Learning to represent surroundings, anticipate motion and take informed actions in unstructured environments

Weiming Zhi

A thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

Faculty of Engineering

University of Sydney

Australia

2023

Declaration

Weiming Zhi
January 2023

Author Attribution and Publications

Elements of this thesis have appeared in peer-reviewed conferences and journal publications. I am the first author of each of these papers. My contributions include, but are not limited to, conceptualising the method, developing the theory, designing and conducting the experiments, and writing the paper. The specific papers are listed here:

1. Chapter 3 has appeared as: (Zhi et al., 2019a) **W. Zhi**, L. Ott, R. Senanayake, F. Ramos. Continuous Occupancy Map Fusion with Fast Bayesian Hilbert Maps. *International Conference on Robotics and Automation (ICRA)*, 2019.

2. Chapter 4 has appeared as: (Zhi et al., 2019b) **W. Zhi**, R. Senanayake, L. Ott, F. Ramos. Spatiotemporal Learning of Directional Uncertainty in Urban Environments. *IEEE Robotics and Automation Letters (RA-L)*, 2019.

3. Chapter 5 has appeared as: (Zhi et al., 2021b) **W. Zhi**, T. Lai, L. Ott, F. Ramos. Anticipatory Navigation in Crowds by Probabilistic Prediction of Pedestrian Future Movements. *International Conference on Robotics and Automation (ICRA)*, 2021.

4. Chapter 6 has appeared as: (Zhi et al., 2021a) **W. Zhi**, Tin Lai, L. Ott, F. Ramos. Trajectory Generation in New Environments from Past Experiences. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

5. Chapter 7 has appeared as: (Zhi et al., 2021c) **W. Zhi**, L. Ott, F. Ramos. Probabilistic Trajectory Prediction with Structural Constraints. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

6. Chapter 8 has appeared as: (Zhi et al., 2022b) **W. Zhi**, T. Lai, L. Ott, F. Ramos. Diffeomorphic Transforms for Generalised Imitation Learning. *Annual Learning for Dynamics and Control Conference (L4DC)*, 2022. **This work received the best paper award at L4DC**.

7. Chapter 9 has been accepted, and will appear as: (Zhi et al., 2023) **W. Zhi**, K. Van Wyk, I. Akinola, N. Ratliff, F. Ramos. Global and Reactive Motion Generation with Geometric Fabric Command Sequences. *IEEE International Conference on Robotics and Automation (ICRA)*, 2023. To appear

Additionally, the following peer-reviewed papers were produced during the author's PhD, but are not components of this thesis.

8. (Zhi et al., 2022a) **W. Zhi**, T. Lai, L. Ott, E. V. Bonilla, F. Ramos. Learning Efficient and Robust Ordinary Differential Equations via Invertible Neural Networks. *International Conference on Machine Learning (ICML)*, 2022.

9. (Zhi et al., 2019) **W. Zhi**, L. Ott, F. Ramos. Kernel Trajectory Maps for Multi-Modal Probabilistic Motion Prediction. *Conference on Robot Learning (CoRL)*, 2019.

10. (Lai et al., 2021) T. Lai, **W. Zhi**, T. Hermans, F. Ramos. Parallelised Diffeomorphic Sampling-based Motion Planning. *Conference on Robot Learning (CoRL)*, 2021.

In addition to the statements above, in cases where I am not the corresponding author of a published item, permission to include the published material has been granted by the corresponding author.

Weiming Zhi
January 2023

Abstract

Contemporary robots have become exceptionally skilled at achieving specific tasks in structured environments. However, they often fail when faced with the limitless permutations of real-world unstructured environments. This motivates robotics methods which *learn* from experience, rather than follow a pre-defined set of rules. In this thesis, we present a range of learning-based methods aimed at enabling robots, operating in dynamic and unstructured environments, to better understand their surroundings, anticipate the actions of others, and take informed actions accordingly.

In the first part of the thesis, we investigate methods which leverage learning to represent the structure and motion in a robot's operating environment, in a *continuous* manner. These methods do not impose a fixed-size grid on the environment, and can be queried at arbitrary resolutions. We present Fast Bayesian Hilbert Maps (Fast-BHMs), an efficient continuous representation of environment occupancy, and contribute a fusion algorithm to merge them in a multi-agent setting. We show that Fast-BHMs can be represented more succinctly than discretised grid-cells, motivating its use in multi-agent map-building. Then, we present a continuous and probabilistic spatiotemporal model of motion distributions in the environment. This allows a robot to understand long-term motion patterns in its surroundings, and reason about how things in this environment move.

Next, in the second part of this thesis, we develop methods for *anticipatory navigation*, where we aim to endow robots with the ability to predict the motion of dynamic obstacles in the vicinity, and to take decisions which account for these predictions. We contribute *Stochastic Process Anticipatory Navigation (SPAN)*, a framework that enables a robot to smoothly navigate through crowds. SPAN learns the future motion of nearby moving entities and represents the motions as stochastic processes, and integrates them into a control problem, which is then continuously solved. Then, we investigate the problem of whether we can transfer motion trajectories observed in past environments to novel environments.

Humans are able to infer likely motion patterns in an environment, by simply reasoning about the floor plan of the environment. We wish to endow robots with the same ability, and introduce *Occupancy-Conditional Trajectory Network (OTNet)*. OTNet embeds occupancy representations as a vector of similarities with previously observed maps to predict a multi-modal distribution over likely trajectories in the environment. This allows us to generate motion trajectories which match motion patterns observed in past environments. Finally, we contribute a combined learning and optimisation framework for probabilistic motion prediction. Motion prediction methods often lack mechanisms to explicitly account for the environment structure as constraints. Our framework enables constraints to be applied to the predictions, such that they remained compliant with the structure of the environment.

