^*(t), \boldsymbol{\zeta}^*(t))$. We extend the trajectory tracking controller (5.24) in Section 5.2.2 to compensate for the disturbance forces, and arrive at a tracking controller $\mathbf{u} = \boldsymbol{\pi}(\mathbf{s}, \mathbf{s}^*, \mathbf{a}; \boldsymbol{\theta})$, consisting of an energy-shaping term $\mathbf{u}_{ES}$, a damping-injection term $\mathbf{u}_{DI}$, and a disturbance compensation term $\mathbf{u}_{DC}$:

$$\begin{aligned}
\mathbf{u}_{ES}(\boldsymbol{\mathfrak{q}}, \boldsymbol{\mathfrak{p}}) &= \mathbf{B}^\dagger(\boldsymbol{\mathfrak{q}}) \left(\boldsymbol{\mathfrak{q}}^{\times\top} \frac{\partial V}{\partial \boldsymbol{\mathfrak{q}}} - \left(\boldsymbol{\mathfrak{p}}^\times - \mathbf{D}(\boldsymbol{\mathfrak{q}}, \boldsymbol{\mathfrak{p}}) \right) \mathbf{M}^{-1} \boldsymbol{\mathfrak{p}} - \mathbf{e}(\boldsymbol{\mathfrak{q}}, \boldsymbol{\mathfrak{q}}^*) + \dot{\boldsymbol{\mathfrak{p}}}^* \right), \\
\mathbf{u}_{DI}(\boldsymbol{\mathfrak{q}}, \boldsymbol{\mathfrak{p}}) &= -\mathbf{B}^\dagger(\boldsymbol{\mathfrak{q}}) \mathbf{K_d} \mathbf{M}^{-1} (\boldsymbol{\mathfrak{p}} - \boldsymbol{\mathfrak{p}}^*),
\end{aligned} \tag{5.32}$$

where $\mathbf{B}^\dagger(\boldsymbol{\mathfrak{q}}) = \left(\mathbf{B}^\top(\boldsymbol{\mathfrak{q}}) \mathbf{B}(\boldsymbol{\mathfrak{q}}) \right)^{-1} \mathbf{B}^\top(\boldsymbol{\mathfrak{q}})$ is the pseudo-inverse of $\mathbf{B}(\boldsymbol{\mathfrak{q}})$ and $\mathbf{K_d} = \text{diag}(k_\mathbf{v}\mathbf{I}, k_{\boldsymbol{\omega}}\mathbf{I})$ is a damping gain with positive terms $k_\mathbf{v}$, $k_{\boldsymbol{\omega}}$. Note that in this section, as energy dissipation can be considered as source of disturbance, we ignore the dissipation matrix $\mathbf{D}(\boldsymbol{\mathfrak{q}}, \boldsymbol{\mathfrak{p}})$ for simplicity. The controller utilizes a generalized coordinate error between $\boldsymbol{\mathfrak{q}}$ and $\boldsymbol{\mathfrak{q}}^*$:

$$\mathbf{e}(\boldsymbol{\mathfrak{q}}, \boldsymbol{\mathfrak{q}}^*) = \begin{bmatrix} \mathbf{e}_\mathbf{p}(\boldsymbol{\mathfrak{q}}, \boldsymbol{\mathfrak{q}}^*) \\ \mathbf{e}_\mathbf{R}(\boldsymbol{\mathfrak{q}}, \boldsymbol{\mathfrak{q}}^*) \end{bmatrix} = \begin{bmatrix} k_\mathbf{p} \mathbf{R}^\top (\mathbf{p} - \mathbf{p}^*) \\ \frac{1}{2} k_\mathbf{R} \left(\mathbf{R}^{*\top} \mathbf{R} - \mathbf{R}^\top \mathbf{R}^* \right)^\vee \end{bmatrix} \tag{5.33}$$

and a generalized momentum error $\boldsymbol{\mathfrak{p}}_e = \boldsymbol{\mathfrak{p}} - \boldsymbol{\mathfrak{p}}^*$:

$$\boldsymbol{\mathfrak{p}}_e = \mathbf{M}(\boldsymbol{\mathfrak{q}}) \begin{bmatrix} \mathbf{e}_\mathbf{v}(\mathbf{s}, \mathbf{s}^*) \\ \mathbf{e}_{\boldsymbol{\omega}}(\mathbf{s}, \mathbf{s}^*) \end{bmatrix} = \mathbf{M}(\boldsymbol{\mathfrak{q}}) \begin{bmatrix} \mathbf{v} - \mathbf{R}^\top \mathbf{R}^* \mathbf{v}^* \\ \boldsymbol{\omega} - \mathbf{R}^\top \mathbf{R}^* \boldsymbol{\omega}^* \end{bmatrix}. \tag{5.34}$$

Please refer to [37] for a detailed derivation of $\mathbf{u}_{ES}$ and $\mathbf{u}_{DI}$.

The disturbance compensation term $\mathbf{u}_{DC}$ in (5.32) requires online estimation of the disturbance feature weights $\mathbf{a}$. Inspired by [53], we design an adaptation law which utilizes the geometric errors (5.33), (5.34) to update the weights $\mathbf{a}$:

$$\dot{\mathbf{a}} = \boldsymbol{\rho}(\mathbf{s}, \mathbf{s}^*, \mathbf{a}; \boldsymbol{\theta}) = \mathbf{W}_{\boldsymbol{\theta}}^\top(\boldsymbol{\mathfrak{q}}, \boldsymbol{\mathfrak{p}}) \begin{bmatrix} c_\mathbf{p} \mathbf{e}_\mathbf{p}(\boldsymbol{\mathfrak{q}}, \boldsymbol{\mathfrak{q}}^*) + c_\mathbf{v} \mathbf{e}_\mathbf{v}(\mathbf{s}, \mathbf{s}^*) \\ c_\mathbf{R} \mathbf{e}_\mathbf{R}(\boldsymbol{\mathfrak{q}}, \boldsymbol{\mathfrak{q}}^*) + c_{\boldsymbol{\omega}} \mathbf{e}_{\boldsymbol{\omega}}(\mathbf{s}, \mathbf{s}^*) \end{bmatrix}, \tag{5.35}$$

where $c_{\mathbf{p}}$, $c_{\mathbf{v}}$, $c_{\mathbf{R}}$, $c_{\boldsymbol{\omega}}$ are positive coefficients. The stability of our adaptive controller $(\boldsymbol{\pi}, \boldsymbol{\rho})$ is shown in Theorem 2, under the assumption that the learned disturbance features $\mathbf{W}_{\boldsymbol{\theta}}$ converge to the true ones $\mathbf{W}(\mathbf{q}, \mathfrak{p})$ after the training process.

Theorem 2. *Consider the Hamiltonian dynamics in* (2.40) *with disturbance model in* (4.13). *Suppose that the parameters* $\mathbf{B}(\mathbf{q})$, $\mathbf{M}(\mathbf{q})$, $V(\mathbf{q})$ *and* $\mathbf{W}(\mathbf{q}, \mathfrak{p})$ *are known but the disturbance feature weights* $\mathbf{a}^*$ *are unknown. Let* $\mathbf{s}^*(t)$ *be a desired state trajectory with bounded angular velocity,* $\|\boldsymbol{\omega}^*(t)\| \leq \gamma$. *Assume that the initial system state lies in the domain* $\mathcal{T} = \{\mathbf{s} \in T^*SE(3) \mid \boldsymbol{\Psi}(\mathbf{R}, \mathbf{R}^*) < \alpha < 2, \|\mathbf{e}_{\boldsymbol{\omega}}(\mathbf{s}, \mathbf{s}^*)\| < \beta\}$ *for some positive constants* α *and* β, *with* $\boldsymbol{\Psi}(\mathbf{R}, \mathbf{R}^*) = \frac{1}{2}\operatorname{tr}(\mathbf{I} - \mathbf{R}^{*\top}\mathbf{R})$. *Consider the tracking controller in* (5.32) *with adaptation law in* (5.35). *Then, there exist positive constants* $k_{\mathbf{p}}$, $k_{\mathbf{R}}$, $k_{\mathbf{v}}$, $k_{\boldsymbol{\omega}}$, $c_{\mathbf{p}} = c_{\mathbf{R}} = c_1$, $c_{\mathbf{v}} = c_{\boldsymbol{\omega}} = c_2$ *such that the tracking errors* $\mathbf{e}(\mathbf{q}, \mathbf{q}^*)$ *and* $\mathfrak{p}_e$ *defined in* (5.33) *and* (5.34) *converge to zero. Also, the estimation error* $\mathbf{e}_{\mathbf{a}} = \mathbf{a} - \mathbf{a}^*$ *is stable in the sense of Lyapunov and uniformly bounded. An estimate of the region of attraction is* $\mathcal{R} = \{\mathbf{s} \in \mathcal{T} \mid \mathcal{V}(\mathbf{s}) \leq \delta\}$, *where:*

$$\mathcal{V}(\mathbf{q}, \mathfrak{p}) = \mathcal{H}_d(\mathbf{q}, \mathfrak{p}) + \frac{c_1}{c_2}\mathbf{e}^\top \mathfrak{p}_e + \frac{1}{2c_2}\|\mathbf{e}_{\mathbf{a}}\|_2^2 \tag{5.36}$$

and $\delta < \lambda_{\min}(\mathbf{Q}_1)\min(\alpha(2-\alpha)k_{\mathbf{R}}^2, \beta^2\lambda_{\min}^2(\mathbf{M}(\mathbf{q})))/2$ *for*

$$\mathbf{Q}_1 = \begin{bmatrix} \min\left\{k_{\mathbf{p}}^{-1}, k_{\mathbf{R}}^{-1}\right\} & -c_1/c_2 \\ -c_1/c_2 & \lambda_{\min}(\mathbf{M}^{-1}(\mathbf{q})) \end{bmatrix}. \tag{5.37}$$

Proof. Please refer to Appendix D. □

5.3 Autonomous Navigation

This section presents our complete online mapping and navigation approach that solves the autonomous navigation problem defined in Section 1.2. Given the sparse Bayesian kernel-based map m_k (Algorithm 2) proposed in Chapter 3, a motion planning algorithm such as A^* [142] or RRT^* [83] may be used with our collision-checking algorithms in Section 5.1 to generate a path that solves the autonomous navigation problem (Algorithm 5). The robot follows the path for some time using the tracking controllers in Section 5.2 and updates the map

Table 5.1. Comparison of sampling-based (SB) method with baseline maps and ours with sampling interval $\Delta = 0.005$.

Map	Feature Dimension	Covariance matrix	Method	Lines	Curves
SBHM	5600	full	SB	380 ms	384 ms
SBHM	5600	diag.	SB	38 ms	37 ms
LARD-HM	3640	-	SB	5.6 ms	6.3 ms
LARD-HM	5460	-	SB	6.2 ms	6.7 ms
LARD-HM	7280	-	SB	6.2 ms	6.5 ms
SBKM	3492	full	ours	7 μs	18 μs
SBKM	3492	λ_{max}	ours	7 μs	18 μs
OM	-	full	SB	21 μs	23 μs

estimate m_{k+1} with new observations. Using the updated map, the robot re-plans the path and follows the new path instead. This process is repeated until the goal is reached or a time limit is exceeded (Algorithm 6).

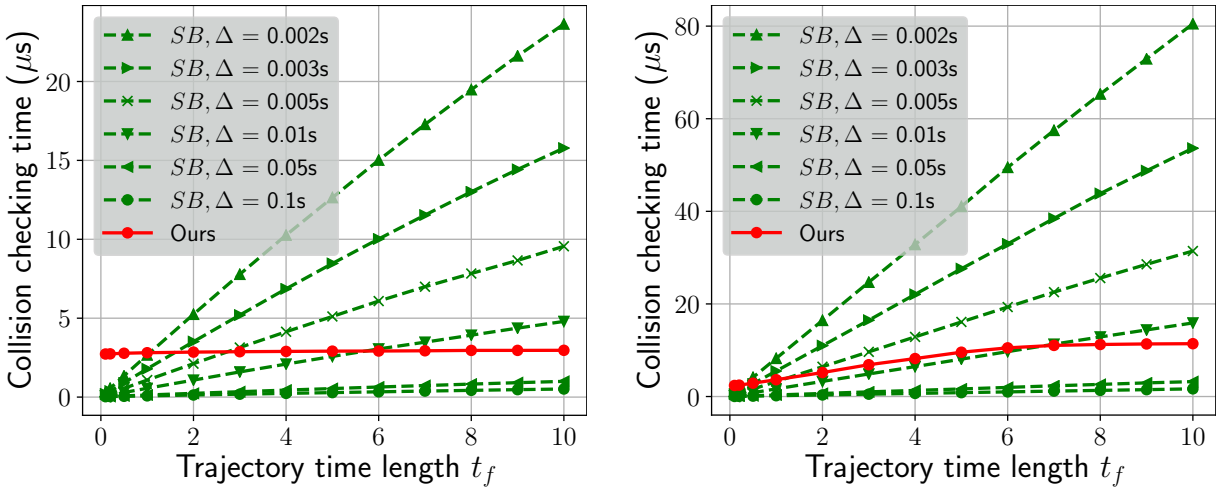
Algorithm 6. Autonomous Mapping and Navigation with a Sparse Bayesian Kernel-based Map

Input: Initial state $\mathbf{s}_0 \in \bar{\mathcal{S}}$; goal region $\mathcal{G}$; prior relevance vectors Λ_0.

1: **for** $k = 0, 1, \ldots$ **do**
2: **if** $\mathbf{s}_k \in \mathcal{G}$ **then break**
3: $\mathbf{z}_k \leftarrow$ new range sensor observation
4: $\mathcal{D}_\mathbf{k} \leftarrow$ Training Data Generation$(\mathbf{z}_k, \mathbf{s}_k)$ ▷ Section 3.4
5: $\Lambda_{k+1} \leftarrow$ Online Map Update$(\Lambda_k, \mathcal{D}_k)$ ▷ Algorithm 2, Chapter 3
6: Path Planning$(\Lambda_{k+1}, \mathbf{s}_k, \mathcal{G})$ ▷ Algorithm 5, Section 5.1
7: Follow the path using a trajectory tracking controller (with learned dynamics). ▷ Section 5.2 and 4.2

5.4 Evaluation

In this section, we verify the effectiveness of our collision checking algorithms (Section 5.4.1) and control designs (Section 5.4.2 and 5.4.3) with simulated ground and aerial robots. We also demonstrate our autonomous navigation approach on real ground and quadrotor platforms (Section 5.4.4 and 5.4.5). Finally, we illustrate the benefits of the uncertainty from our sparse probabilistic occupancy map in an active mapping task (Section 5.4.6).



(a) Checking line segments $\mathbf{p}(t) = \mathbf{p}_0 + \mathbf{v}t$ for $t \in [0, t_f]$ with various t_f values.

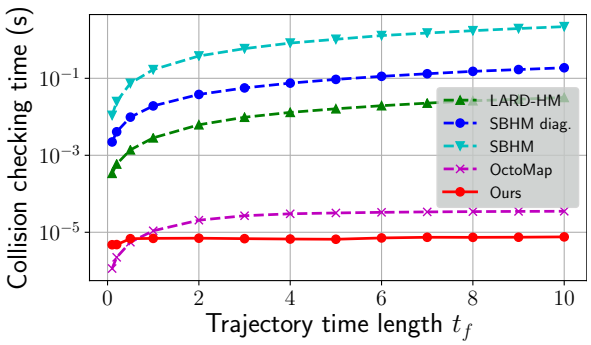
(b) Checking 2^{nd}-order polynomial curves $\mathbf{p}(t) = \mathbf{p}_0 + \mathbf{v}t + \mathbf{a}t^2$ for $t \in [0, t_f]$ with various t_f values.

Figure 5.2. Comparison between sampling-based (SB) method and ours with different sampling intervals Δ.

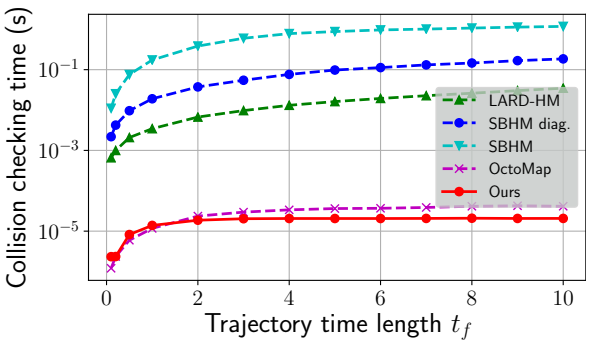
5.4.1 Effectiveness of Collision Checking Algorithms

We compared the average collision checking time over one million random line segments $\mathbf{p}(t) = \mathbf{p}_0 + \mathbf{v}t$ and one million random second order polynomial curves $\mathbf{p}(t) = \mathbf{p}_0 + \mathbf{v}t + \mathbf{a}t^2$ for $t \in [0, t_f]$ using our complete method (Algorithm 3 with Eq. (5.5) for line segments, Algorithm 4 with Eq. (5.7) for curves, $K^+ + K^- = 10$ for score approximation, and $e = -0.01$ for occupancy threshold) and sampling-based methods with different sampling resolutions using the ground truth map. Figure 5.2a and 5.2b show that the time for sampling-based collision checking increased as the time length t_f increased or the sampling resolution decreased. Meanwhile, our method's time was stable at $\sim$ 3 μs for checking line segments and at $\sim$ 11 μs for checking second-order polynomial curves suggesting our collision checking algorithms' suitability for real-time applications.

An advantage of our SBKM map representation is that it can utilize the collision-checking techniques for lines and curves developed in Section 5.1. We checked 1000 random line segments $\mathbf{p}(t) = \mathbf{p}_0 + \mathbf{v}t$ and 1000 second-order polynomial curves $\mathbf{p}(t) = \mathbf{p}_0 + \mathbf{v}t + \mathbf{a}t^2$ for $t \in [0, t_f]$ for collisions using our method (Algorithm 3 with Eq. (5.5) for line segments, Algorithm 4 with Eq. (5.7) for curves, $K^+ + K^- = 20$ for score approximation, and $e = -0.01$ for occupancy threshold)

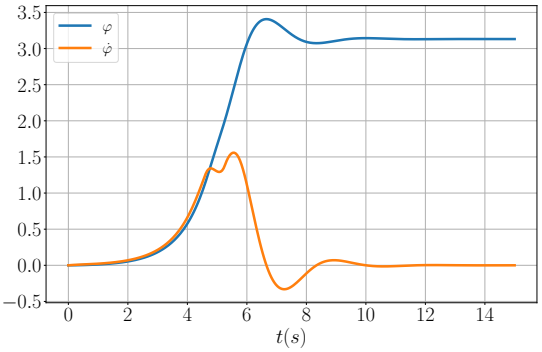


(a) Checking line segments $\mathbf{p}(t) = \mathbf{p}_0 + \mathbf{v}t$ for $t \in [0, t_f]$ with various t_f values.