Finally, in the third part of this thesis, we examine methods to generate motions for robot manipulators in unstructured environments. We present the *Diffeomorphic Template (DT)* framework for generalised imitation learning. Generalised imitation learning seeks to teach a robot, by providing a small set of expert demonstrations, to acquire novel skills, and then generalise the skills when circumstances change. Individual DTs modularise the robot's motion, each imbuing a different behaviour, such as imitating demonstrations or avoiding new obstacles. Multiple DTs can then be combined to generate generalised behaviour. Next, we contribute the Geometric Fabrics Command Sequence (GFCS) method to generate motion trajectories which are both reactive and global. GFCS builds upon Geometric Fabrics, a reactive but local method. The local nature often results in the robot becoming stuck in local minima, unable to reach the designated goals. GFCS formulates an optimisation problem, over parameters of multiple Geometric Fabrics, which is subsequently solved via global optimisation. To speed up the optimisation, we learn a model that conditions on the problem to generate solutions to warm-start the optimiser.

Acknowledgements

This thesis was made possible by the kind support of many others, throughout this adventure.

Foremost, my gratitude goes to my supervisor Fabio Ramos. I would like to thank Fabio for not only providing me with valuable research guidance and unwavering support, but also imparting me with the knowledge of how to become a well-rounded researcher. I consider myself extremely fortunate to pursue my PhD in Fabio's lab. I am also immensely grateful to Lionel Ott, who though not listed on paper as a supervisor, acted very much as my secondary supervisor. He enlightened me with his depth of technical expertise and helped me develop foundational research skills. I would also like to express my gratitude to Edwin Bonilla, who also mentored me closely, tremendously honing my research abilities, and shaping my perspective on machine learning research.

My warmest thanks go to the many lab mates I met throughout my PhD. Many ideas came to fruition alongside coffee chats next to our beloved espresso machine. In particular, I would like to acknowledge the oversized impacts Ransalu Senanayake and Tin lai have played in my PhD journey. As a senior PhD student, Ransalu took me under his wing from the first day of my PhD, and walked me through the intricacies of robotics research. Tin is my serial co-author and force multiplier, who never fails to bring fresh perspectives and spark new research. I deeply enjoyed my time remotely interning with NVIDIA's robotics research lab, led by Dieter Fox. Along with Fabio, I would like to thank my mentors at NVIDIA — Karl Van Wyk, Iretiayo Akinola and Nathan Ratliff. I was also greatly fortunate to collaborate with Tucker Hermans on sampling-based motion planning research.

I am also grateful to Nikita for her indispensable support and patience, and also for keeping me sane during Covid-19 lockdowns. Last but not least, none of this would be possible without the support of my parents and siblings, who have made sacrifices for me to receive the highest quality education, whenever possible — they have my sincerest gratitude.

Nomenclature

General notation

$f(\cdot)$, $g(\cdot)$, $h(\cdot)$	Functions
i, j, k	Indices
$\approx$	Approximately equal to
$>>$	Much greater than
$\|\|\cdot\|\|_p$	p-norm
$\|\cdot\|$	Absolute value of a scalar or cardinality of a set
$\{\cdot\}$	A set
$\min_x(\cdot), \max_x(\cdot)$	Minimise, maximise with respect to x
$\arg\min_x(\cdot), \arg\max_x(\cdot)$	Minimise, maximise with respect to x and return the argument
$\mathbb{1}$	Indicator function
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^{m\times n}$	Set of m by n matrices over the reals
$\mathbb{R}^{m\times m}_{++}$	Set of m by m positive definite matrices over the reals
$\mathbb{N}$	Set of natural numbers

Linear algebra

x	Scalar
$\mathbf{x}$	Vector
$\mathbf{x}^\top$	Transpose of $\mathbf{x}$
$\mathbf{X}$	Matrix
$\mathbf{X}^{-1}$	Inverse of matrix $\mathbf{X}$

$\lvert\mathbf{X}\rvert$	Determinant of matrix $\mathbf{X}$
$\mathbf{X}^{\dagger}$	Generalised inverse matrix $\mathbf{X}$
$\mathbf{I}$	Identity matrix
$\mathrm{vec}(\mathbf{X})$	Vectorise matrix $\mathbf{X}$
$\mathrm{tr}(\mathbf{X})$	Trace of matrix $\mathbf{X}$
$\mathrm{diag}(\mathbf{X})$	Diagonal of matrix $\mathbf{X}$
$\mathrm{LowerTrig}(\mathbf{X})$	Lower triangular matrix of matrix $\mathbf{X}$
Probability theory	
$p(A)$	Probability of event A
$p(A, B)$, $p(A \bigcap B)$	Probability of events A and B
$\boldsymbol{\mu}$, $\mathbf{m}$	Mean vector
$\boldsymbol{\Sigma}$	Covariance matrix
$\sim$	Distributed as
$\mathcal{N}$	Normal distribution
$\mathcal{U}$	Uniform distribution
MN	Matrix normal distribution
VM	von-Mises distribution
erf	Error function
&	Conflation
$D_{KL}(\cdot\|\|\cdot)$	Kullback–Leibler divergence
i.i.d	Independent and identically distributed
$\mathbb{E}$	Expected value
ε	Noise
Machine learning	
$\mathbf{x}$	Vector of inputs
y	Outputs
$\mathbf{x}^*$	Test point of interest
$\mathcal{X}$	Instance space

$\mathcal{Y}$	Label space
D	Dataset
$\mathbf{w}$, $\mathbf{W}$	Weight vector, weight matrix
$\boldsymbol{\theta}$	Parameters
$\mathcal{L}$	Loss function
$\sigma(\cdot)$	Sigmoid function
$\boldsymbol{\phi}$, $\boldsymbol{\Phi}$	Feature vector, feature matrix
$\hat{\mathbf{x}}$, $\hat{t}$, $\hat{\mathbf{s}}$	Inducing points
k	Kernel function
δ_H	Hausdorff distance
$\mathcal{L}$	Loss function
γ, ℓ	Length-scale hyper-parameter
λ	Regularisation hyper-parameter
Motion trajectories and control	
$\mathcal{Q}$	Configuration space
$\mathbf{q}$	Configuration-space coordinates
$\dot{\mathbf{q}}$	Configuration-space velocity
$\ddot{\mathbf{q}}$	Configuration-space acceleration
$\mathbf{q}_g$	Goal configuration
$\mathbf{x}$	Task-space positions
$\mathbf{u}$	Controls
f_e	Forward kinematics
$\mathbf{J}_f$	Jacobian of mapping f
ξ	Trajectories
$\boldsymbol{\Xi}$	Distributions of trajectories
$\mathbf{M}$	Manifold
φ	Diffeomorphism
γ	Flow of ODE