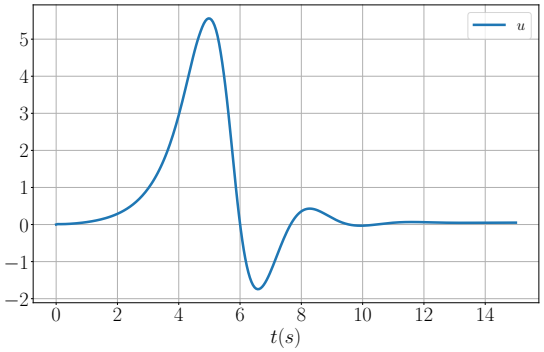


(b) Checking 2^{nd}-order polynomial curves $\mathbf{p}(t) = \mathbf{p}_0 + \mathbf{v}t + \mathbf{a}t^2$ for $t \in [0, t_f]$ with various t_f values.

Figure 5.3. Comparison between sampling-based (SB) method with baseline maps and ours with sampling interval $\Delta = 0.005$.



(a) Angle φ and velocity $\dot{\varphi}$.



(b) Control input u.

Figure 5.4. Evaluation of our energy-based controller on a pendulum system.

and sampling-based methods with sampling resolution $\Delta = 0.005\,s$ using the OctoMap, SBHM, and LARD-HM maps. Figure 5.3a and 5.3b show that the collision-checking time for sampling-based methods increased as the time length t_f increased. Meanwhile, our method's time was stable at $\sim 7\ \mu s$ for checking line segments and at $\sim 20\ \mu s$ for checking second-order polynomial curves, as shown in Table 5.1.

5.4.2 Effectiveness of Trajectory Tracking Control Design

We evaluate our trajectory-tracking control policy on a fully-actuated pendulum and an under-actuated quadrotor.

Pendulum

We tested stabilization of the pendulum based on the learned dynamics to the stable equilibrium at the downward position $\varphi = 0$ and to the unstable equilibrium at the upward position $\varphi = \pi$. Since the pendulum is a fully-actuated system and the desired state has zero velocity, potential energy shaping is enough to drive the system to the desired state $(\mathbf{q}^*, \mathbf{0})$. Our energy-based controller in (5.24) achieves the task with the additional energy $\mathcal{H}_a(\mathbf{q}, \mathfrak{p})$ simplified by removing the position error:

$$\mathcal{H}_a(\mathbf{q}, \mathfrak{p}) = -V(\mathbf{q}) + \frac{1}{2}\operatorname{tr}(\mathbf{K_R}(\mathbf{I} - \mathbf{R}^{*\top}\mathbf{R})) + \frac{1}{2}\mathfrak{p}_{\boldsymbol{\omega}}^{\top}\mathbf{M}^{-1}(\mathbf{q})\mathfrak{p}_{\boldsymbol{\omega}}. \tag{5.38}$$

The controlled angle φ and angular velocity $\dot{\varphi}$ as well as the control inputs $\mathbf{u}$ with gains $\mathbf{K_R} = 2\mathbf{I}$ and $\mathbf{K_d} = \mathbf{I}$ are shown over time in Figure 5.4a and 5.4b. We can see that the controller was able to smoothly drive the pendulum from $\phi = 0$ to $\phi = \pi$, relying only on the learned dynamics.

Crazyflie quadrotors

Finally, we verified our energy-based controller for under-actuated systems in Section 5.2.2 by driving the drone to track a pre-defined trajectory. We are given the desired position $\mathbf{p}^*$ and the desired heading $\boldsymbol{\psi}^*$ by the trajectory and construct an appropriate choice of $\mathbf{R}^*$, $\mathfrak{p}^*$ to be used with the energy-based controller in (5.24). The desired momenta are constructed as follows:

$$\begin{aligned} \mathfrak{p}^* &= \mathbf{M}\begin{bmatrix} \mathbf{R}^{\top}\dot{\mathbf{p}}^* \\ \mathbf{R}^{\top}\mathbf{R}^*\boldsymbol{\omega}^* \end{bmatrix} = \mathbf{M}\begin{bmatrix} \mathbf{R}^{\top}\mathbf{R}^*\mathbf{v}^* \\ \mathbf{R}^{\top}\mathbf{R}^*\boldsymbol{\omega}^* \end{bmatrix}, \\ \dot{\mathfrak{p}}^* &= \mathbf{M}\begin{bmatrix} \mathbf{R}^{\top}\ddot{\mathbf{p}}^* - \hat{\boldsymbol{\omega}}\mathbf{R}^{\top}\dot{\mathbf{p}}^* \\ \mathbf{R}^{\top}\mathbf{R}^*\dot{\boldsymbol{\omega}}^* - \hat{\boldsymbol{\omega}}\mathbf{R}^{\top}\mathbf{R}^*\boldsymbol{\omega}^* \end{bmatrix}. \end{aligned} \tag{5.39}$$

The control input (5.24) becomes:

$$\begin{aligned} \mathbf{u} \;=\; \mathbf{g}^{\dagger}(\mathbf{q}) \Bigg(& \mathbf{q}^{\times\top}\frac{\partial V(\mathbf{q})}{\partial \mathbf{q}} - \mathfrak{p}^{\times}\mathbf{M}^{-1}\mathfrak{p} - \mathbf{e}(\mathbf{q}, \mathbf{q}^*) \\ & - \mathbf{K_d}\mathbf{M}^{-1}(\mathfrak{p} - \mathfrak{p}^*) + \dot{\mathfrak{p}}^* \Bigg). \end{aligned} \tag{5.40}$$

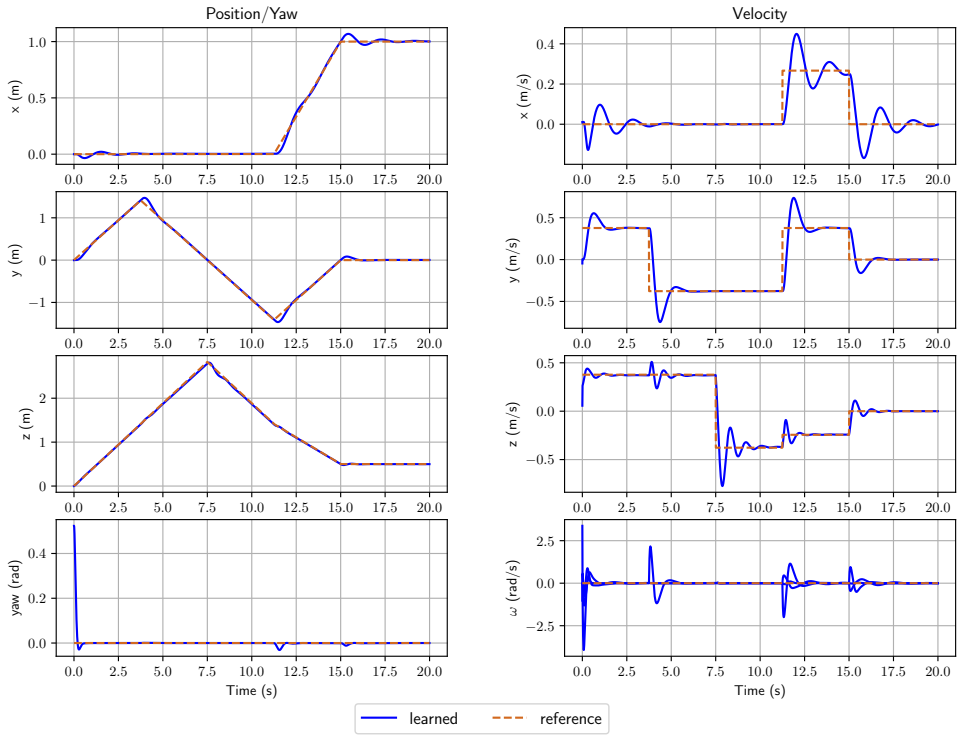


Figure 5.5. Crazyflie quadrotor trajectory (blue) tracking a desired diamond-shaped trajectory (orange) shown in Figure 5.6.

By expanding the terms in (5.40), we have:

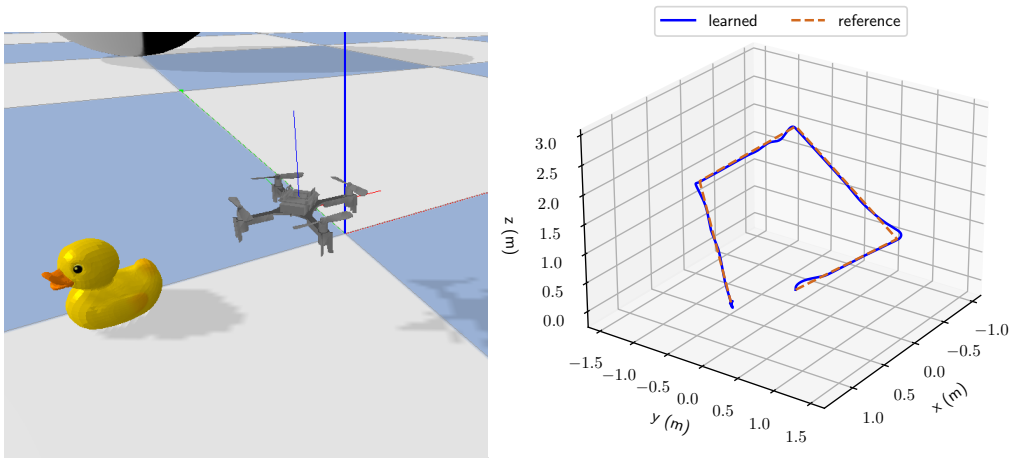
$$\mathfrak{p}^{\times}\mathbf{M}^{-1}\mathfrak{p} \;=\; \mathfrak{p}^{\times}\boldsymbol{\zeta} = \begin{bmatrix} \hat{\mathbf{p}}_{\mathbf{v}}\boldsymbol{\omega} \\ \hat{\mathbf{p}}_{\boldsymbol{\omega}}\boldsymbol{\omega} + \hat{\mathbf{p}}_{\mathbf{v}}\mathbf{v} \end{bmatrix}, \tag{5.41}$$

$$\mathbf{M}^{-1}(\mathfrak{p} - \mathfrak{p}^{*}) \;=\; \begin{bmatrix} \mathbf{v} - \mathbf{R}^{\top}\dot{\mathbf{p}}^{*} \\ \boldsymbol{\omega} - \mathbf{R}^{\top}\mathbf{R}^{*}\boldsymbol{\omega}^{*} \end{bmatrix}, \tag{5.42}$$

$$\mathbf{q}^{\times\top}\frac{\partial V(\mathbf{q})}{\partial \mathbf{q}} \;=\; \begin{bmatrix} \mathbf{R}^{\top}\frac{\partial V(\mathbf{q})}{\partial \mathbf{p}} \\ \sum_{i=1}^{3}\hat{\mathbf{r}}_i\frac{\partial V(\mathbf{q})}{\partial \mathbf{r}_i} \end{bmatrix}. \tag{5.43}$$

Choosing the control gain $\mathbf{K_d}$ of the form $\mathbf{K_d} = \begin{bmatrix} \mathbf{K}_v & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_{\omega} \end{bmatrix}$, the control input can be written explicitly as

$$\mathbf{u} = \mathbf{g}^{\dagger}(\mathbf{q}) \begin{bmatrix} \mathbf{b}_{\mathbf{v}} \\ \mathbf{b}_{\boldsymbol{\omega}} \end{bmatrix}, \tag{5.44}$$



(a) Crazyflie simulator **(b)** Trajectory tracking

Figure 5.6. Trajectory tracking experiment with a Crazyflie quadrotor in the PyBullet simulator [131].

where

$$\begin{aligned}\mathbf{b}_{\mathbf{v}} &= \mathbf{R}^\top \frac{\partial V(\mathbf{q})}{\partial \mathbf{p}} - \hat{\mathbf{p}}_{\mathbf{v}}\boldsymbol{\omega} - \mathbf{R}^\top \mathbf{K}_{\mathbf{p}}(\mathbf{p}-\mathbf{p}^*) \\ &\quad -\mathbf{K}_{\mathbf{v}}(\mathbf{v}-\mathbf{R}^\top\dot{\mathbf{p}}^*) + \mathbf{M}_1(\mathbf{R}^\top\ddot{\mathbf{p}}^* - \hat{\boldsymbol{\omega}}\mathbf{R}^\top\dot{\mathbf{p}}^*), \qquad (5.45)\\ \mathbf{b}_{\boldsymbol{\omega}} &= \sum_{i=1}^{3} \hat{\mathbf{r}}_i \frac{\partial V(\mathbf{q})}{\partial \mathbf{r}_i} - \mathbf{K}_{\boldsymbol{\omega}}(\boldsymbol{\omega} - \mathbf{R}^\top\mathbf{R}^*\boldsymbol{\omega}^*) \\ &\quad -(\hat{\mathbf{p}}_{\boldsymbol{\omega}}\boldsymbol{\omega} + \hat{\mathbf{p}}_{\mathbf{v}}\mathbf{v}) - \frac{1}{2}\left(\mathbf{K}_{\mathbf{R}}\mathbf{R}^{*\top}\mathbf{R} - \mathbf{R}^\top\mathbf{R}^*\mathbf{K}_{\mathbf{R}}^\top\right)^\vee \\ &\quad +\mathbf{M}_2(\mathbf{R}^\top\mathbf{R}^*\dot{\boldsymbol{\omega}}^* - \hat{\boldsymbol{\omega}}\mathbf{R}^\top\mathbf{R}^*\boldsymbol{\omega}^*). \qquad (5.46)\end{aligned}$$

Note that $\mathbf{b}_{\mathbf{v}} \in \mathbb{R}^3$ is the desired thrust in the body frame that depend only on the desired position $\mathbf{p}^*$ and the current pose. It is transformed to the world frame as $\mathbf{R}\mathbf{b}_{\mathbf{v}}$, representing the thrust in the world frame. Inspired by [90], the vector $\mathbf{R}\mathbf{b}_{\mathbf{v}}$ should be the z axis of the body frame, i.e., the third column $\mathbf{b}_3^*$ of the desired rotation matrix $\mathbf{R}^*$. The second column $\mathbf{b}_2^*$ of the desired rotation matrix $\mathbf{R}^*$ can be chosen so that it has the desired yaw angle ψ^* and is perpendicular to $\mathbf{b}_3^*$. This can be done by projecting the second column of the yaw's rotation matrix $\mathbf{b}_2^\psi = [-\sin\psi, \cos\psi, 0]$ onto the plane perpendicular to $\mathbf{b}_3^*$. We have $\mathbf{R}^* = [\mathbf{b}_1^* \quad \mathbf{b}_2^* \quad \mathbf{b}_3^*]$ where:

$$\mathbf{b}_3^* = \frac{\mathbf{R}\mathbf{b}_{\mathbf{v}}}{\|\mathbf{R}\mathbf{b}_{\mathbf{v}}\|}, \mathbf{b}_1^* = \frac{\mathbf{b}_2^\psi \times \mathbf{b}_3^*}{\|\mathbf{b}_2^\psi \times \mathbf{b}_3^*\|}, \mathbf{b}_2^* = \mathbf{b}_3^* \times \mathbf{b}_1^*, \qquad (5.47)$$

and $\hat{\boldsymbol{\omega}}^* = \mathbf{R}^{*\top}\dot{\mathbf{R}}^*$. The derivative $\dot{\mathbf{R}}^*$ is calculated as follows.

$$\dot{\mathbf{b}}_3^* = \mathbf{b}_3^* \times \frac{\mathbf{R}\dot{\mathbf{b}}_\mathbf{v}}{\|\mathbf{R}\mathbf{b}_\mathbf{v}\|} \times \mathbf{b}_3^*, \tag{5.48}$$

$$\dot{\mathbf{b}}_1^* = \mathbf{b}_1^* \times \frac{\dot{\mathbf{b}}_2^\psi \times \mathbf{b}_3^* + \mathbf{b}_2^\psi \times \dot{\mathbf{b}}_3^*}{\|\mathbf{b}_2^\psi \times \mathbf{b}_3^*\|} \times \mathbf{b}_1^*, \tag{5.49}$$

$$\dot{\mathbf{b}}_2^* = \dot{\mathbf{b}}_3^* \times \mathbf{b}_1^* + \mathbf{b}_3^* \times \dot{\mathbf{b}}_1^*. \tag{5.50}$$

By plugging $\mathbf{R}^*$ and $\boldsymbol{\omega}^*$ back in $\mathbf{b}_{\boldsymbol{\omega}}$, we obtain the complete control input $\mathbf{u}$ in (5.44).

Figure 5.6b qualitatively shows that the drone controlled by our energy-based controller successfully finished the task. Since the learned generalized mass $\mathbf{M}_1$ and inertia $\mathbf{M}_2$ converged to constant diagonal matrices, the control gains were chosen as follows in our experiments: $\mathbf{K_p} = \text{diag}([0.8, 0.8, 3.9])$, $\mathbf{K_v} = 0.23\mathbf{I}$, $\mathbf{K_R} = \text{diag}([3.6, 3.6, 6.9])$, $\mathbf{K}_{\boldsymbol{\omega}} = \text{diag}([0.3, 0.3, 0.6])$. Figure 4.4 quantitatively plots the tracking errors for position, yaw angles, linear and angular velocity. Our controller's computation time in Python was $2.5ms$ per control input, including forward passes of the learned neural networks, showing that it is suitable for fast real-time applications.

5.4.3 Effectiveness of Adaptive Control Design

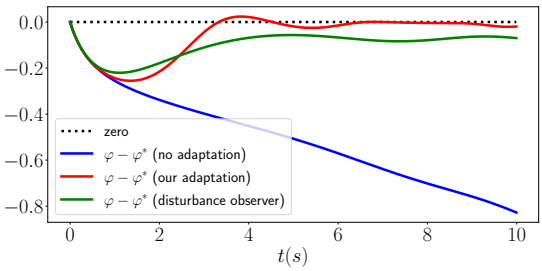
We evaluate our data-driven geometric adaptive controller on a fully-actuated pendulum and an under-actuated quadrotor.

Pendulum

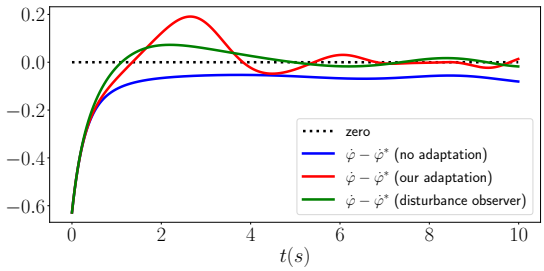
Consider a pendulum with angle φ, scalar control input u, and dynamics function:

$$m\ddot{\varphi} = -5\sin\varphi + u + d,$$

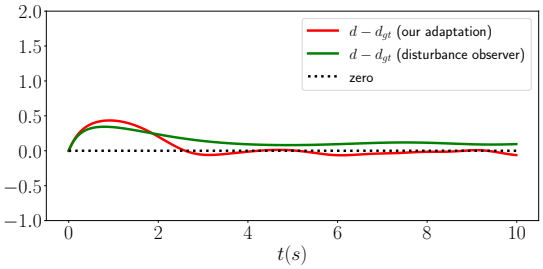
where the mass is $m = 1/3$, the potential energy is $V(\varphi) = 5(1 - \cos\varphi)$, the input gain is $g(\varphi) = 1$, and the disturbance $d = -\mu\dot{\varphi}$ models a friction force with unknown friction coefficient μ. To illustrate our geometric adaptive control approach, we consider φ as a yaw angle specifying a rotation $\mathbf{R}$ around the z axis, with angular velocity $\boldsymbol{\omega} = [0, 0, \dot{\varphi}]$. We remove the position $\mathbf{p}$ and linear velocity $\mathbf{v}$ terms from Hamilton's equations in (2.40) to obtain the pendulum dynamics.



(a) The angle error $\varphi - \varphi^*$.



(b) Velocity error $\dot{\varphi} - \dot{\varphi^*}$.



(c) The disturbance error $d - d_{gt}$ with the ground-truth disturbance $d_{gt} = -2.5\dot{\varphi}$.

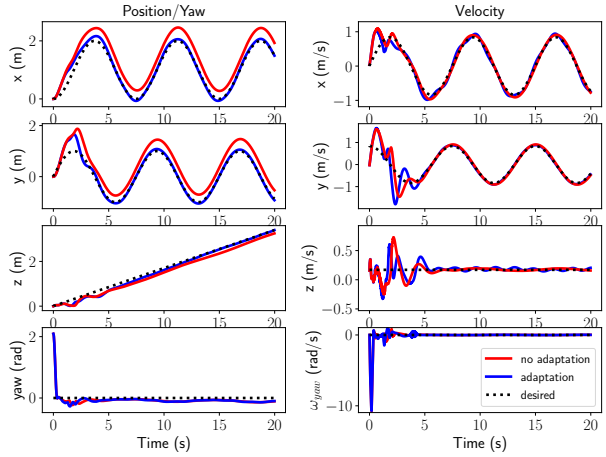
Figure 5.7. Comparison of our learned adaptive controller and a disturbance observer method on a pendulum.

Table 5.2. Angle tracking performance of a pendulum with our adaptive controller, with disturbance observer (DOB), and without adaptation.

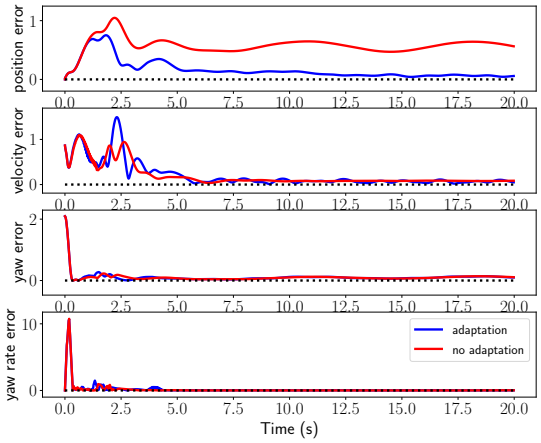
Approach	No adaptation	Our adaptation	DOB
Angle error/time step	0.35 ± 0.14	$\mathbf{0.04 \pm 0.02}$	0.08 ± 0.02
Disturbance error/time step	0.67 ± 0.27	$\mathbf{0.06 \pm 0.03}$	0.10 ± 0.04

To learn the disturbance features, we consider $M = 11$ realizations of the disturbance $d_j = -\mu_j \dot{\varphi}$ with friction coefficient $\mu_j = 0.05(j-1) \in [0, 0.5]$ for $j = 1, \ldots, M$. For each value μ_j, we collect transitions $\mathcal{D}_j = \{\mathbf{x}_0^{(ij)}, \mathbf{u}^{(ij)}, \mathbf{x}_f^{(ij)}, \tau^{(ij)}\}_{i=1}^{1024}$ by applying 1024 random control inputs to the pendulum for a time interval of $\tau^{(ij)} = 0.01$ s. We train the disturbance model (Section 4.4) for 4000 iterations with learning rate 10^{-4}.

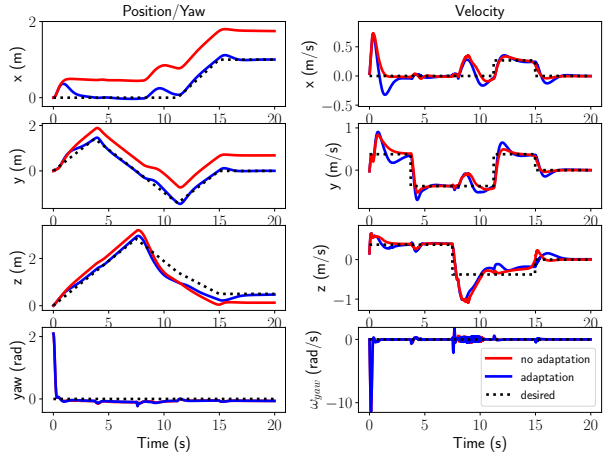
We verify our adaptive controller $(\boldsymbol{\pi}, \boldsymbol{\rho})$ in Section 4.4 with the task of tracking a desired angle $\varphi^*(t) = \pi t/5 + \pi t^2/50$. We simplify the controller $\boldsymbol{\pi}$ in (5.32) and the adaptation law $\boldsymbol{\rho}$ in (5.35) by removing the position and linear velocity components. The controller gains are: $k_{\mathbf{R}} = 1$, $k_{\mathbf{d}} = 2$, $c_{\mathbf{R}} = 75$, $c_{\boldsymbol{\omega}} = 10$. We compare our approach with a disturbance observer method [22, 94] for the pendulum. As the disturbance features in (4.13) are unknown, we design an observer to estimate d online. Let z be the observer state with dynamics $m\dot{z} =$



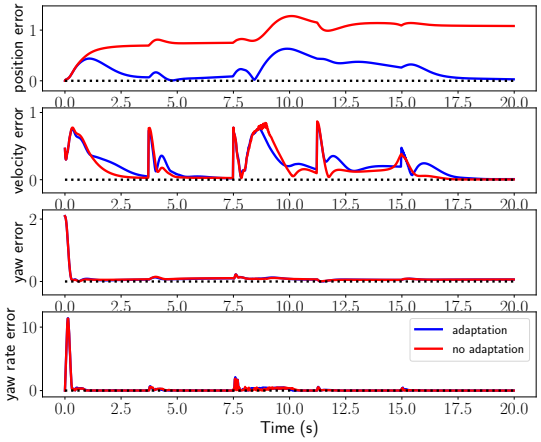
(a) Robot states (scenario 1).



(b) Tracking errors (scenario 1).



(c) Robot states (scenario 2).

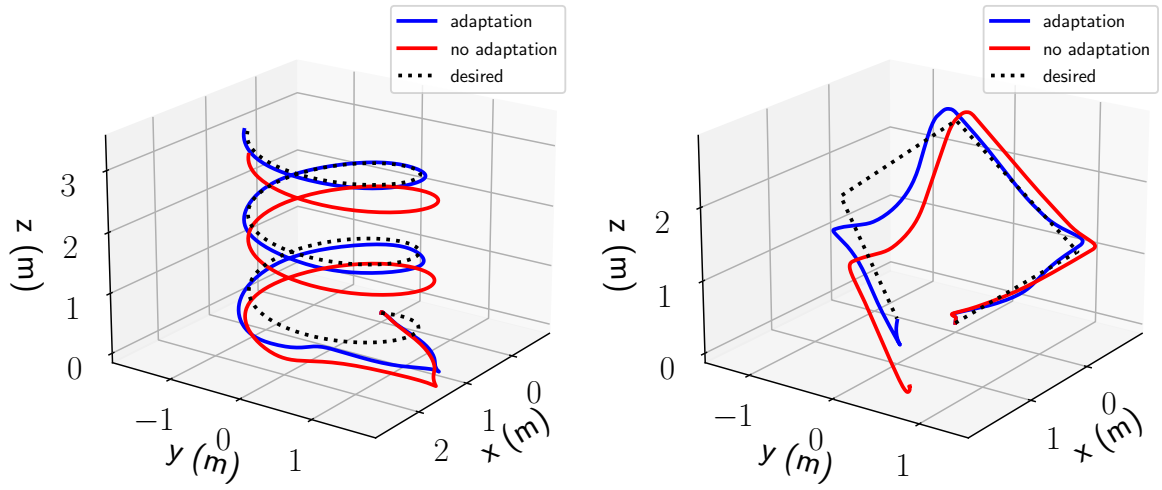


(d) Tracking errors (scenario 2).

Figure 5.8. Tracking performance under an external wind $\mathbf{d}_w = [0.075\ 0.075\ 0]$ and two defective rotors from the beginning (scenario 1) and after 8s (scenario 2) both with $(\delta_1, \delta_2) = (80\%, 80\%)$.

$-l(\dot{\varphi})z - l(\dot{\varphi})\left(r(\dot{\varphi}) - 5\sin(\varphi) + u\right)$, where $l(\dot{\varphi}) = \frac{\partial r(\dot{\varphi})}{\partial \dot{\varphi}}$ for some function $r(\dot{\varphi})$. The disturbance d is estimated as $\hat{d} = z + r(\dot{\varphi})$. The disturbance estimation error is $e_d = \hat{d} - d$ satisfying $\dot{e}_d = \dot{z} + \frac{\partial r(\dot{\varphi})}{\partial \dot{\varphi}}\ddot{\varphi} = -l(\dot{\varphi})e_d/m$. We choose $r(\dot{\varphi}) = \dot{\varphi}$ so that the disturbance estimation errors converges to 0 asymptotically. While it is hard to provide a fair comparison between controllers, e.g. different control gain tuning, we try our best to match the experiment settings. For example, we use the same tracking controller $\boldsymbol{\pi}$ in (5.32) to compensate the estimated disturbance.

We run the experiments 100 times with a friction coefficient μ uniformly sampled from the range $[0.5, 3]$. Table 5.2 shows the angle tracking errors and the disturbance estimation errors



(a) Scenario 1. **(b)** Scenario 2.

Figure 5.9. Tracking visualization with and without our adaptation law.

Table 5.3. Tracking error of a quadrotor with and without our adaptation.

Experiments	Diamond-shaped	Spiral
Scenario 1 (without adaptation)	0.71 ± 0.15 (m)	0.55 ± 0.13 (m)
Scenario 1 (with adaptation)	$\mathbf{0.12 \pm 0.02}$ (m)	$\mathbf{0.13 \pm 0.01}$ (m)
Scenario 2 (without adaptation)	0.62 ± 0.13 (m)	0.64 ± 0.10 (m)
Scenario 2 (with adaptation)	$\mathbf{0.12 \pm 0.02}$ (m)	$\mathbf{0.16 \pm 0.02}$ (m)

with our adaptive controller, with the disturbance observer (DOB), and without adaptation. Our adaptive controller achieves better tracking error and disturbance estimation error than the DOB approach. Figure 5.7 plots the tracking errors and disturbance estimation error with $\mu = 2.5$, showing that we achieve the desired angle $\varphi^*(t)$ and are able to converge to the state-dependent ground-truth disturbance $d_{gt} = -2.5\dot{\varphi}$. Without knowing the disturbance features, the DOB method lags behind the changes in ground-truth disturbances caused by the velocity $\dot{\varphi}$. This illustrates the benefit of our approach – the learned disturbance features improve the performance of the adaptive controller.

Crazyflie Quadrotor

Next, we consider a Crazyflie quadrotor, simulated using the PyBullet physics engine [131], with control input $\mathbf{u} = [f, \boldsymbol{\tau}]$ including the thrust $f \in \mathbb{R}_{\geq 0}$ and torque $\boldsymbol{\tau} \in \mathbb{R}^3$ generated by the 4 rotors. The mass of the quadrotor is $m = 0.027$ kg and the inertia matrix is

$\mathbf{J} = 10^{-5}\,\mathrm{diag}([1.4, 1.4, 2.2])$, leading to the generalized mass matrix $\mathbf{M}(\mathbf{q}) = \mathrm{diag}(m\mathbf{I}, \mathbf{J})$. The potential energy is $V(\mathbf{q}) = mg \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \mathbf{p}$ with $g \approx 9.8\, ms^{-2}$, where $\mathbf{p}$ is the position of the quadrotor. We consider disturbances from three sources: 1) horizontal wind, simulated as an external force $\mathbf{d}_w = \begin{bmatrix} w_x & w_y & 0 \end{bmatrix}^\top \in \mathbb{R}^3$ in the world frame, i.e., $\mathbf{R}^\top \mathbf{d}_w$ in the body frame; 2) two defective rotors 1 and 2, generating δ_1 and δ_2 percents of the nominal thrust, respectively; and 3) near-ground, drag, and downwash effects in the PyBullet simulated quadrotor.

As described in Section 4.4, we learn the disturbance features $\mathbf{W}_{\boldsymbol{\theta}}(\mathbf{q}, \mathbf{p})$ from a dataset $\mathcal{D}$ of transitions using a Hamiltonian-based neural ODE network. We collect a dataset $\mathcal{D} = \{\mathcal{D}_j\}_{j=1}^{M}$ with $M = 8$ realizations of the disturbances $\mathbf{d}_{wj}$, δ_{1j}, and δ_{2j}. Specifically, the wind components w_{xj}, w_{yj} are chosen from the set $\{\pm 0.025, \pm 0.05\}$, while the values of δ_{1j} and δ_{2j} are sampled from the range $[94\%, 98\%]$. For each disturbance realization, a PID controller provided by [131] is used to drive the quadrotor from a random starting point to 9 different desired poses, providing transitions $\mathcal{D}_j = \{\mathbf{x}_0^{(ij)}, \mathbf{u}^{(ij)}, \mathbf{x}_f^{(ij)}, \tau^{(ij)}\}_{i=1}^{1080}$ with $\tau^{(ij)} = 1/240$ s.

We verify our geometric adaptive controller with learned disturbance features by having the quadrotor track pre-defined trajectories in the presence of the aforementioned disturbances. The desired trajectory is specified by the desired position $\mathbf{p}^*(t)$ and the desired heading $\boldsymbol{\psi}^*(t)$. We construct an appropriate choice of $\mathbf{R}^*$ and $\boldsymbol{\omega}^*$ from $\boldsymbol{\psi}^*(t)$, as described in [37, 53], to be used with the adaptive controller. The tracking controller in (5.32) with gains $k_{\mathbf{p}} = 0.135, k_{\mathbf{v}} = 0.0675, k_{\mathbf{R}} = 1.0$, and $k_{\boldsymbol{\omega}} = 0.08$, is used to obtain the control input $\mathbf{u}$ that compensates for the disturbances. The disturbances $\mathbf{d}$ are estimated by updating the weights $\mathbf{a}$ using the adaptation law (5.35) with gains $c_{\mathbf{p}} = c_{\mathbf{R}} = 0.08, c_{\mathbf{v}} = c_{\boldsymbol{\omega}} = 0.04$.

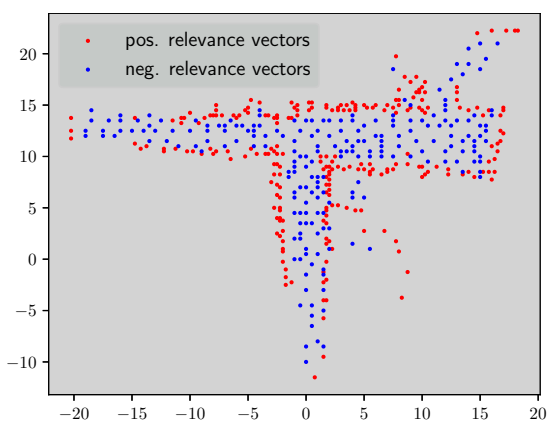
We test the controller with wind $\mathbf{d}_w$, rotors 1 and 2 that become defective from the beginning (scenario 1) or during flight at $t = 8$ s (scenario 2), and near-ground, drag, and downwash effects enabled in PyBullet. We track diamond-shaped and spiral trajectories 100 times with w_x and w_y uniformly sampled from $[0, 0.075]$ and δ_1 and δ_2 drawn uniformly from $[80\%, 99\%]$. Table 5.3 shows the mean and standard deviation of the tracking errors with and without adaptation from the 100 flights. The errors with adaptation are ~ 5 times lower than without adaptation, illustrating the benefit of our adaptive control design. For $\mathbf{d}_w = \begin{bmatrix} 0.075 & 0.075 & 0 \end{bmatrix}$ and $(\delta_1, \delta_2) = (80\%, 80\%)$, the quadrotor in scenario 1 without adaptation

drifts while our adaptive controller estimates the disturbances online after a few seconds and successfully tracks the trajectory as seen in Figure 5.8a, 5.8b and 5.9 (left). For the same wind, the quadrotor in scenario 2 with our controller starts to track the trajectory, then drops down at $t = 8$ s, due to the rotors becoming defective, but recovers as our adaptation law updates the disturbances accordingly, as seen in Figure 5.8d and 5.9 (right). The velocity error spikes in Figure 5.8d are caused by sharp turns in the diamond-shaped trajectory and the defective rotors at $t = 8$ s. Without adaptation, the quadrotor drops to the ground at $t \approx 12.5$ s, shown in Figure 5.9 (right). In Figure 5.8 and 5.9, the tracking errors with adaptation stabilize close to but not exactly 0 because of the model error between $\mathbf{W}_{\boldsymbol{\theta}}(\mathbf{q}, \dot{\mathbf{p}})$ and $\mathbf{W}(\mathbf{q}, \dot{\mathbf{p}})$, and the control input discretization in time.

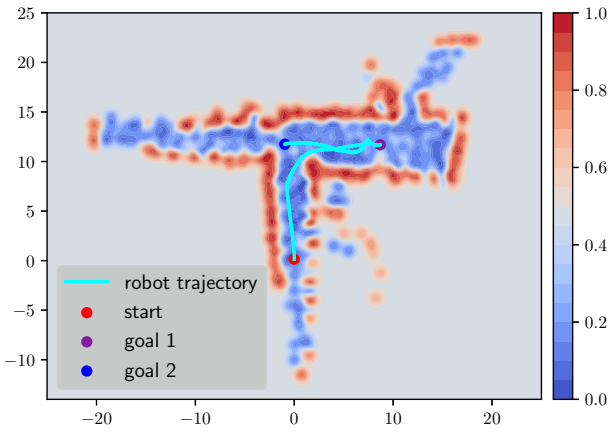
5.4.4 Real Experiments with Ground Robots

Real experiments were carried out on an 1/10th scale Racecar robot equipped with a Hokuyo UST-10LX Lidar and Nvidia TX2 computer. The robot body was modeled by a ball of radius $r = 0.25\ m$. The online training data (Section 3.3.2) were generated from a grid with resolution $0.25\ m$. We used an RBF kernel parameter $\Gamma = \sqrt{\gamma}\mathbf{I}$ with $\gamma = 3.0$ and an R^*-tree approximation of the score $F(\mathbf{x})$ with $K^+ + K^- = 20$ nearest support vectors around the robot location $\mathbf{p}_k$ for map updating. For motion planning, second-order polynomial motion primitives were generated with time discretization of $\tau = 1\ s$ as described in Section 5.3. The motion cost was defined as $c(\mathbf{s}, \mathbf{a}) := (\|\mathbf{a}\|^2 + 2)\tau$ to encourage both smooth and fast motion [100]. Algorithm 4 with Eq. (5.7), $\varepsilon = 0.1$, score approximation with $K^+ = K^- = 2$, and threshold $e = -0.01$ was used for collision checking in Algorithm 5. The trajectory generated by an A^* motion planner was tracked using a closed-loop controller [4]. The robot navigated in an unknown hallway to two destinations consequently chosen by a human operator. Figure 5.10a shows the learned relevance vectors representing the environment. Figure 5.10b shows the probabilistic map recovered from the relevance vectors together with the robot trajectory and the two chosen destinations.