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

Robots have become exceedingly adept at completing specific tasks in controlled conditions, such as in warehouses or manufacturing plants. The introduction of autonomous robots in these limited environments have reduced human fatigue, mitigated safety hazards, and improved execution precision. The increasing visibility of a diverse range of robots give the general public a perception that the ubiquitous deployment of general-purpose robots, capable of operating for long durations in the wild, is just around the corner. The reality is far more sobering – modern robots are often delicate, vulnerable to unexpected situations, and confined to performing tasks in a narrow domain. In its essence, existing robot systems lack the remarkably human-esque ability to understand the context of the task at hand in its entirety and extrapolate from prior experience to acquire new skills.

A colossal bounty is placed on solving the open problem of how to develop robotic systems that are more multi-purpose and can conduct a variety of tasks safely in the unstructured world. Solving this challenge would bring about profound changes to human society, paving the way for a range of emerging robotics applications in the service, transport and construction sectors. Envision a future where robots are capable of safely coexisting and collaborating with humans in everyday environments – helping in the kitchen, walking the dog in a park, delivering parcels through crowds. These technological advancements have the potential to release a torrent of productivity, and provide a strong response to persistent macroeconomic woes brought about by aging populations.

Operating in varied and dynamics environments, with minimal human intervention, re-

(a) Robot Manipulator (b) Autonomous vehicle (c) Quadrotor drone (d) Quadruped robot

Figure 1.1: Contemporary robots deployed in their respective environments. Courtesy of Wikimedia Commons.

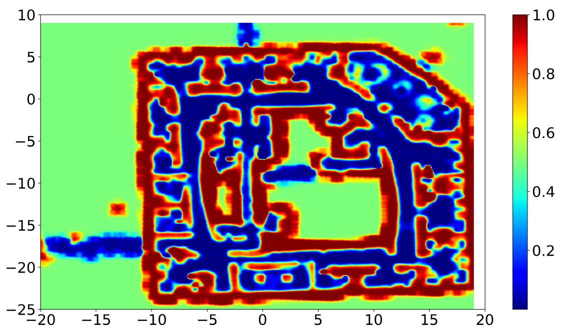
quires robots to handle changes in its vicinity in a natural and predictable manner. At the same time, the unstructured nature of real-world conditions can present the robot with endless permutations of scenarios to handle. In many cases, it becomes infeasible to hand-specify an exhaustive set of predefined rules to capture the variety of conditions which may arise. Advances in the field of machine learning have provided roboticists with novel approaches to leverage the data for generalisation. This thesis presents contributions developed by the author to endow robots with the ability to extract patterns from data, and learn to adapt to its environment from experience. Machine learning can be applied to a diverse range of components throughout the autonomy stack, allowing the robot to learn both *about* and *from* its surroundings. In particular, we explore ideas and develop methods (1) to aid robots in constructing concise representations of its surroundings in multi-agent and dynamic settings; (2) to probabilistically predict the movement patterns of dynamic agents in the environment and act according to the predictions; (3) to learn to generate motion for robot manipulators in complex environments.

1.2 Contributions

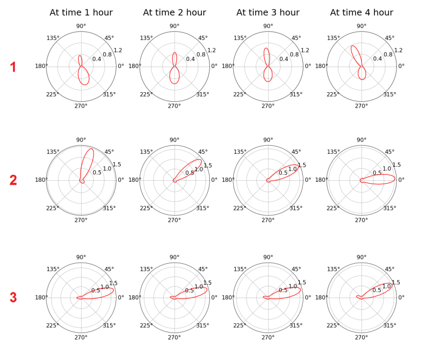
Under the overarching goal of integrating machine learning in robots to perform more general tasks in unstructured environments, we develop robot learning methods to represent the environment, anticipate the movement of others in the vicinity, and generate informed motions. Specifically, in each of the three parts of this thesis, we tackle a distinct research theme and make several contributions within each theme.

Highlight figures of Part I: Learning Continuous Environment Representation

Chapter 3: Fusion of Continuous Occupancy Maps



Chapter 4: Continuous Spatiotemporal Maps of Motion Directions



1. **Part I: Learning Continuous Environment Representations:** We seek to apply learning to aid robots in representing their perceived surroundings. We develop efficient methods to learn models which represent structure and motion patterns within robots' environments in a continuous manner, particularly in a multi-agent setup or in a dynamic environment. Early methods to represent robots' operating environment sought to enforce a grid structure and compute values for each grid cell. We take an alternative approach, and work with learning-based approaches for robots to continuously represent their surroundings without discretising the inherently continuous real-world. Specifically, we are interested in answering the following question:

   How do we leverage learning to produce compact representations of occupancy and movement in a multi-agent set-up or a dynamic environment?

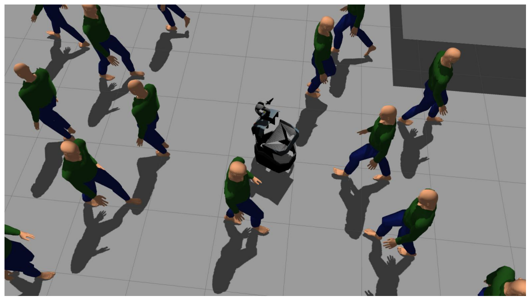
   The specific contributions include:

   - **Fusion of continuous occupancy representations:** In chapter 3, we introduce the *Fast-BHM* method, an efficient variant of *Bayesian Hilbert Maps* (BHMs) (Senanayake and Ramos, 2017), to build maps which are continuous and compact. We show that the compact nature of Fast-BHMs makes them ideal to use in bandwidth-limited multi-agent setups. To this end, we develop a fusion algorithm to merge multiple fast-BHMs models maps iteratively, enabling a group of robots to build separate maps, which are then fused in a decentralised manner into an aggregated model.