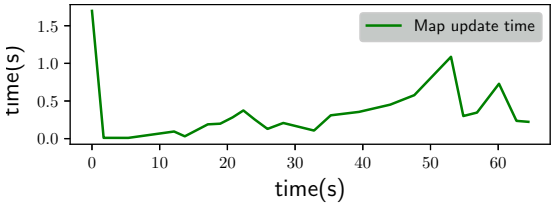
The time taken by Algorithm 2 to update the relevance vectors from one lidar scan and the A^* replanning time per motion primitive are shown in Figure 5.10c and 5.10d. Map updates implemented in Python took 0.4 s on average. It took a longer time ($\sim 1\ s$) to update the



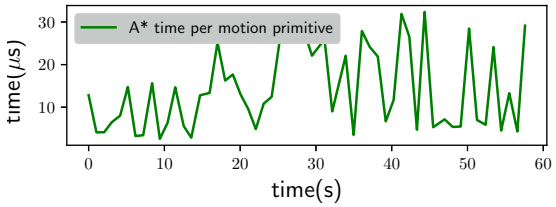
(a) The final relevance vectors.



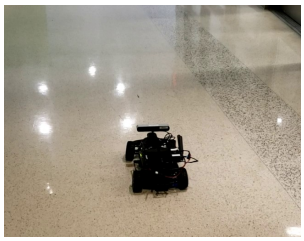
(b) The final probabilistic map.



(c) Map update time.

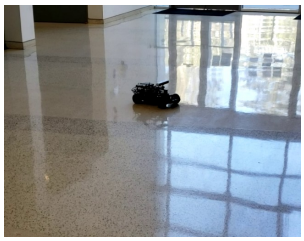


(d) Planning time per motion primitive.



(e) At $t = 0s$.

(f) At $t = 23s$.



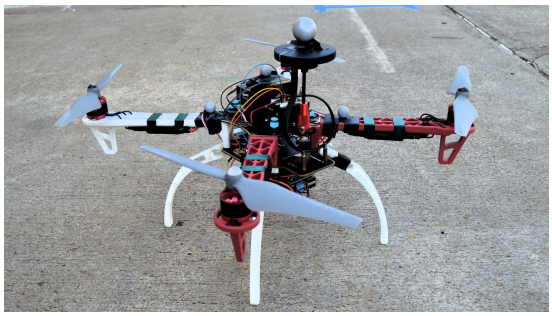
(g) At $t = 44s$.

(h) At $t = 60s$.

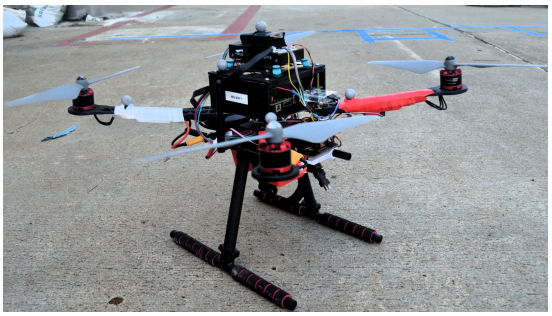
Figure 5.10. Real experiment with an autonomous Racecar robot navigating in an unknown hallway environment.

map when the robot observed new large parts of the environment, e.g., at the beginning and toward the end of our experiment. To evaluate collision checking time, the A^* replanning time was normalized by the number of motion primitives being checked to account for differences in planning to nearby and far goals. The planning time per motion primitive was $\sim$ 15 μs on average and $\sim$ 30 μs at most, suggesting our collision checking algorithms' suitability for real-time applications.

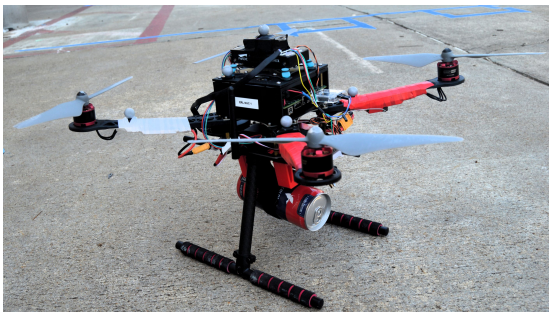
In both the simulations and the real experiment, the free area contains multiple blobs of points with low occupancy probability, caused by the sparse map representation and the fixed kernel parameters of SBKM. To improve this, kernel parameter learning, e.g. using variational



(a) Our RaspberryPi drone.



(b) Our Intel NUC drone.



(c) Our Intel NUC drone with a payload.

Figure 5.11. Our customized quadrotors with different frames, computers, sensors, and payload.

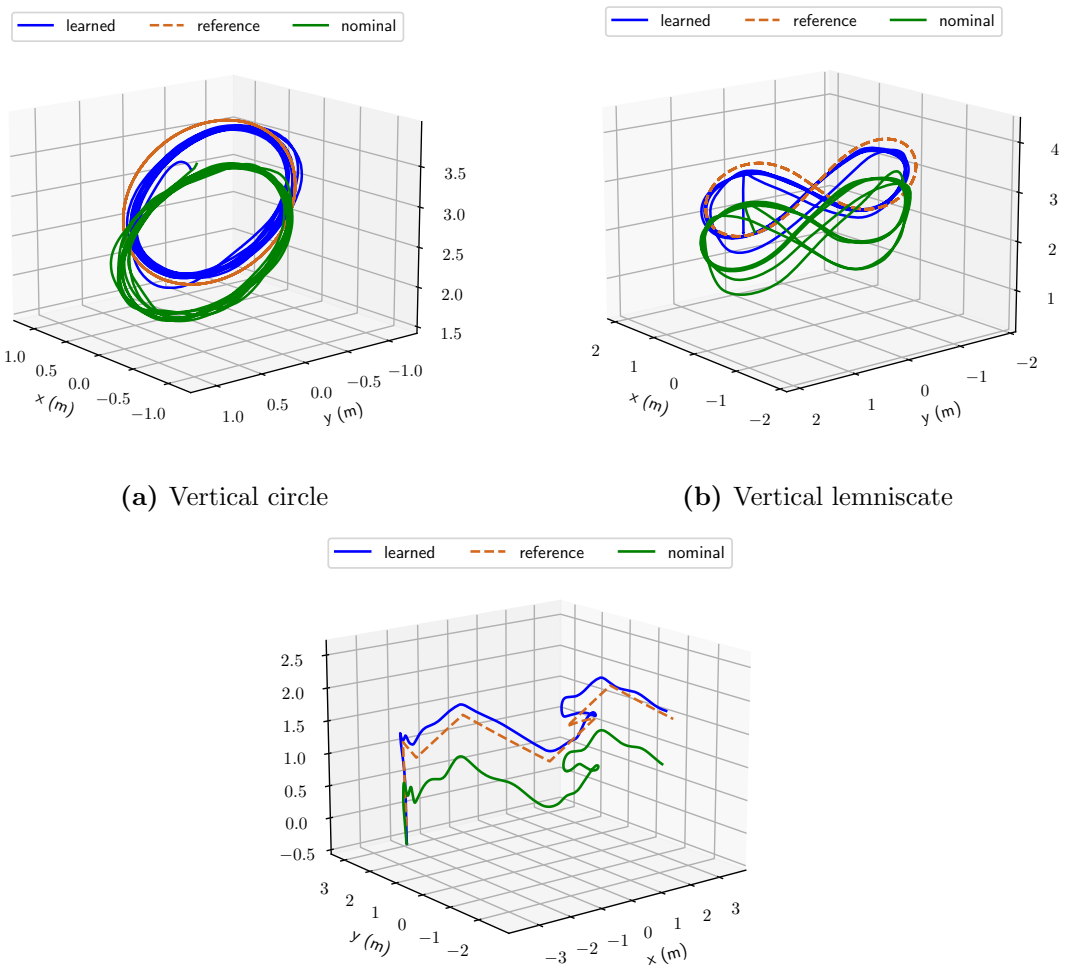
inference [151], clustering and automatic relevance determination [58, 59], or kernel parameter dictionaries, e.g., pre-trained in small and simple environments [167], can be used to to adaptively update the kernel parameters at different location based on the depth measurements. This extension is a promising avenue for future research.

5.4.5 Real Experiments with Quadrotors

In this section, we verify our approach with a customized PX4 quadrotor shown in Figure 5.11b, equipped with an onboard i7 Intel NUC computer and a PX4 flight controller. The quadrotor's pose and twist are provided by a motion capture system.

Learning robot dynamics after quadrotor upgrade

We consider a scenario that our quadrotor is upgraded with a new frame and a new onboard computer, leading to changes in the robot dynamics that we aim to learn from data. The estimated mass and inertia matrix of a previous quadrotor model (Figure 5.11a with a RaspberryPi computer and an F450 frame) is used as the nominal mass $\mathbf{M}_{\mathbf{v}0} = 1.3\mathbf{I}$ and inertia



(a) Vertical circle

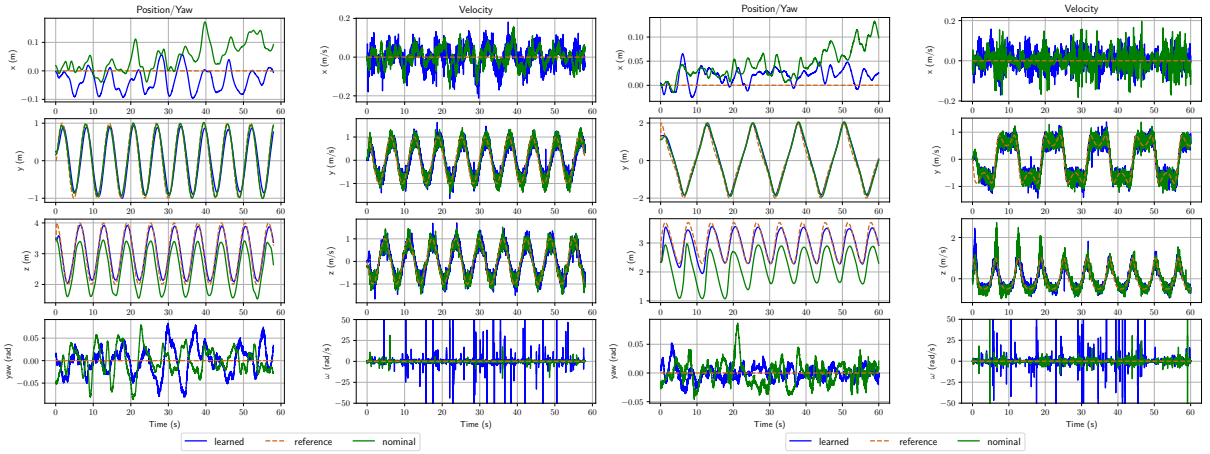
(b) Vertical lemniscate

(c) Piecewise linear

Figure 5.12. Trajectory tracking experiments with our real quadrotors and different trajectories using our learned model and controller.

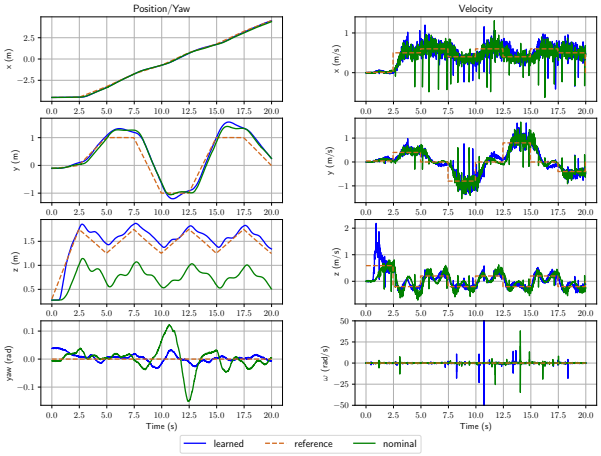
matrix $\mathbf{M}_{\boldsymbol{\omega}0} = \text{diag}([0.12, 0.12, 0.2])$ for the upgraded quadrotor (Figure 5.11b). The other nominal matrices are set to zero: $\mathbf{D}_{\mathbf{v}0}(\mathbf{q}, \mathbf{p}) = \mathbf{0}$, $\mathbf{D}_{\boldsymbol{\omega}0}(\mathbf{q}, \mathbf{p}) = \mathbf{0}$, $V_0(\mathbf{q}) = \mathbf{0}$ and $\mathbf{g}_0(\mathbf{q}) = \mathbf{0}$. We collect 12 state-control trajectories by modifying the PX4 firmware to expose the normalized thrust and normalized torque being sent to the motors and flying the quadrotor from a starting pose to 12 different poses using a PID controller provided by the PX4 flight controller. The trajectories were used to generate a dataset $\mathcal{D} = \{t^{(i)}_{0:N}, \mathbf{q}^{(i)}_{0:N}, \boldsymbol{\zeta}^{(i)}_{0:N}, \mathbf{u}^{(i)})\}_{i=1}^{D}$ with $N = 1$ and $D = 10000$. We train our model as described in Section 4.2.4 for 5000 steps.

The trained model is used with the control policy in Section 4.5.2 to track different

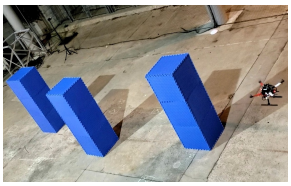


(a) Vertical circle trajectory

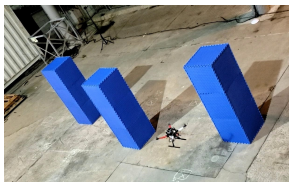
(b) Vertical lemniscate trajectory



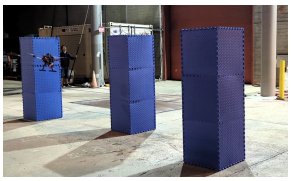
(c) Piecewise-linear trajectory



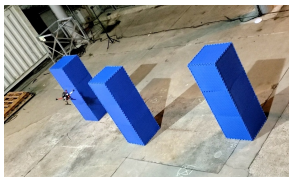
(d) Tracking at $t = 0$s



(e) Tracking at $t = 3$s



(f) Tracking at $t = 9$s



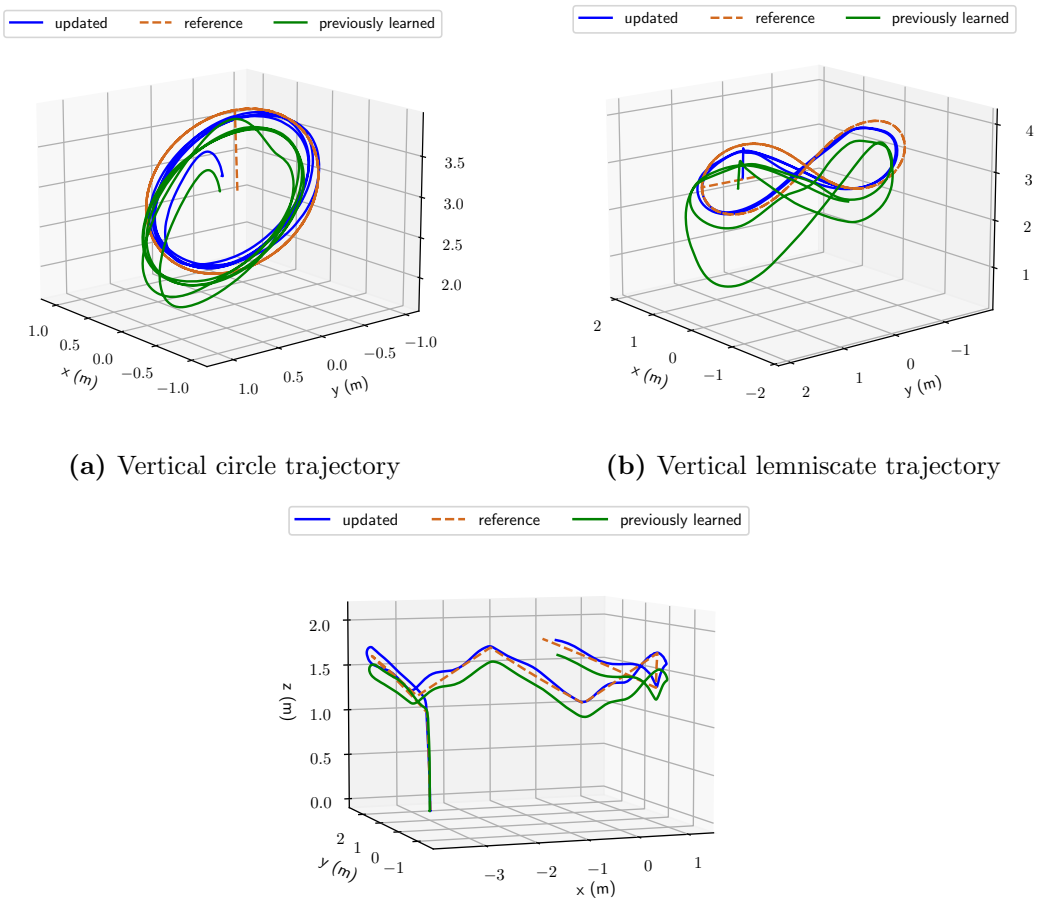
(g) Tracking at $t = 12$s

Figure 5.13. Tracking performance using our learned model and controller and using a nominal model and a geometric controller [90].

trajectories: a vertical circle, a vertical lemniscate, and a 3D piecewise-linear trajectories. Figure 5.12 and 5.13 show that we achieve better tracking performance using our learned dynamics and control design compared to the nominal model and the geometric controller in [90]. The tracking errors of our learned model improve by $2-4$ times compared to those of the nominal model, as shown in Table 5.4.