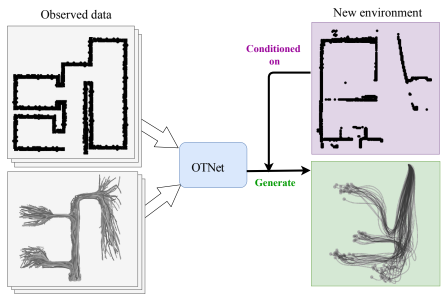
- **Continuous spatiotemporal model of direction distributions:** In chapter 4, we introduce a continuous spatiotemporal model to represent the conditional probability distribution of movement directions of dynamic objects. This distribution is capable of capturing multi-modality and is correctly wrapped around a unit circle. The contributed model allows the robot to understand long-term patterns of how dynamic objects move in the environment.

Highlight figures of Part II: Learning for Anticipatory Navigation

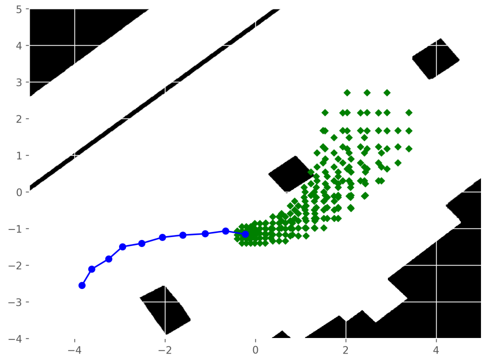
Chapter 5: Stochastic Process Anticipatory Navigation



Chapter 6: Trajectory Generation in New Environments from Past Experiences



Chapter 7: Structurally Constrained Motion Prediction



2. **Part II: Learning for Anticipatory Navigation:** Humans are able to anticipate how others in the vicinity move, and act accordingly to negotiate a path through dynamic environments – we seek to enable robots to do the same. We study the problem of navigating through environments with crowds while anticipating how the pedestrians in the environment move. We are interested in answering the questions:

How do we learn a probabilistic predictive model of dynamic objects, such as pedestrians, from data and past experience? Additionally, how can we take these predictions into account and anticipate the motion of others during navigation?

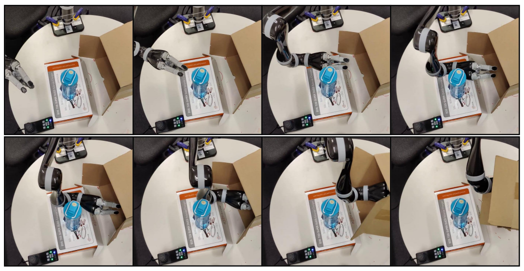
The specific contributions include:

- **Stochastic Process Anticipatory Navigation:** In chapter 5, we introduce the *Stochastic Process Anticipatory Navigation* (SPAN) framework. A probabilistic predictive model produces distributions of where dynamic pedestrians are expected to move. This predictive model is integrated into a time-to-collision controller to smoothly navigate through moving crowds. The robot is able to actively predict and preempt the behaviours of other agents in its proximity. We observe the emergence of behaviours, such as the controlled robot following behind others that are moving in the same direction, when squeezing through crowds.
- **Occupancy-Conditioned Trajectory Networks:** In chapter 6, we introduce *Occupancy-Conditioned Trajectory Network* (OTNet), a model to generalise motion patterns observed in past environments to new environments. Humans have the ability to intuit probable motion trajectories in an environment simply by examining the environment structure from a floor plan. OTNet aims to use machine learning to allow robots to do the same. OTNet encodes the environment occupancy as a vector of similarities between an environment of interest and previously observed environments, and trains a neural network to map these embedding vectors to multi-modal probability distributions over trajectories. The trajectory distributions can be further refined by enforcing the start points of the trajectories. We envision that the trajectory distribution over the entire environment can be used as a prior for trajectory prediction models, which can then be refined to accurately extrapolate partially observed trajectories.
- **Probabilistic motion prediction model with structural constraints:** In chapter 7, we introduce a method to predict motion trajectories capable of imposing structural constraints to multi-modal distributions over trajectories. This enables the probabilistic motion prediction model to enforce known constraints in the environment. For example, we know that moving objects in the environment are highly unlikely to go through occupied coordinates, so we explicitly construct and subsequently enforce chance-constraints on the predicted distribution of fu-

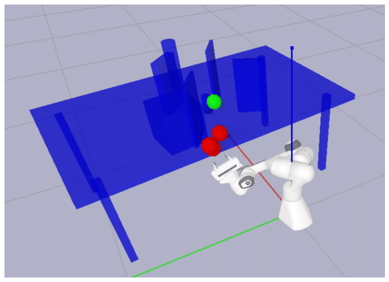
ture trajectories, such that it complies with the known occupancy structure of the environment.

Highlight figures of Part III: Learning for Robot Manipulator Motion Generation

Chapter 8: Diffeomorphic Transforms for Generalised Imitation Learning



Chapter 9: Motion Generation with Geometric Fabric Command Sequences



3. **Part III: Learning for Robot Manipulator Motion Generation:** We study robot learning methods to generate motion trajectories for manipulators. Compared to ground robots, robot manipulators often have higher degrees of freedom, resulting in higher dimensional motion trajectories. However, they are typically not subject to non-holonomic constraints typical of ground robots. In part III, we seek to enable robot manipulators to operate in dynamic environments around humans. Specifically, we seek answers to the questions:

   How can we enable robots to (i) learn from demonstrations to acquire new skills and generalise these skills when the environment or problem setting changes, and (ii) efficiently produce reactive motions that are also globally optimal?