Learning robot dynamics with extra payload

In this section, we demonstrate that after the our dynamics model is trained, if there is a change in the dynamics, e.g. extra payload, we are able to update the dynamics quickly starting



(a) Vertical circle trajectory

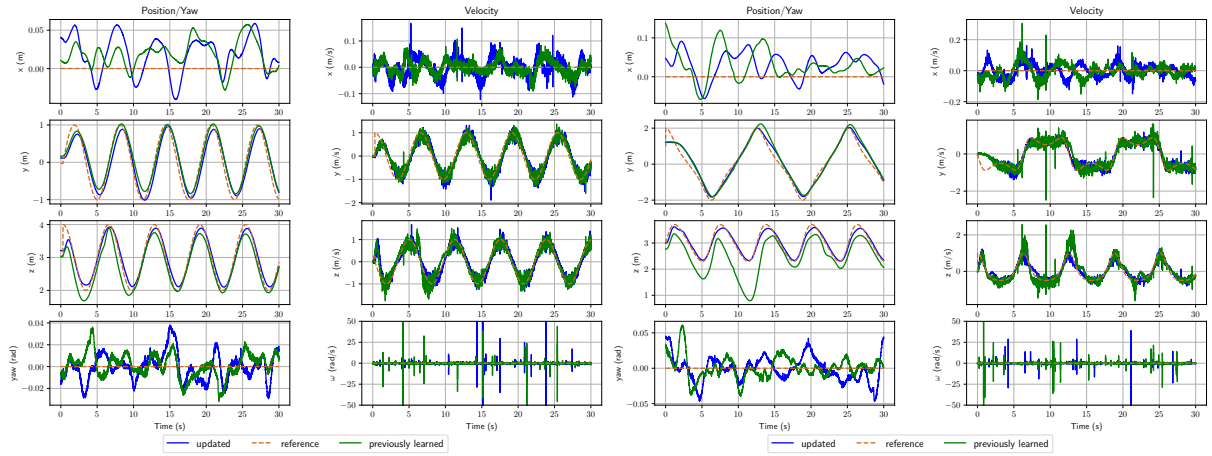
(b) Vertical lemniscate trajectory

(c) Piecewise linear trajectory

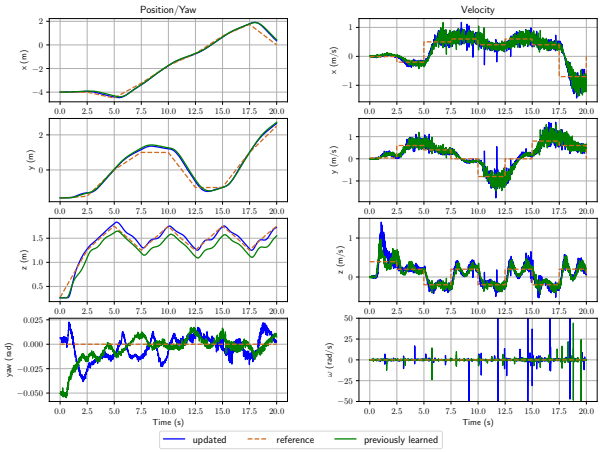
Figure 5.14. Trajectory tracking experiments with extra payload using our controller with previously learned model and updated model.

from the previously trained model. We attach a coffee can to the quadrotor frame (Figure 5.11c) to change the mass and inertia matrix of the robot. We then recollect the dataset also by driving the quadrotor to 12 different poses, and train our dynamics model only in 100 steps, initialized with the trained model in Section 5.4.5.

With the coffee can, the tracking performance of the controller with the previously learned model degrades as shown in Figure 5.14 and 5.15. Meanwhile, after a quick model update, we were able to track the trajectories accurately again. Table 5.4 shows that our updated model improves the tracking errors by $1.5 - 4$ times compared to the previous one.



(a) Vertical circle **(b)** Vertical lemniscate



(c) Piecewise-linear

Figure 5.15. Tracking performance with extra payload using our previously learned and updated models.

5.4.6 Active Mapping

Our SBKM representation enables uncertainty quantification which besides for collision checking can be used for active mapping. This ability is not offered by non-Bayesian mapping methods, such as SKM. In this section, we demonstrate active mapping of an unknown simulated environment using SBKM. Our approach estimates the map uncertainty in different regions and chooses the region with the highest uncertainty as the goal region. Specifically, we maintain a frontier, defined as a list of L candidate poses $\mathcal{P}_l$, $l = 1, 2, \ldots, L$. For each pose $\mathcal{P}_l$, we calculate the map uncertainty $H(\mathcal{S}_l)$ of the field of view $\mathcal{S}_l$ of the depth sensor. The map uncertainty of

Table 5.4. Position errors of our real quadrotor using our nominal and learned models with and without payload

Model	Train with payload	Test with payload	Circle	Lemniscate	Piecewise-linear
Nominal	-	No	0.26 (m)	0.52 (m)	0.62 (m)
Learned	No	No	**0.13(m)**	**0.14(m)**	**0.22(m)**
Learned	No	Yes	0.20 (m)	0.40 (m)	0.30 (m)
Learned	Yes	Yes	**0.13 (m)**	**0.12 (m)**	**0.21 (m)**

a region $\mathcal{S}$ is measured as the average marginal entropy over the region:

$$H(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \int_{\mathcal{S}} h(\mathbf{x}) d\mathbf{x}, \tag{5.51}$$

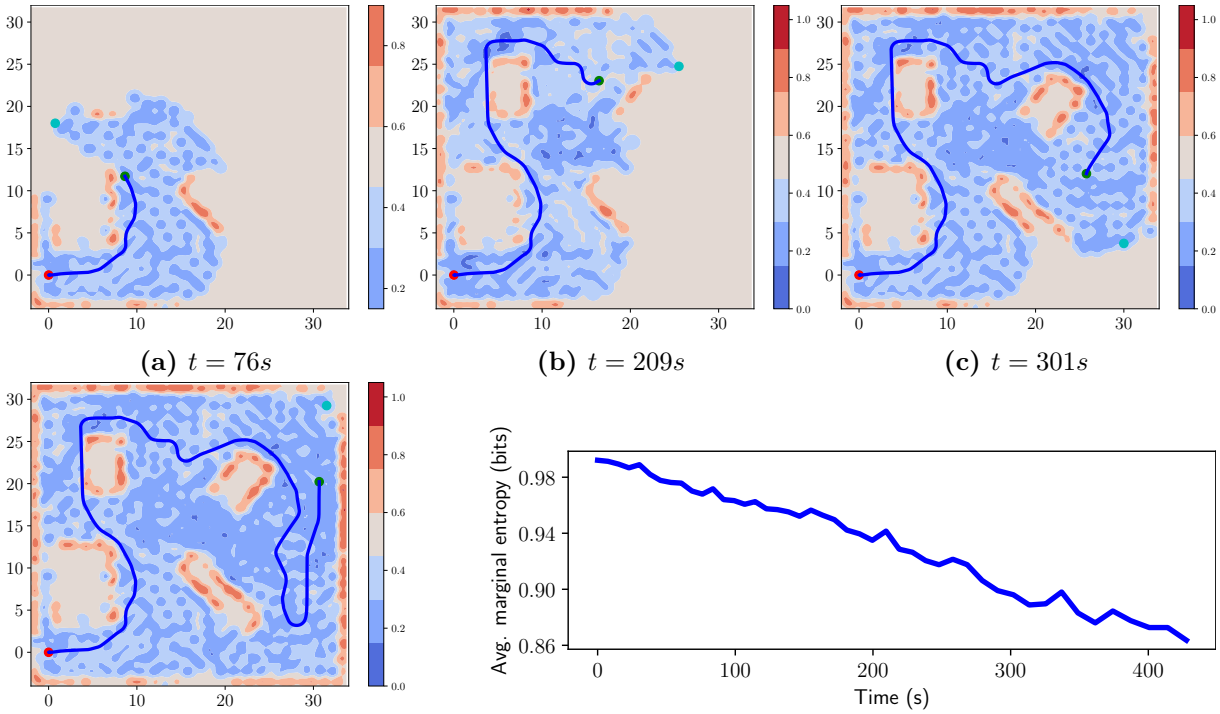
where $h(\mathbf{x})$ is the marginal entropy of a point $\mathbf{x}$ in the region, calculated using the predictive distribution in Def 10 as

$$\begin{aligned} h(\mathbf{x}) = \quad & -P(y=1|\mathbf{x},\boldsymbol{\xi}) \log_2 P(y=1|\mathbf{x},\boldsymbol{\xi}) \\ & -P(y=0|\mathbf{x},\boldsymbol{\xi}) \log_2 P(y=0|\mathbf{x}_i,\boldsymbol{\xi}), \end{aligned} \tag{5.52}$$

$y \in \{-1, 1\}$ is the predictive label of the point $\mathbf{x}$, and $|\mathcal{S}|$ denotes the area of the region $\mathcal{S}$. We choose the region S_{l^*} with the largest average marginal entropy to explore:

$$l^* = \underset{l=1,2,\ldots,L}{\operatorname{argmax}} H(\mathcal{S}_l). \tag{5.53}$$

In our active mapping experiment, the candidate poses $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_L$ in the frontier were sampled from the laser endpoints up to the current time t with 4 different yaw angles: $0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}$. Since the laser scans could not see through obstacles, we gained little information of the environment by placing the robot near the occupied regions. Therefore, only the endpoints with maximum lidar range, i.e. the laser ray did not hit an obstacle, and at least 2 m away from the positive relevance vectors were considered. A hypothetical lidar field of view $\mathcal{S}_l$ (similar to Figure 3.2b without the obstacles) simulating a Hokuyo UST-10LX lidar, was placed at each candidate pose $\mathcal{P}_l$. We sampled $N = 100$ points $\mathbf{x}_i, i = 1, \ldots, N$, from $\mathcal{S}_l$ and computed the



(a) $t = 76s$ **(b)** $t = 209s$ **(c)** $t = 301s$

(d) $t = 414s$ **(e)** Average marginal entropy of a point in the map over time.

Figure 5.16. Illustration of an active mapping task over time. The red, green and cyan dots are the initial and current robot positions, and the chosen goal.

marginal entropy $H(\mathbf{x}_i)$. The average marginal entropy (5.51) of the region $\mathcal{S}_l$ was approximated as

$$H(\mathcal{S}_l) \approx \frac{1}{N} \sum_{i=1}^{N} h(\mathbf{x}_i). \tag{5.54}$$

The robot picked the goal region $\mathcal{S}_{l^*}$ with the highest map uncertainty from the set $\{\mathcal{S}_i\}_{i=1}^{L}$ every 0.5 s and planned a trajectory to reach the goal using our collision checking methods and the same A^* planner used in Section 5.4.4. Figure 5.16 shows the SBKM map, the robot trajectory and the candidate pose associated with the goal region at different times as the robot successfully explored and actively built the map of the environment. The average marginal entropy $H(\mathcal{S}_l)$, estimated using Eq. (5.54) with $N = 20736$ points sampled on a regular grid of resolution 0.25 m, shows our active mapping approach reduced the map uncertainty over time.

5.5 Summary

This chapter presents a complete solution for autonomous navigation with our learned robot dynamics and sparse map representations. Given the current estimate of the map in Chapter 3, we derived efficient collision checking algorithms, that can be integrated into common motion planners, to generate a desired trajectory. We propose trajectory tracking controllers, with and without the presence of disturbances, for the learned Hamiltonian dynamics on Lie groups in Chapter 4. The effectiveness of our proposed approach is verified with both ground and aerial robot platforms.

Acknowledgments

Chapter 5, in part, is a reprint of the material as it appears in T. Duong, M. Yip, N. Atanasov, "*Autonomous Navigation in Unknown Environments with Sparse Bayesian Kernel-based Occupancy Mapping*", IEEE Transactions on Robotics (T-RO), vol. 38, no. 6, pp. 3694-3712, 2022, in T. Duong, N. Das, M. Yip, N. Atanasov, "*Autonomous Navigation in Unknown Environments using Sparse Kernel-based Occupancy Mapping*", International Conference on Robotics and Automation (ICRA), pp. 9666-9672, 2020, in T. Duong, N. Atanasov, "*Hamiltonian-based Neural ODE Networks on the SE(3) Manifold For Dynamics Learning and Control*", Robotics: Science and Systems (RSS), 2021, in T. Duong, N. Atanasov, "*Physics-guided Learning-based Adaptive Control on the SE(3) Manifold*", Physical Reasoning and Inductive Biases for the Real World Workshop, 2021, and in T. Duong, N. Atanasov, "*Adaptive Control of SE(3) Hamiltonian Dynamics with Learned Disturbance Features*", IEEE Control Systems Letters (L-CSS), vol. 6, pp. 2773-2778, 2022. Chapter 5, in part, has been submitted for publication of the material as it may appear in T. Duong, A. Altawaitan, J. Stanley, N. Atanasov, "*Port-Hamiltonian-based Neural ODE Networks on Lie Groups For Robot Dynamics Learning and Control*", Transactions on Robotics, 2024. The dissertation author was the primary investigator and author of these papers.

Chapter 6

Conclusions and Future Work

Autonomous navigation has undoubted potential to impact various applications such as surveillance and reconnaissance, search and rescue, transportation and agriculture. It depends on onboard sensors to perceive the world and build a model of the environment, which in turns, can be used for other downstream tasks such as path planning, safe control, localization and exploration. It also relies on the availability of an accurate dynamics model and control policy to safely and reliably follow a path to execute a task. Having an effective autonomous navigation framework that efficiently integrates environment representations, planning, dynamics models and control designs into one solution is a great technical challenge that we aim to address. As data generated from sensors such as Lidars, cameras, and inertial measurement units has recently become prevalent on robots, this dissertation tackles this challenge by developing novel machine learning techniques to learn large-scale, yet compact, representations of the environment and accurate robot dynamics, equipped with a trajectory tracking control design, efficiently from data for autonomous navigation purposes.

Chapter 2 provides the necessary background for the following chapters. It describes kernel perceptron and relevance vector machine (RVM) classifiers, which are used to develop our online mapping algorithms in Chapter 3. It provides definitions of kinematic constraints on matrix Lie groups, equations of motions for robot dynamics in Hamiltonian formulation and neural ODE networks, needed to design the proposed dynamics model in Chapter 4.

In Chapter 3, we propose sparse binary and probabilistic occupancy map representations that model the obstacle boundary as a decision boundary of a classifier, trained incrementally based on the depth measurements from onboard sensors. We develop online training algorithms

that choose a sparse set of critical data points, e.g. support vectors in our kernel perceptron-based maps and relevance vectors in our RVM-based maps, to represent the environments. We verify the benefit of our map sparseness in terms of storage requirements and update time compared to various baseline binary and probabilistic map representations. The map uncertainty from our probabilistic occupancy map representations can be used in other downstream tasks such as active mapping and exploration.

Chapter 4 presents our approach to learn accurate continuous-time robot dynamics from data while preserving the law of energy conservation and Lie group constraints via Hamiltonian formulation. The Hamiltonian-based model is trained to fit state-control trajectories using a neural ODE network. The approach is applied to learn dynamics of rigid-body robots such as pendulum, ground and aerial robots, whose state evolves on the Lie groups $SE(3)$. To handle dynamics changes and disturbances arising from new operational conditions, we learn a parametric disturbance model from data, which can be used to design an adaptive controller with better tracking performance. We verify that the proposed dynamics and disturbance model can be learned efficiently from just a few robot trajectories.

Chapter 5 combines the novel environment and dynamics representations in the previous chapters into a complete solution for autonomous navigation. Given a sparse occupancy map from Chapter 3, we develop efficient collision checking algorithms for line segments and general curves, representing potential robot trajectories. These algorithms are compatible with common planners, such as A^* or RRT^*, which return a desired path or trajectories for the robot to finish its navigation task. To be able to follow the trajectory, we develop an energy-based trajectory tracking controller based on the learned Hamiltonian dynamics. To handle dynamics changes described by our learned parametric disturbance model, we derive an adaptive controller that estimates and compensates for the disturbance online. Finally, we integrate our online sparse mapping, efficient collision checking and planning, accurate dynamics model and stable and adaptive control design in an autonomous navigation framework with demonstrations on ground and aerial robot platforms.

In summary, this dissertation offers an efficient autonomous navigation approach that builds scalable and compact map representations and learns accurate and data-efficient dynamics

models from data. Several promising potential directions, rising from the current results to improve long-term robot operations with more complicated tasks, are summarized below.

Contact dynamics. When robots, e.g. legged robots and manipulators, have contact with objects in the environment, their dynamics become non-smooth and thus, are challenging to learn. Imposing prior knowledge of the dynamics discontinuity on the dynamics model to improve its accuracy and data efficiency is a promising direction, e.g. via differentiable linear complementarity systems (LCSs) [134, 79]. The learned contact model is beneficial for improving control performance and simulating realistic robot interaction with the environment to reduce the sim-to-real gaps.

Efficient task and motion planning. The learned dynamics can be used to form accurate dynamics constraints in a trajectory optimization problem [161], leading to better trajectory for the robot to follow. Moreover, guidance from previous experiments or a generative AI model, e.g., based on semantic information, can be used to assist the task planner or to synthesize new robot skills. Semantic information and guidance from a language model can also be used to steer the task and motion planner towards the optimal solution. For example, Dai et al. [29] formulate guidance from a large language model as a heuristic function in a multi-heuristic A^* motion planner, while still providing optimality of the motion plan.

Learning with sensor models. Our approach can be extended to learn sensor models, e.g. biases in inertial measurement unit (IMU) [182, 101, 26, 17], that in turns, potentially improve state estimation, e.g. odometry, or (continuous-time) simultaneous localization and mapping (SLAM) problems. Another interesting direction is to learn robot dynamics and control by observing the environment, e.g., via Lidars [2] or cameras, where the features with data association detected from point clouds or images can be used to train the dynamics model directly using cycle consistency loss function [2].

Active and online learning. Instead of collecting data manually or from previous experiments for dynamics learning, it is promising to plan a trajectory for the robot to reduce the dynamics uncertainty, e.g., via Gaussian process regression model of the robot dynamics [169]. It is also compelling to plan a trajectory to explore an unknown environment using the map uncertainty from our learned map representations, e.g., using mutual information [5, 183, 18].

Learning robot dynamics online is another interesting direction where the robot can adapt quickly to dynamics changes.

Safe and robust control with learned dynamics. Analyzing the model error of the learned dynamics for robot safety and robust control is necessary to be able to reliably deploy the learned dynamics model in a robotics system. For example, we can consider the model error as a bounded disturbance applied to the learned system and develop a robust safe tracking controller that takes the disturbance into account via a reference governor design [96, 97, 95]. Given a desired reference path, the governor state serves as a regulation point that moves along the reference path adaptively, balancing the system energy level (Hamiltonian function), model uncertainty bounds, and distance to safety violation to guarantee robustness and safety.

Learning distributed control. Preserving prior knowledge in distributed control policy for multi-robot settings will potentially improve data efficiency in learning from demonstrations and reinforcement learning. For example, an energy-based multi-robot control policy, which models robot interactions by energy exchange and dissipation, can be used to form a physics-informed neural distributed control policy. The policy can be trained from demonstrations of a small team [148] or by maximizing rewards in a reinforcement learning setting [149] and deployed in larger robot teams.

Learning discrete-time dynamics. In many applications such as model predictive control (MPC), robot dynamics are discretized to predict future robot states, e.g. using Euler or Runge-Kutta methods. While our learned continuous-time dynamics preserve Lie group constraints and energy conservation, common integrators do not preserve these laws, leading to high accumulated errors and poor long-term predictions. Preserving Hamiltonian or Lagrangian structure in an integrator to efficiently learn discrete-time dynamics for long-term predictions is a promising direction. For example, Lie group forced variational integrator networks [41] preserve both Lie group constraints and Hamiltonian structure, by discretizing the Lagrange-d'Alembert principle [115] directly instead of the Euler-Lagrange equations of motions.

Chapter A

Software and Supplementary Material

A.1 Sparse Bayesian Occupancy Maps and Collision Checking

Software and videos supplementing our sparse occupancy map representations in Chapters 3 and 5:

https://thaipduong.github.io/sbkm.

A.2 Hamiltonian Dynamics Learning and Control

Software and videos supplementing our Hamiltonian dynamics learning approach and trajectory tracking control design in Chapters 4 and 5:

https://thaipduong.github.io/LieGroupHamDL.

Chapter B

Proofs of Propositions in Chapter 3

B.1 Proof of Proposition 1

Proof. A point $\mathbf{x}$ is considered free if:

$$\mathbf{\Phi}_{\mathbf{x}}^{\top}\boldsymbol{\mu} + b < e\sqrt{1 + \mathbf{\Phi}_{\mathbf{x}}^{\top}\mathbf{\Sigma}\mathbf{\Phi}_{\mathbf{x}}}, \tag{B.1}$$

where $\mathbf{\Phi}_{\mathbf{x}}$ is the feature vector $\mathbf{\Phi}_{\mathbf{x}} = [k_1(\mathbf{x}), k_2(\mathbf{x}), \ldots, k_M(\mathbf{x})]^{\top}$. We use the following lower bound and upper bound on $\mathbf{\Phi}_{\mathbf{x}}^{\top}\mathbf{\Sigma}\mathbf{\Phi}_{\mathbf{x}}$: $0 \leq \mathbf{\Phi}_{\mathbf{x}}^{\top}\mathbf{\Sigma}\mathbf{\Phi}_{\mathbf{x}} \leq \lambda_{\max}\sum_{m=1}^{M}(k_m(\mathbf{x}))^2$ where $\lambda_{\max} \geq 0$ is the largest eigenvalue of the covariance matrix $\mathbf{\Sigma}$. Since $k_m(\mathbf{x}) > 0$ for all m, we have:

$$1 \leq \sqrt{1 + \mathbf{\Phi}_{\mathbf{x}}^{\top}\mathbf{\Sigma}\mathbf{\Phi}_{\mathbf{x}}} \leq 1 + \sqrt{\lambda_{\max}}\sum_{m=1}^{M}(k_m(\mathbf{x})). \tag{B.2}$$

Therefore, the point $\mathbf{x}$ is still free if

$$\mathbf{\Phi}_{\mathbf{x}}^{\top}\boldsymbol{\mu} + b \leq e(1 + \mathbb{1}_{\{e\leq 0\}}\sqrt{\lambda_{\max}}\sum_{m=1}^{M}(k_m(\mathbf{x}))), \tag{B.3}$$

or in other words, we have: $\sum_{m=1}^{M}(\boldsymbol{\mu}_m - e\mathbb{1}_{\{e\leq 0\}}\sqrt{\lambda_{\max}})k_m(\mathbf{x}) + b - e \leq 0$. □

B.2 Proof of Proposition 2

Proof. A point $\mathbf{x}$ is free if Eq. (3.7) holds. Let $\mathbf{x}_*^+$ be the closest positive relevance vector to $\mathbf{x}$ and $\mathbf{x}_j^-$ be any negative relevance vector. We have:

$$\begin{aligned}\sum_{i=1}^{M^+}\nu_i^+k(\mathbf{x},\mathbf{x}_i^+)-\sum_{j=1}^{M^-}\nu_j^-k(\mathbf{x},\mathbf{x}_j^-)+b-e\leq\\ \leq(\sum_{i=1}^{M^+}\nu_i^+)k(\mathbf{x},\mathbf{x}_*^+)-\nu_j^-k(\mathbf{x},\mathbf{x}_j^-)+b-e\end{aligned}$$

Under Assumptions 1 and 2, both terms $\nu_j^-k_j(\mathbf{x})$ and $e-b$ are non-negative. By the arithmetic mean- geometric mean inequality, we have:

$$\begin{aligned}\nu_j^-k(\mathbf{x},\mathbf{x}_j^-)+e-b &= n_2\frac{\nu_j^-k(\mathbf{x},\mathbf{x}_j^-)}{n_2}+n_1\frac{e-b}{n_1}\\ &\geq (n_1+n_2)\left(\frac{\nu_j^-k(\mathbf{x},\mathbf{x}_j^-)}{n_2}\right)^{\frac{n_2}{n_1+n_2}}\left(\frac{e-b}{n_1}\right)^{\frac{n_1}{n_1+n_2}}\\ &= \rho(e-b,\nu_j^-k(\mathbf{x},\mathbf{x}_j^-)).\end{aligned}$$

Therefore, a point $\mathbf{x}$ is free if

$$(\sum_{i=1}^{M^+}\nu_i^+)k(\mathbf{x},\mathbf{x}_*^+)-\rho(e-b,\nu_j^-k(\mathbf{x},\mathbf{x}_j^-))\leq 0. \tag{B.4}$$

□

B.3 Proof of Proposition 3

Proof. By plugging $k(\mathbf{x},\mathbf{x}_*^+)=\eta e^{-\|\Gamma(\mathbf{x}-\mathbf{x}_*^+)\|^2}$, and $k(\mathbf{x},\mathbf{x}_j^-)=\eta e^{-\gamma\|\Gamma(\mathbf{x}-\mathbf{x}_j^-)\|^2}$ into Eq. (B.4), a point $\mathbf{x}$ is free if

$$e^{-\|\mathbf{\Gamma}(\mathbf{x}-\mathbf{x}_*^+)\|^2+\frac{n_2}{n_1+n_2}\|\mathbf{\Gamma}(\mathbf{x}-\mathbf{x}_j^-)\|^2}\leq\frac{\rho(e-b,\nu_j^-)}{\eta^{\frac{n_1}{n_1+n_2}}\sum_{i=1}^{M^+}\nu_i^+} \tag{B.5}$$

Substituting the test point $\mathbf{x}$ by $\mathbf{p}(t)=\mathbf{p}_0+t\mathbf{v}$ in Eq. (B.5), the point $\mathbf{p}(t)$ is free if:

$$\begin{aligned}V(t,\mathbf{x}_*^+,\mathbf{x}_j^-) &= -(n_1+n_2)\|\mathbf{\Gamma}(\mathbf{p}_0+t\mathbf{v}-\mathbf{x}_*^+)\|^2\\ &\quad +n_2\|\mathbf{\Gamma}(\mathbf{p}_0+t\mathbf{v}-\mathbf{x}_j^-)\|^2-(n_1+n_2)\beta\leq 0,\end{aligned}$$

where $\beta = \log \frac{\rho(e-b,\nu_j^-)}{\frac{n_1}{\eta^{\overline{n_1+n_2}}} \sum_{i=1}^{M^+} \nu_i^+}$. By expanding the quadratic norms in $V(t, \mathbf{x}_*^+, \mathbf{x}_j^-)$, the point $\mathbf{p}(t)$ is free if:

$$\begin{aligned} V(t, \mathbf{x}_*^+, \mathbf{x}_j^-) &= at^2 + b(\mathbf{x}_*^+, \mathbf{x}_j^-)t + c(\mathbf{x}_*^+, \mathbf{x}_j^-) \leq 0 \\ \text{where } a &= -n_1 \|\mathbf{\Gamma v}\|^2, \\ b(\mathbf{x}_*^+, \mathbf{x}_j^-) &= -2\mathbf{v}^\top \mathbf{\Gamma}^\top \mathbf{\Gamma}(n_1 \mathbf{p}_0 - (n_1+n_2)\mathbf{x}_*^+ + n_2 \mathbf{x}_j^-), \\ c(\mathbf{x}_*^+, \mathbf{x}_j^-) &= -(n_1+n_2)\|\mathbf{\Gamma}(\mathbf{p}_0 - \mathbf{x}_*^+)\|^2 \\ &\quad + n_2 \|\mathbf{\Gamma}(\mathbf{p}_0 - \mathbf{x}_j^-)\|^2 - (n_1+n_2)\beta. \end{aligned} \tag{B.6}$$

Note that $V(t, \mathbf{x}_*^+, \mathbf{x}_j^-)$ is a quadratic polynomial in t and the point $\mathbf{p}(t)$ is free if $V(t, \mathbf{x}_*^+, \mathbf{x}_j^-) \leq 0$.

1. If it has less than 2 roots, Eq. (B.6) is satisfied for all t.
2. If it has 2 roots $t_1 < t_2$, then $V(t, \mathbf{x}_*^+, \mathbf{x}_j^-) \leq 0$ for $t \geq t_2$ or $t \leq t_1$. There are three cases:
   (a) $t_1 < t_2 \leq 0$: $V(t, \mathbf{x}_*^+, \mathbf{x}_j^-) \leq 0$ for all $t \geq 0$ or the entire ray $s(t)$ is free;
   (b) $0 \leq t_1 < t_2$: $V(t, \mathbf{x}_*^+, \mathbf{x}_j^-) \leq 0$ for $t \in [0, t_1]$ or the ray $s(t)$ is free for $t \in [0, t_1]$.
   (c) $t_1 \leq 0 \leq t_2$: $V(0, \mathbf{x}_*^+, \mathbf{x}_j^-) \geq 0$ or the ray $s(t)$ is colliding.

Let's define

$$\tau(\mathbf{p}_0, \mathbf{x}_*^+, \mathbf{x}_j^-) = \begin{cases} +\infty, & \text{if } V(t, \mathbf{x}_*^+, \mathbf{x}_j^-) \text{ has less than 2 roots} \\ +\infty, & \text{if } V(t, \mathbf{x}_*^+, \mathbf{x}_j^-) \text{ has 2 roots } t_1 < t_2 \leq 0 \\ t_1 & \text{if } V(t, \mathbf{x}_*^+, \mathbf{x}_j^-) \text{ has 2 roots } 0 \leq t_1 < t_2 \\ 0 & \text{if } V(t, \mathbf{x}_*^+, \mathbf{x}_j^-) \text{ has 2 roots } t_1 \leq 0 \leq t_2 \end{cases}.$$

Note that $\mathbf{x}_*^+$ varies with t but belongs to a finite set, we can calculate $\tau(\mathbf{p}_0, \mathbf{x}_i^+, \mathbf{x}_j^-)$ for all positive relevance vectors $\mathbf{x}_i^+$ and take the minimum value. Therefore, $\mathbf{p}(t)$ is free as long as:

$$t \leq t_u = \min_{i=1,\ldots,M^+} \tau(\mathbf{p}_0, \mathbf{x}^+, \mathbf{x}_j^-) \tag{B.7}$$

Note that Eq. (B.7) holds for any negative relevance vector $\mathbf{x}_j^-$. Therefore, the point $\mathbf{p}(t)$ is free as long as $t \leq t_u^* = \max_{j=1,\ldots,M^-} \min_{i=1,\ldots,M^+} \tau(\mathbf{p}_0, \mathbf{x}^+, \mathbf{x}_j^-)$. □

B.4 Proof of Proposition 4

Proof. Consider an arbitrary ray $\mathbf{p}'(t) = \mathbf{p}_0 + t\mathbf{v}', 0 \leq t < \infty$. If we scale the velocity $\mathbf{v}'$ by a positive constant λ, i.e. $\mathbf{v} = \lambda\mathbf{v}'$, the ray $\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v}, 0 \leq t < \infty$ represents the same ray as $\mathbf{p}'(t)$. If we scale the vector v' by $\lambda = \frac{1}{\|\mathbf{\Gamma}v'\|}$, the velocity vector $\mathbf{v}$ satisfies $\|\mathbf{\Gamma}v\| = 1$. Using the Cauchy-Schwarz inequality in Eq. (B.6) in Appendix B.3, we have:

$$\begin{aligned} & -2t\mathbf{v}^\top\mathbf{\Gamma}^\top\mathbf{\Gamma}(n_1\mathbf{p}_0 - (n_1+n_2)\mathbf{x}_*^+ + n_2\mathbf{x}_j^-) \\ \leq \; & 2t\|\mathbf{\Gamma}(n_1\mathbf{p}_0 - (n_1+n_2)\mathbf{x}_*^+ + n_2\mathbf{x}_j^-)\| \end{aligned}$$

Therefore, the point $\mathbf{p}(t)$ is free if $\bar{V}(t, \mathbf{x}_*^+, \mathbf{x}_j^-) \leq 0$. Following the same reasoning as Prop. 3, the point $\mathbf{p}(t)$ is free for $0 < t < r_u$ or $0 < t < r_u^*$. In other words, the interior of the ellipsoids $\mathcal{E}(\mathbf{p}_0, r_u) \subseteq \mathcal{E}(\mathbf{p}_0, r_u^*)$ is free. □

Chapter C

Implementation Details for Chapter 4

We used fully-connected neural networks whose architecture is shown below. The first number is the input dimension while the last number is the output dimension. The numbers in between are the hidden layers' dimensions and activation functions. The value of $\varepsilon_{\mathbf{v}}$ and $\varepsilon_{\boldsymbol{\omega}}$ in (4.7) is set to 0.01.

1. Pendulum:

   - Input dimension: 9. Action dimension: 1.
   - $\mathbf{L}(\mathbf{q})$:
     9 - 300 Tanh - 300 Tanh - 300 Tanh - 300 Linear - 6.
   - $V(\mathbf{q})$: 9 - 50 Tanh - 50 Tanh - 50 Linear - 1.
   - $\mathbf{g}(\mathbf{q})$: 9 - 300 Tanh - 300 Tanh - 300 Linear - 3.

2. Pybullet quadrotor:

   - Input dimension: 12. Action dimension: 4.
   - $\mathbf{L}_1(\mathbf{q})$ only takes the position $\mathbf{p} \in \mathbb{R}^3$ as input:
     3 - 400 Tanh - 400 Tanh - 400 Tanh - 400 Linear - 6.
   - $\mathbf{L}_2(\mathbf{q})$ only takes the rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times 3}$ as input:
     9 - 400 Tanh - 400 Tanh - 400 Tanh - 400 Linear - 6.
   - $V(\mathbf{q})$: 12 - 400 Tanh - 400 Tanh - 400 Linear - 1.
   - $\mathbf{g}(\mathbf{q})$: 12 - 400 Tanh - 400 Tanh - 400 Linear - 24.

3. Real PX4 quadrotor:

   - Input dimension: 12. Action dimension: 4.
   - $\mathbf{L_v}(\mathbf{q})$ only takes the position $\mathbf{p} \in \mathbb{R}^3$ as input:

     3 - 20 Tanh - 20 Tanh - 20 Tanh - 20 Linear - 6.
   - $\mathbf{L_\omega}(\mathbf{q})$ only takes the rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times 3}$ as input:

     9 - 20 Tanh - 20 Tanh - 20 Tanh - 20 Linear - 6.
   - $\mathbf{D_v}(\mathbf{q})$ only takes the position $\mathbf{p} \in \mathbb{R}^3$ as input:

     3 - 20 Tanh - 20 Tanh - 20 Tanh - 20 Linear - 6.
   - $\mathbf{D_\omega}(\mathbf{q})$ only takes the rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times 3}$ as input:

     9 - 20 Tanh - 20 Tanh - 20 Tanh - 20 Linear - 6.
   - $V(\mathbf{q})$: 12 - 20 Tanh - 20 Tanh - 20 Linear - 1.
   - $\mathbf{g}(\mathbf{q})$: 12 - 20 Tanh - 20 Tanh - 20 Linear - 24.

Chapter D

Proof of Theorem 2 in Chapter 5

Proof. We drop function parameters to simplify the notation. The derivative of the generalized coordinate error satisfies:

$$\begin{aligned}\dot{\mathbf{e}} = \begin{bmatrix}\dot{\mathbf{e}}_{\mathbf{p}} \\ \dot{\mathbf{e}}_{\mathbf{R}}\end{bmatrix} &= \begin{bmatrix}-\hat{\boldsymbol{\omega}}\mathbf{e}_{\mathbf{p}} + k_{\mathbf{p}}\mathbf{e}_{\mathbf{v}} \\ k_{\mathbf{R}}\mathbf{E}(\mathbf{R},\mathbf{R}^*)\mathbf{e}_{\boldsymbol{\omega}}\end{bmatrix} \\ &= -\begin{bmatrix}\hat{\boldsymbol{\omega}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{e} + \begin{bmatrix}k_{\mathbf{p}}\mathbf{I} & \mathbf{0} \\ \mathbf{0} & k_{\mathbf{R}}\mathbf{E}(\mathbf{R},\mathbf{R}^*)\end{bmatrix}\mathbf{M}^{-1}\mathbf{p}_e,\end{aligned} \tag{D.1}$$

where $\mathbf{E}(\mathbf{R},\mathbf{R}^*) = \frac{1}{2}\left(\mathrm{tr}(\mathbf{R}^\top\mathbf{R}^*)\mathbf{I} - \mathbf{R}^\top\mathbf{R}^*\right)$ satisfies $\|\mathbf{E}(\mathbf{R},\mathbf{R}^*)\| \leq 1$. By construction of the IDA-PBC controller [37]:

$$\dot{\mathbf{p}}_e = -\mathbf{e} - \mathbf{K}_d\mathbf{M}^{-1}\mathbf{p}_e - \mathbf{W}\mathbf{e}_{\mathbf{a}}. \tag{D.2}$$

Consider the adaptation law $\dot{\mathbf{a}} = c_1\mathbf{W}^\top\mathbf{e} + c_2\mathbf{W}^\top\mathbf{M}^{-1}\mathbf{p}_e$ in (5.35) with $c_1 = c_{\mathbf{p}} = c_{\mathbf{R}}$ and $c_2 = c_{\mathbf{v}} = c_{\boldsymbol{\omega}}$. In the domain $\mathcal{T}$, $\boldsymbol{\Psi}(\mathbf{R},\mathbf{R}^*) < \alpha < 2$ and $\frac{k_{\mathbf{R}}^{-2}}{2}\|\mathbf{e}_{\mathbf{R}}\|_2^2 \leq \boldsymbol{\Psi}(\mathbf{R},\mathbf{R}^*) \leq \frac{k_{\mathbf{R}}^{-2}}{2-\alpha}\|\mathbf{e}_{\mathbf{R}}\|_2^2$ by [53, Prop. 1]. For $\mathbf{z} := [\|\mathbf{e}\| \ \|\mathbf{p}_e\|]^\top \in \mathbb{R}^2$, the Lyapunov function candidate $\mathcal{V}$ in (5.36) is bounded as:

$$\frac{1}{2}\mathbf{z}^\top\mathbf{Q}_1\mathbf{z} + \frac{1}{2c_2}\|\mathbf{e}_{\mathbf{a}}\|_2^2 \leq \mathcal{V} \leq \frac{1}{2}\mathbf{z}^\top\mathbf{Q}_2\mathbf{z} + \frac{1}{2c_2}\|\mathbf{e}_{\mathbf{a}}\|_2^2, \tag{D.3}$$

where the matrix $\mathbf{Q}_1$ is specified in (5.37) and $\mathbf{Q}_2$ is:

$$\mathbf{Q}_2 = \begin{bmatrix}\max\left\{k_{\mathbf{p}}^{-1}, \frac{2k_{\mathbf{R}}^{-1}}{2-\alpha}\right\} & c_1/c_2 \\ c_1/c_2 & \lambda_{\max}(\mathbf{M}^{-1})\end{bmatrix}. \tag{D.4}$$