   The specific contributed deliverables include:

   - **Diffeomorphic Templates for generalised imitation learning:** In chapter 8, we introduce *Diffeomorphic Templates* for generalised imitation learning. Oftentimes, it is difficult to specify the exact skill that the robot is expected to perform. In these setups, we aim to provide a framework for humans to describe the desired motion by providing a small number of demonstrations for the robot to learn from. Additionally, we require the robot to *generalise* and account for changes in the environment or other additional user specifications, which may not be present in the demonstration data. Our Diffeomorphic Templates framework decomposes each specific behaviour of the motion, such as imitating data, or avoiding new obstacles, as individual modules. These modules can then be combined in a principled manner to produce stable dynamical systems representing the desired combined motion.

- **Geometric Fabric Command Sequences:** In chapter 9, we introduce *Geometric Fabric Command Sequences (GFCS)*, a method capable of generating collision-free and kinematically feasible robot motion from start to goal. The produced motion is both reactive and globally optimal. The proposed GFCS extends upon, and solves an important limitation of, *Geometric Fabrics* (Van Wyk et al., 2022). Geometric Fabrics can reactively generate smooth motion, but the generated motion trajectories often suffer from a tendency of becoming trapped in local non-convex regions, and are unable to reach the goals. We formulate global motion generation as a global optimisation problem over the parameters of several Geometric Fabric models, which is then solved by a black-box optimiser. We additionally develop a self-supervised learning framework to speed up the optimisation by learning a generative model to warm start the black-box optimisation.

1.3 Thesis Structure

We begin in **chapter 2** by providing background knowledge on relevant topics, and describe some of the frequently-used methods which appear in subsequent contribution chapters. These topics span both machine learning and robotics, including supervised regression with linear regression with non-linear features and neural networks, optimisation, environment representations, robot kinematics, and dynamical systems.

Next, in **part I**, we detail contributions to environment representation via learning. In **chapter 3**, we introduce approaches to fuse *Fast Bayesian Hilbert Maps*, a continuous occupancy representation. In **chapter 4**, we propose a continuous spatiotemporal model to capture the distributions of movement directions in an environment.

Subsequently, in **part II**, we describe contributions to learning for anticipation navigation. In **chapter 5**, we present the *Stochastic Process Anticipatory Navigation* (SPAN) framework, to predict the movement of other agents and use these predictions when navigating through crowds. In **chapter 6**, we introduce *Occupancy-conditioned Trajectory Network*

(OTNet) a model capable of conditioning on an environment's occupancy to generate the likely motion patterns in the environment; in **chapter 7**, we outline a combined learning and optimisation framework, which enforce constraints on probabilistic motion prediction models.

Then, in **part III**, we elaborate on contributions to generating motion for robot manipulators. In **chapter 8**, we propose the *Diffeomorphic Templates* (DTs) framework for generalised imitation learning, which enables robots to learn from expert demonstrations and to generalise acquired behaviour when the problem setup changes. In **chapter 9**, we introduce *Geometric Fabric Command Sequence* (GFCS), along with a self-supervised learning framework to integrate GFCS, to produce manipulator motion which is both reactive and global.

Finally, in **chapter 10**, we provide concluding remarks for this thesis, revisit connections between the presented themes, and outline possible future research directions.

Chapter 2

Background

2.1 Overview

In this chapter, we shall provide background to tools used to develop the robot learning methods introduced in the later chapters. We begin by laying out the background to some of the topics relevant to machine learning. First, in section 2.2, we describe supervised regression problems, which are some of the most commonly encountered machine learning problems. To tackle these regression problems, in section 2.2.1, we outline the background around fitting linear models, which can be used in conjunction with non-linear features. These models are easy to implement and interpret. We give an illustrative example of non-linear features in section 2.2.2, and show how kernel functions can be used to greatly reduce computation. However, linear models, even when applied with non-linear features, are often limited in their flexibility. Hence, in section 2.2.3, we discuss fully-connected neural networks, the quintessential neural network model. These flexible models can be highly parameterised, while also trained efficiently on modern GPUs. Both linear models and neural network models have been used throughout this thesis to approximate functions of interest. Then in section 2.3, we sketch the background for some of the optimisation techniques used in this thesis. This includes *ADAptive Moment estimation* (ADAM) to estimate parameters of neural network models, *Sequential Quadratic Programming* (SQP) to solve constrained optimisation problems, where first and second derivatives are available, and *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES), a black-box optimiser which makes little assumptions about the function that is optimised, and does not require function derivatives to find globally optimal solutions.

Then, we provide context to the relevant topics from robotics. We begin, in section 2.4, by providing background on methods to represent a robot's environment. We discuss occupancy grid maps, an early model to discretely represent the environment of a robot, and outline how the grid values can be updated using Baye's rule. We also introduce Hilbert Maps, a learning-based approach that produces continuous representations of the environment. Next, in section 2.5, we provide a brief overview of the kinematics of robot manipulators, and explain the various concepts, such as *configuration space*, *task space* and *forward kinematics*. We then give an example of how to derive the forward kinematics for a simple planar robot. Finally, in section 2.6, we give a gentle background to the concepts around dynamical systems and ODEs which are relevant to our contributions. Robot motion is often described as a dynamical system and individual robot motion trajectories are integrals or roll-outs of the system. Additionally, we also outline the concept of asymptotic stability for dynamical systems. This is revisited in chapter 8, where we tackle generalised imitation learning while maintaining stability.

2.2 Supervised Regression Problems

Many robot learning problems that arise in this thesis fall under the general umbrella of supervised learning. In particular, the supervised regression problems that appear in this thesis include the problem of finding continuous trajectories in chapter 5 and chapter 7, as well as that of learning from expert demonstrations in chapter 8.