The time derivative of the Lyapunov candidate satisfies:

$$\begin{aligned}
\dot{\mathcal{V}} &= \mathfrak{p}_e^\top \mathbf{M}^{-1}\dot{\mathfrak{p}}_e + \mathbf{e}^\top \mathbf{M}^{-1}\mathfrak{p}_e + \frac{c_1 \mathbf{e}^\top \dot{\mathfrak{p}}_e}{c_2} + \frac{c_1 \dot{\mathbf{e}}^\top \mathfrak{p}_e}{c_2} + \frac{\mathbf{e}_\mathbf{a}^\top \dot{\mathbf{a}}}{c_2} \\
&= -\mathfrak{p}_e^\top \mathbf{M}^{-1}\mathbf{K_d}\mathbf{M}^{-1}\mathfrak{p}_e - \frac{c_1}{c_2}\mathbf{e}^\top \mathbf{e} \\
&\quad - \frac{c_1}{c_2}\mathbf{e}^\top \mathbf{K_d}\mathbf{M}^{-1}\mathfrak{p}_e + \frac{c_1}{c_2}\mathbf{e}^\top \begin{bmatrix} \hat{\mathbf{e}}_{\boldsymbol{\omega}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathfrak{p}_e \\
&\quad + \frac{c_1}{c_2}\mathbf{e}^\top \begin{bmatrix} \mathbf{R}^\top \mathbf{R}^* \hat{\boldsymbol{\omega}}^* \mathbf{R}^{*\top} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathfrak{p}_e \\
&\quad + \frac{c_1}{c_2}\mathfrak{p}_e^\top \mathbf{M}^{-1} \begin{bmatrix} k_\mathbf{p}\mathbf{I} & \mathbf{0} \\ \mathbf{0} & k_\mathbf{R}\mathbf{E}(\mathbf{R}, \mathbf{R}^*) \end{bmatrix} \mathfrak{p}_e,
\end{aligned}$$

where we use (D.1), (D.2), and that $\boldsymbol{\omega} = \mathbf{e}_{\boldsymbol{\omega}} + \mathbf{R}^\top \mathbf{R}^* \boldsymbol{\omega}^*$ by definition of $\mathbf{e}_{\boldsymbol{\omega}}$. Hence, in the domain $\mathcal{T}$, we have:

$$\frac{d}{dt}\mathcal{V} \leq -\mathbf{z}^\top \mathbf{Q}_3 \mathbf{z} = -\mathbf{z}^\top \begin{bmatrix} q_1 & q_2 \\ q_2 & q_3 \end{bmatrix} \mathbf{z}, \tag{D.5}$$

where

$$\begin{aligned}
q_1 &= \frac{c_1}{c_2}, \\
q_2 &= -\frac{c_1}{c_2}\left(\lambda_{\max}(\mathbf{K_d}\mathbf{M}^{-1}) + \beta + \gamma\right), \text{and} \\
q_3 &= \lambda_{\min}(\mathbf{M}^{-1}\mathbf{K_d}\mathbf{M}^{-1}) - \frac{c_1}{c_2}\max\left\{k_\mathbf{p}, k_\mathbf{R}\right\}\lambda_{\max}(\mathbf{M}^{-1}).
\end{aligned} \tag{D.6}$$

Since $\mathbf{K_d} = \operatorname{diag}(k_\mathbf{v}\mathbf{I}, k_{\boldsymbol{\omega}}\mathbf{I})$ can be chosen arbitrarily large and c_1/c_2 can be chosen arbitrarily small, there exists a choice of constants that ensures that the matrices $\mathbf{Q}_1$, $\mathbf{Q}_2$, and $\mathbf{Q}_3$ are positive definite. Consider the sub-level set of the Lyapunov function $\mathcal{R} = \{\mathbf{x} \in \mathcal{T} | \mathcal{V}(\mathbf{x}) \leq \delta\}$ where $\delta < \lambda_{\min}(\mathbf{Q}_1)\min(\alpha(2-\alpha)k_\mathbf{R}^2, \beta^2\lambda_{\min}^2(\mathbf{M}))/2$. For $\mathbf{x}_0 \in \mathcal{R}$, we have $\boldsymbol{\Psi}(\mathbf{R}, \mathbf{R}^*) \leq \frac{k_\mathbf{R}^{-2}\|\mathbf{e}_\mathbf{R}\|_2^2}{(2-\alpha)} \leq \frac{2\delta k_\mathbf{R}^{-2}}{(2-\alpha)\lambda_{\min}(\mathbf{Q}_1)} < \alpha$, and $\|\mathbf{e}_{\boldsymbol{\omega}}(\mathbf{x}, \mathbf{x}^*)\|^2 \leq \frac{2\delta}{\lambda_{\min}(\mathbf{Q}_1)\lambda_{\min}^2(\mathbf{M})} \leq \beta^2$ for all $\mathbf{x}(t), t > 0$, i.e., $d\mathcal{V}/dt \leq 0$ for all $t > 0$ and $\mathcal{R}$ is a positively invariant set. Therefore, for any system trajectory starting in $\mathcal{R}$, the tracking errors $\mathbf{e}$, $\mathfrak{p}_e$ are asymptotically stable, while the estimation error $\mathbf{e_a}$ is stable and uniformly bounded, by the LaSalle-Yoshizawa theorem [89, Thm. A.8]. □

Bibliography

[1] José Ángel Acosta, MI Sanchez, and Aníbal Ollero. Robust Control of Underactuated Aerial Manipulators via IDA-PBC. In *IEEE Conference on Decision and Control*, 2014.

[2] Abdullah Altawaitan, Jason Stanley, Sambaran Ghosal, Thai Duong, and Nikolay Atanasov. Hamiltonian Dynamics Learning from Point Cloud Observations for Nonholonomic Mobile Robot Control. In *IEEE International Conference on Robotics and Automation*, 2024.

[3] Philip Arathoon. Coadjoint orbits of the special Euclidean group. Master's thesis, University of Manchester, 2015.

[4] Omur Arslan and Daniel Koditschek. Exact Robot Navigation Using Power Diagrams. In *IEEE International Conference on Robotics and Automation*, 2016.

[5] Arash Asgharivaskasi and Nikolay Atanasov. Semantic Octree Mapping and Shannon Mutual Information Computation for Robot Exploration. *IEEE Transactions on Robotics*, 2023.

[6] Timothy Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017.

[7] Thomas Beckers. Data-driven Bayesian Control of Port-Hamiltonian Systems. In *IEEE Conference on Decision and Control*, 2023.

[8] Thomas Beckers, Jacob Seidman, Paris Perdikaris, and George Pappas. Gaussian Process Port-Hamiltonian Systems: Bayesian Learning with Physics Prior. In *IEEE Conference on Decision and Control*, 2022.

[9] Jens Behley and Cyrill Stachniss. Efficient Surfel-based SLAM Using 3D Laser Range Data in Urban Environments. In *Robotics: Science and Systems*, 2018.

[10] Tom Bertalan, Felix Dietrich, Igor Mezić, and Ioannis Kevrekidis. On Learning Hamiltonian Systems from Data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 2019.

[11] Joshua Bialkowski, Michael Otte, Sertac Karaman, and Emilio Frazzoli. Efficient Collision Checking in Sampling-based Motion Planning via Safety Certificates. *International Journal of Robotics Research*, 2016.

[12] Mahdis Bisheban and Taeyoung Lee. Geometric Adaptive Control with Neural Networks for a Quadrotor in Wind Fields. *IEEE Transactions on Control Systems Technology*, 2020.

[13] Antoine Bordes, Seyda Ertekin, Jason Weston, Léon Botton, and Nello Cristianini. Fast Kernel Classifiers with Online and Active Learning. *Journal of Machine Learning Research*, 2005.

[14] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[15] Nicolas Boumal. *An introduction to optimization on smooth manifolds*. Cambridge University Press, 2023.

[16] Sean Bowman, Nikolay Atanasov, Kostas Daniilidis, and George Pappas. Probabilistic Data Association for Semantic SLAM. In *IEEE International Conference on Robotics and Automation*, 2017.

[17] Russell Buchanan, Varun Agrawal, Marco Camurri, Frank Dellaert, and Maurice Fallon. Deep IMU Bias Inference for Robust Visual-Inertial Odometry with Factor Graphs. *IEEE Robotics and Automation Letters*, 2022.

[18] Benjamin Charrow, Sikang Liu, Vijay Kumar, and Nathan Michael. Information-theoretic Mapping using Cauchy-Schwarz Quadratic Mutual Information. In *IEEE International Conference on Robotics and Automation*, 2015.

[19] Jing Chen and Shaojie Shen. Improving Octree-based Occupancy Maps Using Environment Sparsity with Application to Aerial Robot Navigation. In *IEEE International Conference on Robotics and Automation*, 2017.

[20] Renyi Chen and Molei Tao. Data-driven Prediction of General Hamiltonian Dynamics via Learning Exactly-symplectic Maps. In *International Conference on Machine Learning*, 2021.

[21] Ricky Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems*, 2018.

[22] Wen-Hua Chen, Donald J Ballance, Peter J Gawthrop, and John O'Reilly. A Nonlinear Disturbance Observer for Robotic Manipulators. *IEEE Transactions on industrial Electronics*, 2000.

[23] Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic Recurrent Neural Networks. *International Conference on Learning Representations*, 2020.

[24] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep Reinforcement Learning in a Handful of Trials Using Probabilistic Dynamics Models. In *Advances in Neural Information Processing Systems*, 2018.

[25] Oscar Cieza and Johann Reger. IDA-PBC for Underactuated Mechanical Systems in Implicit Port-Hamiltonian Representation. In *European Control Conference*, 2019.

[26] Giovanni Cioffi, Leonard Bauersfeld, Elia Kaufmann, and Davide Scaramuzza. Learned Inertial Odometry for Autonomous Drone Racing. In *IEEE Robotics and Automation Letters*, 2023.

[27] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian Neural Networks. In *ICLR Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[28] Brian Curless and Marc Levoy. A Volumetric Method for Building Complex Models from Range Images. In *Conference on Computer Graphics and Interactive Techniques*, pages 303–312, 1996.

[29] Zhirui Dai, Arash Asgharivaskasi, Thai Duong, Shusen Lin, Maria-Elizabeth Tzes, George Pappas, and Nikolay Atanasov. Optimal scene graph planning with large language model guidance. In *IEEE International Conference on Robotics and Automation*, 2024.

[30] Nikhil Das, Naman Gupta, and Michael Yip. Fastron: An Online Learning-based Model and Active Learning Strategy for Proxy Collision Detection. In *Conference on Robot Learning*, 2017.

[31] Nikhil Das and Michael Yip. Learning-based Proxy Collision Detection for Robot Motion Planning Applications. *IEEE Transactions on Robotics*, 2020.

[32] Alessandro De Luca, Giuseppe Oriolo, and Marilena Vendittelli. Stabilization of the Unicycle via Dynamic Feedback Linearization. *IFAC Proceedings Volumes*, 2000.

[33] Marc Deisenroth and Carl Rasmussen. PILCO: A Model-based and Data-efficient Approach to Policy Search. In *International Conference on Machine Learning*, 2011.

[34] J. Delmerico and D. Scaramuzza. A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots. In *IEEE International Conference on Robotics and Automation*, 2018.

[35] Daniel Dirksz and Jacquelien Scherpen. Structure Preserving Adaptive Control of Port-Hamiltonian Systems. *IEEE Transactions on Automatic Control*, 2012.

[36] Thai Duong, Abdullah Altawaitan, Jason Stanley, and Nikolay Atanasov. Port-Hamiltonian Neural ODE Networks on Lie Groups For Robot Dynamics Learning and Control. *arXiv preprint arXiv:2401.09520*, 2024.

[37] Thai Duong and Nikolay Atanasov. Hamiltonian-based Neural ODE Networks on the SE(3) Manifold For Dynamics Learning and Control. In *Robotics: Science and Systems*, 2021.

[38] Thai Duong and Nikolay Atanasov. Adaptive Control of SE(3) Hamiltonian Dynamics With Learned Disturbance Features. *IEEE Control Systems Letters*, 2022.

[39] Thai Duong, Nikhil Das, Michael Yip, and Nikolay Atanasov. Autonomous Navigation in Unknown Environments Using Sparse Kernel-based Occupancy Mapping. In *IEEE International Conference on Robotics and Automation*, 2020.

[40] Thai Duong, Michael Yip, and Nikolay Atanasov. Autonomous Navigation in Unknown Environments With Sparse Bayesian Kernel-Based Occupancy Mapping. *IEEE Transactions on Robotics*, 2022.

[41] Valentin Duruisseaux, Thai P Duong, Melvin Leok, and Nikolay Atanasov. Lie Group Forced Variational Integrator Networks for Learning and Control of Robot Systems. In *Learning for Dynamics and Control Conference*, 2023.

[42] Karthik Elamvazhuthi, Xuechen Zhang, Samet Oymak, and Fabio Pasqualetti. Learning on Manifolds: Universal Approximations Properties using Geometric Controllability Conditions for Neural ODEs. In *Learning for Dynamics and Control Conference*, 2023.

[43] Alberto Elfes. Using Occupancy Grids for Mobile Robot Perception and Navigation. *Computer*, 1989.

[44] Matthias Faessler, Antonio Franchi, and Davide Scaramuzza. Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-speed Trajectories. *IEEE Robotics and Automation Letters*, 2017.

[45] Luca Falorsi and Patrick Forré. Neural Ordinary Differential Equations on Manifolds. *arXiv preprint arXiv:2006.06663*, 2020.

[46] Marc Finzi, Ke Alexander Wang, and Andrew Wilson. Simplifying Hamiltonian and Lagrangian Neural Networks via Explicit Constraints. In *Advances in Neural Information Processing Systems*, 2020.

[47] Paolo Forni, Dimitri Jeltsema, and Gabriel Lopes. Port-Hamiltonian Formulation of Rigid-body Attitude Control. *IFAC-PapersOnLine*, 2015.

[48] Jaume Franch and Jose-Manuel Rodriguez-Fortun. Control and Trajectory Generation of an Ackerman Vehicle by Dynamic Linearization. In *European Control Conference*, 2009.

[49] David Fridovich-Keil, Erik Nelson, and Avideh Zakhor. AtomMap: A Probabilistic Amorphous 3D Map Representation for Robotics and Surface Reconstruction. In *IEEE International Conference on Robotics and Automation*, 2017.

[50] Luca Furieri, Clara Lucía Galimberti, Muhammad Zakwan, and Giancarlo Ferrari-Trecate. Distributed Neural Network Control with Dependability Guarantees: A Compositional Port-Hamiltonian Approach. In *Learning for Dynamics and Control Conference*, 2022.

[51] Aditya Gahlawat, Pan Zhao, Andrew Patterson, Naira Hovakimyan, and Evangelos Theodorou. L1-GP: L1 Adaptive Control with Bayesian Learning. In *Conference on Learning for Dynamics and Control*, 2020.

[52] Clara Lucía Galimberti, Luca Furieri, Liang Xu, and Giancarlo Ferrari-Trecate. Hamiltonian Deep Neural Networks Guaranteeing Nonvanishing Gradients by Design. *IEEE Transactions on Automatic Control*, 2023.

[53] Farhad Goodarzi, Daewon Lee, and Taeyoung Lee. Geometric Adaptive Tracking Control of a Quadrotor Unmanned Aerial Vehicle on SE(3) for Agile Maneuvers. *Journal of Dynamic Systems, Measurement, and Control*, 2015.

[54] Robert Grande, Girish Chowdhary, and Jonathan How. Nonparametric Adaptive Control Using Gaussian Processes with Online Hyperparameter Estimation. In *IEEE Conference on Decision and Control*, 2013.

[55] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks. In *Advances in Neural Information Processing Systems*, 2019.

[56] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics*, 2007.

[57] Lars Grüne and Jürgen Pannek. *Nonlinear Model Predictive Control*. Springer, 2017.

[58] Vitor Guizilini and Fabio Ramos. Large-scale 3D Scene Reconstruction with Hilbert Maps. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2016.

[59] Vitor Guizilini and Fabio Ramos. Towards Real-time 3D Continuous Occupancy Mapping Using Hilbert Maps. *International Journal of Robotics Research*, 2018.

[60] Jayesh Gupta, Kunal Menda, Zachary Manchester, and Mykel Kochenderfer. A General Framework for Structured Learning of Mechanical Systems. *arXiv preprint arXiv:1902.08705*, 2019.

[61] Jerome Guzzi, Alessandro Giusti, Luca Gambardella, Guy Theraulaz, and Gianni Di Caro. Human-friendly Robot Navigation in Dynamic Environments. In *IEEE International Conference on Robotics and Automation*, 2013.

[62] Brian Hall. *Lie Groups, Lie Algebras, and Representations*. Springer, 2013.

[63] Luxin Han, Fei Gao, Boyu Zhou, and Shaojie Shen. FIESTA: A Fast Incremental Euclidean Distance Fields for Online Quadrotor Motion Planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019.

[64] Drew Hanover, Philipp Foehn, Sihao Sun, Elia Kaufmann, and Davide Scaramuzza. Performance, Precision, and Payloads: Adaptive Nonlinear MPC for Quadrotors. *arXiv preprint arXiv:2109.04210*, 2021.

[65] James Harrison, Apoorva Sharma, and Marco Pavone. Meta-learning Priors for Efficient Online Bayesian Regression. In *Workshop on the Algorithmic Foundations of Robotics*, 2020.

[66] Kris Hauser. Lazy Collision Checking in Asymptotically-Optimal Motion Planning. In *IEEE International Conference on Robotics and Automation*, 2015.

[67] Aaron Havens and Girish Chowdhary. Forced Variational Integrator Networks for Prediction and Control of Mechanical Systems. In *Learning for Dynamics and Control*, 2021.

[68] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. RGB-D Mapping: Using Kinect-style Depth Cameras for Dense 3D Modeling of Indoor Environments. *International Journal of Robotics Research*, 2012.

[69] Darryl Holm. *Geometric Mechanics*. World Scientific Publishing Company, 2008.

[70] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots*, 2013.

[71] Naira Hovakimyan and Chengyu Cao. $\mathcal{L}_1$ *Adaptive Control Theory: Guaranteed Robustness with Fast Adaptation*. SIAM, 2010.