Consider a dataset containing N instances of input-output pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Here, we assume that the inputs $\mathbf{x}$ are d-dimensional vectors from a set $\mathcal{X} \subseteq \mathbb{R}^d$. We shall, in particular, focus on regression problems, where the outputs y are real-valued scalars from a set $\mathcal{Y} \subseteq \mathbb{R}$. Though the presented methods can be easily generalised to multi-variate outputs. We assume that observed instances are independently drawn from a joint distribution $p(\mathbf{x}, y)$ over $\mathcal{X}$ and $\mathcal{Y}$. Additionally, the observed y_i are outputs of some unknown target function on the inputs, corrupted by zero-mean Gaussian noise with constant variance σ^2, i.e.

$$y_i = g(\mathbf{x}_i) + \varepsilon \qquad\qquad \text{where } \varepsilon \sim N(0, \sigma^2). \tag{2.1}$$

We refer to the inputs, $\mathbf{x}_i \in \mathcal{X}$, as *features* and the outputs, $y_i \in \mathbb{R}$ as *labels.* Our aim is to approximate the target function $g : \mathcal{X} \to \mathcal{Y}$ as accurately as possible with another function

f. We refer to f as a function approximator.

2.2.1 Linear Regression Models

Here we study linear regression, one of the simplest parametric approaches to tackle supervised learning. We begin by making assumptions about the set of trial functions to search in, and specify a parametric form for the unknown function. Take the linear case: if we assume that g is linear and restrict our approximation to be likewise linear, omitting the intercept for brevity, our approximation takes the form:

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}, \tag{2.2}$$

where vector $\mathbf{w} \in \mathbb{R}^d$ contains the weight parameters of our model. Fitting a linear model on given inputs under the regression problem setup is referred to as *linear regression.* More generally, we can introduce non-linearities to our modelling by adopting a set of k non-linear basis functions (also known as feature maps), $\{\phi_i : \mathbb{R}^d \to \mathbb{R}\}_{i=1}^k$, and typically $k > d$. We define $\boldsymbol{\phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), \ldots, \phi_k(\mathbf{x})]^\top$ as our transformed set of features, and our model is now:

$$f(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}), \tag{2.3}$$

where the vector of parameters is now $\mathbf{w} \in \mathbb{R}^k$. Note that although the function f is no longer linear with respect to the original inputs $\mathbf{x}$, it remains linear with respect to the transformed features $\boldsymbol{\phi}(\mathbf{x})$. Therefore, we can simply view this as applying linear regression to the new set of transformed features.

After assuming a suitable parametric form of our approximator f, our task is now to find the set of parameters $\mathbf{w}$ such that f is as "close" as possible to g. To accomplish this, we need to define a *loss function* to measure the ”closeness” of our model prediction $f(\mathbf{x})$, when the label is y. A commonly used loss function used in regression problems is the square error loss:

$$\mathcal{L}(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2 \tag{2.4}$$

The best-fit function approximator is then found by optimising the parameters $\mathbf{w} \in \mathbb{R}^k$ such that the expected value of the loss function is minimised. This expected value cannot be directly computed, as the joint distribution $p(\mathbf{x}, y)$ is unknown. However, we can compute an approximation, or the *empirical risk*, by taking the average loss over the training dataset.

Specifically,

$$
\begin{aligned}
\mathbf{w}^* &= \arg\min_{\mathbf{w}\in\mathbb{R}^k} \mathbb{E}_{p(\mathbf{x},y)}[\mathcal{L}(f(\mathbf{x}), y)] && (2.5)\\
&\approx \arg\min_{\mathbf{w}\in\mathbb{R}^k} \frac{1}{N}\sum_{i=1}^{N}(f(\mathbf{x}_i) - y_i)^2, && (2.6)\\
\text{and } & f(\mathbf{x}_i) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_i), && (2.7)
\end{aligned}
$$

where the expected value of the loss, over the joint distribution $p(\mathbf{x}, y)$, is empirically approximated by the i.i.d samples contained in the dataset $\mathcal{D}$. We refer to the computed loss as the *mean squared error* (MSE) loss. We can then proceed to solve eq. (2.7) to find the optimal parameters $\mathbf{w}^*$. In the case of linear models with a MSE loss, the solution admits a closed-form:

$$
\mathbf{w}^* = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{y}, \qquad \text{where } \boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(\mathbf{x}_1) \\ \vdots \\ \boldsymbol{\phi}(\mathbf{x}_N) \end{bmatrix} \qquad \text{and } \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}. \qquad (2.8)
$$

This allows us to directly efficiently obtain the optimised weights, without resorting to numerical optimisation approaches.

2.2.2 Non-linear Features and Kernel Functions: An Example

As outlined in the previous subsection section 2.2.1, non-linear features extend linear regression models to estimate non-linear functions from inputs to targets, by first mapping the original d-dimensional inputs to k-dimensional non-linear features with a feature map, $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^k$, and then applying linear methods on the features. Intuitively, the unknown function may be linear in the generally higher dimensional *feature space*, while being non-linear with respect to the original inputs. Here, we show an illustrative example of regression with polynomial features, that are explicitly computed. Then, we shall discuss the concept of "kernelisation", which speeds up computations by avoiding explicitly computing inner products on features.

Here, we consider an example non-linear regression problem from Baddoo et al. (2022). Suppose we have 3-dimensional inputs $\mathbf{x} \in \mathbb{R}^3$ and aim to find a function approximator f

which maps these inputs to targets. Suppose we know *a priori*, from physical intuition or empirical evidence, that the set of pair-wise products spans the space of candidate functions – our function approximator can be expressed as a linear combination of pair-wise quadratic basis functions,

$$\boldsymbol{\phi}(\mathbf{x}) = [x_1x_2, x_1x_3, x_2x_3, x_1^2, x_2^2, x_3^2]^\top \in \mathbb{R}^6, \qquad f(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}), \tag{2.9}$$

where $\mathbf{w} \in \mathbb{R}^6$ are weights which can be computed via eq. (2.8). We observe that solving for the weights requires products between the features. If we explicitly construct features from two inputs $\mathbf{x}, \mathbf{x}'$, and compute their dot product, $\boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}')$, the number of operations required is 23. We need to perform 6 products to explicitly construct each feature vector, and then 6 products and 5 sums to compute the dot product. This computation can be sped up by constructing an equivalent *kernel function*.