[72] Andrew Howard and Nicholas Roy. The Robotics Dataset Repository (Radish), 2003.

[73] Jinwook Huh and Daniel Lee. Learning High-dimensional Mixture Models for Fast Collision Detection in Rapidly-exploring Random Trees. In *IEEE International Conference on Robotics and Automation*, 2016.

[74] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to Train Your Robot with Deep Reinforcement Learning: Lessons We Have Learned. *International Journal of Robotics Research*, 2021.

[75] Petros Ioannou and Jing Sun. *Robust Adaptive Control*. Prentice Hall, 1996.

[76] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *ACM Sym. on User Interface Software and Technology (UIST)*, 2011.

[77] Maani Ghaffari Jadidi, Jaime Valls Miro, and Gamini Dissanayake. Warped Gaussian Processes Occupancy Mapping with Uncertain Inputs. *IEEE Robotics and Automation Letters*, 2017.

[78] Lucas Janson, Tommy Hu, and Marco Pavone. Safe Motion Planning in Unknown Environments: Optimality Benchmarks and Tractable Policies. In *Robotics: Science and Systems*, 2018.

[79] Wanxin Jin, Alp Aydinoglu, Mathew Halm, and Michael Posa. Learning Linear Complementarity Systems. In *Learning for Dynamics and Control Conference*, 2022.

[80] Girish Joshi, Jasvir Virdi, and Girish Chowdhary. Asynchronous Deep Model Reference Adaptive Control. In *Conference on Robot Learning*, 2020.

[81] Michael Kaess. Simultaneous Localization and Mapping with Infinite Planes. In *IEEE International Conference on Robotics and Automation*, 2015.

[82] Olaf Kähler, Victor Prisacariu, and David Murray. Real-time Large-scale Dense 3D Reconstruction with Loop Closure. In *European Conference on Computer Vision*, 2016.

[83] Sertac Karaman and Emilio Frazzoli. Incremental Sampling-based Algorithms for Optimal Motion Planning. *Robotics: Science and Systems*, 2010.

[84] Sertac Karaman and Emilio Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. *International Journal of Robotics Research*, 2011.

[85] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson Surface Reconstruction. In *Eurographics Symposium on Geometry Processing*, 2006.

[86] Hassan K Khalil. *Nonlinear Systems*. Prentice hall, 2002.

[87] Matthew Klingensmith, Ivan Dryanovski, Siddhartha Srinivasa, and Jizhong Xiao. Chisel: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device. In *Robotics: Science and Systems*, 2015.

[88] Sven Koenig and Yury Smirnov. Sensor-based Planning with the Free Space Assumption. In *IEEE International Conference on Robotics and Automation*, 1997.

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

[90] Taeyoung Lee, Melvin Leok, and N Harris McClamroch. Geometric tracking control of a quadrotor UAV on SE(3). In *IEEE Conference on Decision and Control*, 2010.

[91] Taeyoung Lee, Melvin Leok, and N Harris McClamroch. *Global Formulations of Lagrangian and Hamiltonian Dynamics on Manifolds*. Springer, 2017.

[92] Richard Lehoucq, Danny Sorensen, and Chao Yang. *ARPACK Users' Guide: Solution of Large-scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, 1998.

[93] Benedict Leimkuhler and Sebastian Reich. *Simulating Hamiltonian Dynamics*. Cambridge University Press, 2004.

[94] Shihua Li, Jun Yang, Wen-Hua Chen, and Xisong Chen. *Disturbance Observer-based Control: Methods and Applications*. CRC press, 2014.

[95] Zhichao Li, Thai Duong, and Nikolay Atanasov. Safe Robot Navigation in Cluttered Environments Using Invariant Ellipsoids and a Reference Governor. *arXiv preprint arXiv:2005.06694*, 2020.

[96] Zhichao Li, Thai Duong, and Nikolay Atanasov. Robust and Safe Autonomous Navigation for Systems with Learned SE(3) Hamiltonian Dynamics. *IEEE Open Journal of Control Systems*, 2022.

[97] Zhichao Li, Thai Duong, and Nikolay Atanasov. Safe Autonomous Navigation for Systems with Learned SE(3) Hamiltonian Dynamics. In *Learning for Dynamics and Control Conference*. PMLR, 2022.

[98] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime Search in Dynamic Graphs. *Artificial Intelligence*, 2008.

[99] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based Motion Planning for Quadrotors using Linear Quadratic Minimum Time Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[100] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based Motion Planning for Quadrotors Using Linear Quadratic Minimum Time Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[101] Wenxin Liu, David Caruso, Eddy Ilg, Jing Dong, Anastasios Mourikis, Kostas Daniilidis, Vijay Kumar, and Jakob Engel. TLIO: Tight Learned Inertial Odometry. *IEEE Robotics and Automation Letters*, 2020.

[102] Lennart Ljung. System Identification. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 1999.

[103] Brett Lopez and Jonathan How. Aggressive 3-D Collision Avoidance for High-speed Navigation. In *IEEE International Conference on Robotics and Automation*, 2017.

[104] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning High-speed Flight in the Wild. *Science Robotics*, 2021.

[105] Aaron Lou, Derek Lim, Isay Katsman, Leo Huang, Qingxuan Jiang, Ser Nam Lim, and Christopher De Sa. Neural Manifold Ordinary Differential Equations. In *Advances in Neural Information Processing Systems*, 2020.

[106] Yupu Lu, Shijie Lin, Guanqi Chen, and Jia Pan. ModLaNets: Learning Generalisable Dynamics via Modularity and Physical Inductive Bias. In *International Conference on Machine Learning*, 2022.

[107] Jingru Luo and Kris Hauser. An Empirical Study of Optimal Motion Planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[108] Anatolii Isakovich Lurie. *Analytical Mechanics*. Springer Science & Business Media, 2013.

[109] Michael Lutter, Kim Listmann, and Jan Peters. Deep Lagrangian Networks for End-to-end Learning of Energy-based Control for Under-actuated Systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019.

[110] Michael Lutter and Jan Peters. Combining Physics and Deep Learning to Learn Continuous-time Dynamics Models. *International Journal of Robotics Research*, 2023.

[111] Michael Lutter, Christian Ritter, and Jan Peters. Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning. In *International Conference on Learning Representations*, 2018.

[112] Kevin Lynch and Frank Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[113] Andrzej Maciejewski. Hamiltonian Formalism for Euler Parameters. *Celestial Mechanics*, 1985.

[114] David MacKay. The Evidence Framework Applied to Classification Networks. *Neural Computation*, 1992.

[115] J. E. Marsden and M. West. Discrete Mechanics and Variational Integrators. *Acta Numerica*, 2001.

[116] Jerrold Marsden and Tudor Ratiu. *Introduction to Mechanics and Symmetry: A Basic Exposition of Classical Mechanical Systems*. Springer Science & Business Media, 2013.

[117] Justice Mason, Christine Allen-Blanchette, Nicholas Zolman, Elizabeth Davison, and Naomi Leonard. Learning Interpretable Dynamics from Images of a Freely Rotating 3D Rigid Body. *arXiv preprint arXiv:2209.11355*, 2022.

[118] S. A. S. Mohamed, M. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen, and J. Plosila. A Survey on Odometry for Autonomous Navigation Systems. *IEEE Access*, 2019.

[119] Manasi Muglikar, Zichao Zhang, and Davide Scaramuzza. Voxel Map for Visual SLAM. In *IEEE International Conference on Robotics and Automation*, 2020.

[120] Ian Nabney. Efficient Training of RBF Networks for Classification. *International Journal of Neural Systems*, 2004.

[121] Subramanya Nageshrao, Gabriel Lopes, Dimitri Jeltsema, and Robert Babuška. Port-Hamiltonian Systems in Adaptive and Learning Control: A Survey. *IEEE Transactions on Automatic Control*, 2015.

[122] Radford Neal. *Bayesian Learning for Neural Networks*. Springer Science & Business Media, 2012.

[123] Cyrus Neary and Ufuk Topcu. Compositional Learning of Dynamical System Models Using Port-Hamiltonian Neural Networks. In *Learning for Dynamics and Control Conference*, 2023.

[124] Lachlan Nicholson, Michael Milford, and Niko Sünderhauf. QuadricSLAM: Dual Quadrics from Object Detections as Landmarks in Object-Oriented SLAM. *IEEE Robotics and Automation Letters*, 2019.

[125] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3D Reconstruction at Scale Using Voxel Hashing. *ACM Transactions on Graphics*, 32(6), 2013.

[126] Simon O'Callaghan and Fabio Ramos. Gaussian Process Occupancy Maps. *International Journal of Robotics Research*, 2012.

[127] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-board MAV Planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[128] Romeo Ortega, Mark Spong, Fabio Gómez-Estern, and Guido Blankenstein. Stabilization of a Class of Underactuated Mechanical Systems via Interconnection and Damping Assignment. *IEEE Transactions on Automatic Control*, 47(8), 2002.

[129] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A General Purpose Library for Collision and Proximity Queries. In *IEEE International Conference on Robotics and Automation*, 2012.

[130] Jia Pan and Dinesh Manocha. Efficient Configuration Space Construction and Optimization for Motion Planning. *Engineering*, 2015.

[131] Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela Schöllig. Learning to Fly: a PyBullet Gym Environment to Learn the Control of Multiple Nano-quadcopters. https://github.com/utiasDSL/gym-pybullet-drones, 2020.

[132] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, 2019.

[133] Karime Pereida and Angela Schoellig. Adaptive Model Predictive Control for High-accuracy Trajectory Tracking in Changing Conditions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018.

[134] Samuel Pfrommer, Mathew Halm, and Michael Posa. Contactnets: Learning Discontinuous Contact Dynamics with Smooth, Implicit Representations. In *Conference on Robot Learning*, 2021.

[135] Enrico Piazza, Andrea Romanoni, and Matteo Matteucci. Real-time CPU-based Large-scale Three-dimensional Mesh Reconstruction. *IEEE Robotics and Automation Letters*, 2018.

[136] Maziar Raissi, Paris Perdikaris, and George Karniadakis. Multistep Neural Networks for Data-driven Discovery of Nonlinear Dynamical Systems. *arXiv preprint arXiv:1801.01236*, 2018.

[137] Fabio Ramos and Lionel Ott. Hilbert Maps: Scalable Continuous Occupancy Mapping with Stochastic Gradient Descent. *International Journal of Robotics Research*, 2016.

[138] Ramy Rashad, Federico Califano, and Stefano Stramigioli. Port-Hamiltonian Passivity-based Control on SE(3) of a Fully Actuated UAV for Aerial Physical Interaction Near-hovering. *IEEE Robotics and Automation Letters*, 2019.

[139] Spencer Richards, Navid Azizan, Jean-Jacques Slotine, and Marco Pavone. Adaptive-Control-Oriented Meta-Learning for Nonlinear Systems. In *Robotics: Science and Systems*, 2021.

[140] Manuel Roehrl, Thomas Runkler, Veronika Brandtstetter, Michel Tokic, and Stefan Obermayer. Modeling System Dynamics with Physics-Informed Neural Networks Based on Lagrangian Mechanics. *IFAC-PapersOnLine*, 2020.

[141] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: An Open-source Library for Real-time Metric-semantic Localization and Mapping. In *IEEE International Conference on Robotics and Automation*, 2020.

[142] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 2009.

[143] Lars Ruthotto and Eldad Haber. Deep Neural Networks Motivated by Partial Differential Equations. *Journal of Mathematical Imaging and Vision*, 2019.

[144] Tim Salzmann, Elia Kaufmann, Jon Arrizabalaga, Marco Pavone, Davide Scaramuzza, and Markus Ryll. Real-time Neural MPC: Deep Learning Model Predictive Control for Quadrotors and Agile Robotic Platforms. *IEEE Robotics and Automation Letters*, 2023.

[145] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph Networks as Learnable Physics Engines for Inference and Control. In *International Conference on Machine Learning*, 2018.

[146] Sosale Shankara Sastry and Alberto Isidori. Adaptive Control of Linearizable Systems. *IEEE Transactions on Automatic Control*, 1989.

[147] Davide Scaramuzza and Elia Kaufmann. Learning Agile, Vision-Based Drone Flight: From Simulation to Reality. In *The International Symposium of Robotics Research*, 2022.

[148] Eduardo Sebastián, Thai Duong, Nikolay Atanasov, Eduardo Montijano, and Carlos Sagüés. LEMURS: Learning Distributed Multi-robot Interactions. In *IEEE International Conference on Robotics and Automation*, 2023.

[149] Eduardo Sebastian, Thai Duong, Nikolay Atanasov, Eduardo Montijano, and Carlos Sagues. Physics-Informed Multi-Agent Reinforcement Learning for Distributed Multi-Robot Problems. *arXiv preprint arXiv:2401.00212*, 2023.

[150] Ransalu Senanayake and Fabio Ramos. Bayesian Hilbert Maps for Dynamic Continuous Occupancy Mapping. In *Conference on Robot Learning*, 2017.

[151] Ransalu Senanayake, Anthony Tompkins, and Fabio Ramos. Automorphing Kernels for Nonstationarity in Mapping Unstructured Environments. In *Conference on Robot Learning*, 2018.

[152] Mo Shan, Qiaojun Feng, and Nikolay Atanasov. OrcVIO: Object Residual Constrained Visual-Inertial Odometry. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.

[153] Guanya Shi, Kamyar Azizzadenesheli, Soon-Jo Chung, and Yisong Yue. Meta-Adaptive Nonlinear Control: Theory and Algorithms. In *Advances in Neural Information Processing Systems*, 2021.

[154] Ravishankar Shivarama and Eric Fahrenthold. Hamilton's Equations with Euler Parameters for Rigid Body Dynamics Modeling. *Journal of Dynamic Systems, Measurement, and Control*, 2004.

[155] Jean-Jacques Slotine and MD Di Benedetto. Hamiltonian Adaptive Control of Spacecraft. *IEEE Transactions on Automatic Control*, 1990.

[156] Jean-Jacques Slotine and Weiping Li. On the Adaptive Control of Robot Manipulators. *The International Journal of Robotics Research*, 1987.

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

[158] Oswin So, Gongjie Li, Evangelos Theodorou, and Molei Tao. Data-driven Discovery of Non-Newtonian Astronomy via Learning Non-Euclidean Hamiltonian. In *ICML Machine Learning and the Physical Sciences Workshop*, 2022.

[159] Yunlong Song, Angel Romero, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Reaching the Limit in Autonomous Racing: Optimal Control Versus Reinforcement Learning. *Science Robotics*, 2023.

[160] Cristian Souza, Vianna Raffo, and Eugenio Castelan. Passivity-based Control of a Quadrotor UAV. *IFAC Proceedings Volumes*, 2014.

[161] Bhavya Sukhija, Nathanael Köhler, Miguel Zamora, Simon Zimmermann, Sebastian Curi, Andreas Krause, and Stelian Coros. Gradient-based Trajectory Optimization with Learned Dynamics. In *IEEE International Conference on Robotics and Automation*, 2023.

[162] Lucas Teixeira and Margarita Chli. Real-time Mesh-based Scene Estimation for Aerial Inspection. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016.

[163] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT press, 2005.

[164] Michael Tipping. The Relevance Vector Machine. In *Advances in Neural Information Processing Systems*, 2000.

[165] Michael Tipping. Sparse Bayesian Learning and the Relevance Vector Machine. *Journal of Machine Learning Research*, 2001.

[166] Michael Tipping, Anita Faul, et al. Fast Marginal Likelihood Maximisation for Sparse Bayesian Models. In *International Conference on Artificial Intelligence and Statistics*, 2003.

[167] Anthony Tompkins, Ransalu Senanayake, and Fabio Ramos. Online Domain Adaptation for Occupancy Mapping. In *Robotics: Science and Systems*, 2020.

[168] Peter Toth, Danilo J Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian Generative Networks. In *International Conference on Learning Representations*, 2019.

[169] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe Exploration for Interactive Machine Learning. In *Advances in Neural Information Processing Systems*, 2019.

[170] Arjan Van Der Schaft and Dimitri Jeltsema. Port-Hamiltonian Systems Theory: An Introductory Overview. *Foundations and Trends in Systems and Control*, 2014.

[171] Emanuele Vespa, Nikolay Nikolov, Marius Grimm, Luigi Nardi, Paul Kelly, and Stefan Leutenegger. Efficient Octree-based Volumetric SLAM Supporting Signed-distance and Occupancy Mapping. *IEEE Robotics and Automation Letters*, 2018.

[172] Jinkun Wang and Brendan Englot. Fast, Accurate Gaussian Process Occupancy Maps via Test-data Octrees and Nested Bayesian Fusion. In *IEEE International Conference on Robotics and Automation*, 2016.

[173] Kaixuan Wang, Fei Gao, and Shaojie Shen. Real-time Scalable Dense Surfel Mapping. In *IEEE International Conference on Robotics and Automation*, 2019.

[174] Rui Wang, Robin Walters, and Rose Yu. Incorporating Symmetry into Deep Dynamics Models for Improved Generalization. In *International Conference on Learning Representations*, 2021.

[175] Z Wang and P Goldsmith. Modified Energy-balancing-based Control for the Tracking Problem. *IET Control Theory & Applications*, 2008.

[176] Thomas Whelan, Renato Salas-Moreno, Ben Glocker, Andrew Davison, and Stefan Leutenegger. ElasticFusion: Real-time Dense SLAM and Light Source Estimation. *International Journal of Robotics Research*, 2016.

[177] Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating Physics-based Modeling with Machine Learning: A Survey. *arXiv preprint arXiv:2003.04919*, 2020.

[178] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James Rehg, Byron Boots, and Evangelos Theodorou. Information Theoretic MPC for Model-based Reinforcement Learning. In *IEEE International Conference on Robotics and Automation*, 2017.

[179] Yannik Wotte, Federico Califano, and Stefano Stramigioli. Optimal Potential Shaping on SE(3) via Neural ODEs on Lie Groups. *arXiv preprint arXiv:2401.15107*, 2024.

[180] Shichao Yang and Sebastian Scherer. CubeSLAM: Monocular 3-D Object SLAM. *IEEE Transactions on Robotics*, 2019.

[181] Ming Zeng, Fukai Zhao, Jiaxiang Zheng, and Xinguo Liu. Octree-based Fusion for Realtime 3D Reconstruction. *Graphical Models*, 2013.

[182] Kunyi Zhang, Chenxing Jiang, Jinghang Li, Sheng Yang, Teng Ma, Chao Xu, and Fei Gao. DIDO: Deep Inertial Quadrotor Dynamical Odometry. *IEEE Robotics and Automation Letters*, 2022.

[183] Zhengdong Zhang, Theia Henderson, Sertac Karaman, and Vivienne Sze. FSMI: Fast Computation of Shannon Mutual Information for Information-Theoretic Mapping. *International Journal of Robotics Research*, 2020.

[184] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control. In *International Conference on Learning Representations*, 2019.

[185] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Dissipative SymODEN: Encoding Hamiltonian Dynamics with Dissipation and Control into Deep Learning. In *ICLR Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.