Without loss of generality, we consider a slightly re-scaled feature map for a simpler reduction of coefficients,

$$\boldsymbol{\varphi}(\mathbf{x}) = [\sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3, x_1^2, x_2^2, x_3^2]^\top \in \mathbb{R}^6. \tag{2.10}$$

The dot product between two the feature vectors of two inputs $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^3$ is then,

$$\begin{aligned}
\boldsymbol{\varphi}(\mathbf{x})^\top \boldsymbol{\varphi}(\mathbf{x}') &= 2x_1x_2x_1'x_2' + 2x_1x_3x_1'x_3' + 2x_2x_3x_2'x_3' + x_1^2(x_1')^2 + x_2^2(x_2')^2 + x_3^2(x_3')^2 && (2.11)\\
&= (x_1x_1' + x_2x_2' + x_3x_3')^2 && (2.12)\\
&= (\mathbf{x}^\top \mathbf{x}')^2 := k(\mathbf{x}, \mathbf{x}'). && (2.13)
\end{aligned}$$

Lo and behold, there is an alternative route to evaluating the dot product between the feature vectors of two inputs! We can instead evaluate the defined kernel function $k : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$, which requires only 6 operations – 3 products and 2 sums for the dot product, and 1 for the square. The computational savings of evaluating the kernel function instead of explicitly evaluating feature vectors is even more apparent when the inputs are higher dimensional or the interaction terms of higher dimensions are modelled. Kernel functions have been designed for a variety of common feature maps, and rules to combine kernel functions have also been explored. More details on kernels and kernel designing can be found in Chapter 2 of Duvenaud (2014).

2.2.3 Regression with Fully-Connected Neural Networks

Linear models on non-linear features have been used extensively throughout this thesis, e.g. to construct continuous environment representations (chapters 3 and 4), continuous trajectory representations (chapters 5 and 7), and invertible functions (chapter 8). They are straightforward to apply and easy to interpret, when we have *a priori* knowledge of what basis functions to select. However, the true workhorses of contemporary machine learning are neural network models. Neural networks have been used in each of the chapters between chapter 4 to chapter 9. In this section, we shall introduce fully-connected neural networks, often also referred to as "feedforward neural networks" or "dense neural networks", which are the quintessential neural network models.

Consider the regression problem outlined in section 2.2. Instead of assuming that the function approximator f is linear, we instead assume that f is a neural network given by a composition of q layers, i.e.,

$$f(\mathbf{x}) = h^q(\ldots h^2(h^1(\mathbf{x}))), \tag{2.14}$$

where q is known as the *depth* of network, and each h^i, $i = 1, \ldots, q$ is known as a fully-connected layer. Each layer itself is a linear function, with a weight matrix $\mathbf{W}^i$ and bias $\mathbf{b}^i$, along with an activation function φ^i which may be non-linear, that is:

$$\mathbf{u}^{i+1} = h^i(\mathbf{u}^i) = \varphi^i({\mathbf{W}^i}^\top \mathbf{u}^i + \mathbf{b}^i), \tag{2.15}$$

where $\mathbf{u}^i$ is the input from the previous layer and $\mathbf{u}^{i+1}$ is the output of the layer. Specifically, the input of the first layer h^1 is $\mathbf{x}$, and the inputs of the remaining layers are the outputs of the previous layer. Typically, in regression problems, the activation function at the final layer, φ^q, is simply the identity function, while the other activation functions are non-linear. Various non-linear functions can be used as activation functions including the $ReLU$, $Tanh$, and $Softmax$ functions (Goodfellow et al., 2016).

Unlike linear models under an MSE loss scheme, the parameters of the best fit neural network function cannot be expressed in closed form, and instead requires numerical optimisation. Let us denote the neural network function approximator as $f_{\boldsymbol{\theta}}$, where $\boldsymbol{\theta} = \{\mathbf{W}^i, \mathbf{b}^i\}_{i=1}^q$ is the set of all parameters of the network. We apply first-order gradient descent methods, such as Stochastic Gradient Descent (SGD), where the most basic update rule for the parameters, with respect to a loss $\mathcal{L}$, is:

$$\boldsymbol{\theta}^{j+1} \leftarrow \boldsymbol{\theta}^{j} - \alpha \nabla_{\boldsymbol{\theta}} \frac{1}{|I^j|} \sum_{i \in I^j} \mathcal{L}(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i) \tag{2.16}$$

where j denotes the current iterations, and α is the step-size. It is often prohibitively expensive to compute the derivatives of the parameters with respect to the loss over all of the examples. Therefore, we estimate this derivative by randomly selecting a subset of examples from the dataset. The set I^j contains a batch of randomly selected indices of the dataset, drawn every iteration. The number of corresponding data points selected in each batch is in practice significantly smaller than the entire dataset, with the batch known as a *mini-batch*. The parameters are updated iteratively, with the initial values of the parameters $\boldsymbol{\theta}^0$ drawn from a random distribution. Selecting a fit-for-purpose optimiser impacts the performance of the network. To this end, various newer and more sophisticated optimisers have been developed upon SGD, such as ADAM (Kingma and Ba, 2015).

We observe that a challenge of utilising neural networks is efficiently obtaining derivatives of the loss, with respect to network parameters, over a sizeable batch of data examples. Indeed, the adoption of neural networks is driven by the availability of efficient automatic differentiation libraries such as *PyTorch* and (Paszke et al., 2019) and *Tensorflow* (Abadi et al., 2016), which can be parallelised on modern GPUs. These developments have given rise to neural network models which are increasingly deep, and allow for training on datasets which are increasingly large in size.

2.3 Optimisation

Beyond using gradient descent to train neural networks, many methods developed in this thesis make use of optimisation techniques. In this section, we provide an introduction to the optimisation methods used. Optimisation seeks to optimise (typically minimise) an *objective function* with respect to a set of *decision variables*, while satisfying a set of *constraints*. We can get a general sense of the difficulty of the optimisation problem by the global structure (convex vs non-convex) and local structure (derivatives available vs derivatives not available) of the objective and constraints. In this thesis, we use (1) ADAptive Moment estimation (ADAM) a variant of Stochastic Gradient Descent (SGD) (discussed in section 2.2.3) to optimise the parameters of neural network models, (2) Sequential Least

SQuares Programming (SLSQP) Kraft (1988), an implementation of Sequential Quadratic Programming (SQP) in chapter 7, and (3) the derivative-free Covariance Matrix Adaptation Evolution Strategy (CMA-ES) Hansen (2016) in chapter 9. We shall provide background for these approaches.

2.3.1 ADAptive Moment estimation (ADAM)

Parameters of neural networks in this thesis are typically optimised with ADAptive Moment estimation (ADAM) (Kingma and Ba, 2015), a first-order optimiser which extends Stochastic Gradient Descent (SGD). Vanilla SGD has been shown to converge much more efficiently and better escape local minima when *momentum* is considered (Qian, 1999). That is, when we update the parameters in the current gradient descent iteration, the updates in the previous iterations are also taken into account. A line of work, begin from *AdaGrad* (Duchi et al., 2011), *RMSProp* (Tieleman et al., 2012), to ADAM makes use of momentum. Here, we provide background on ADAM, which has become the go-to optimiser for modern neural network models.

When optimising a set of d-dimensional parameters, ADAM keeps track of two properties of the optimiser at its current iteration: (1) the first moment, $\mathbf{m} \in \mathbb{R}^d$: the exponential moving average of gradients; (2) the second moment, $\mathbf{v} \in \mathbb{R}^d$: the exponential moving average of the gradients square of gradients. Assuming that we are optimising a loss $\mathcal{L}$ with model parameters θ and the decay rates for the exponential moving average are given hyper-parameters, $\beta_1, \beta_2 \in [0, 1)$, for the first and second moments respectively. Then, the update rules for the loss gradients and moments at iteration $j + 1$ are:

$$\mathbf{g}^{j+1} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^j) \tag{2.17}$$

$$\mathbf{m}^{j+1} \leftarrow \beta_1 \mathbf{m}^j + (1 - \beta_1)\mathbf{g}^{j+1} \tag{2.18}$$

$$\mathbf{v}^{j+1} \leftarrow \beta_2 \mathbf{v}^j + (1 - \beta_2)(\mathbf{g}^{j+1} \odot \mathbf{g}^{j+1}), \tag{2.19}$$

where $\odot$ refers to the Hadamard (element-wise) product, and the initial first and second moments are set as vectors of 0. To account for the bias towards zeros of the computed moments, ADAM performs bias-corrections for the moments:

$$\widehat{\mathbf{m}}^{j+1} \leftarrow \frac{\widehat{\mathbf{m}}^{j+1}}{(1 - (\beta_1)^{j+1})}, \qquad \widehat{\mathbf{v}}^{j+1} \leftarrow \frac{\widehat{\mathbf{v}}^{j+1}}{(1 - (\beta_2)^{j+1})}. \tag{2.20}$$

Here, $(\beta_1)^{j+1}$ and $(\beta_1)^{j+1}$ refers to the decay parameters to the power of $j + 1$. We can then use the corrected moments to update our decision variables (i.e. the neural network parameters),

$$\boldsymbol{\theta}^{j+1} \leftarrow \boldsymbol{\theta}^{j} - \alpha \frac{\widehat{\mathbf{m}}^{j+1}}{\sqrt{\widehat{\mathbf{v}}^{j+1}} + \epsilon}, \tag{2.21}$$

where α is the learning rate, ϵ is a small value (typically 10^{-8}) to prevent divide by zero errors, and $\sqrt{\widehat{\mathbf{v}}^{j+1}}$ refers to the element square-root of $\widehat{\mathbf{v}}^{j+1}$. In the next subsection, we shall provide background on optimisers which are used beyond optimising for parameters for learning models.

2.3.2 Sequential Quadratic Programming

Sequential quadratic programming (SQP) can be applied to constrained optimisation problems, where the objective and constraints are twice continuously differentiable. SQP is an iterative method which begins at a random guess as the solution, and sequentially solves *quadratic programs*, one at each iteration to refine the solution. A quadratic program (QP) has a quadratic objective function with linear constraints and can be efficiently solved to optimality (Nocedal and Wright, 2006, Chap 16). Intuitively, SQP resembles applying Newton's method to the objective function, while accounting for the constraints. When SQP is applied to an unconstrained problem, SQP reduces to Newton's method, as solving an unconstrained quadratic program amounts to finding the minimum of a convex quadratic function. SQP will converge on a locally-optimal solution, and can be used in conjunction with global search meta-heuristics to escape local minima and find more global solutions. Consider a canonical constrained optimisation problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) \tag{2.22}$$

$$\text{subject to: } g(\mathbf{x}) = 0, \tag{2.23}$$

$$h(\mathbf{x}) \geq 0. \tag{2.24}$$

where $\mathbf{x} \in \mathbb{R}^d$ denotes the decision variables, $f : \mathbb{R}^d \to \mathbb{R}$ is the objective function, $g : \mathbb{R}^d \to \mathbb{R}^p$ gives p equality constraints and $h : \mathbb{R}^d \to \mathbb{R}^q$ gives q inequality constraints. The Lagrangian of this problem is given by:

$$L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\mu}^\top g(\mathbf{x}) + \boldsymbol{\lambda}^\top h(\mathbf{x}), \tag{2.25}$$

where $\boldsymbol{\mu}, \boldsymbol{\lambda}$ are Lagrangian multipliers. We construct a quadratic program by using a quadratic approximation of the objective and linear approximations of the constraints, to give the increment step for the current iteration of the solution. For the k^{th} iterate $\mathbf{x}_k$, we seek to find the current increment step $\mathbf{d}_k$ via the QP:

$$\min_{\mathbf{d}_k} \quad f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \mathbf{d}_k + \frac{1}{2}\mathbf{d}_k^\top \nabla^2_{\mathbf{x}_k\mathbf{x}_k} L \mathbf{d}_k, \tag{2.26}$$

$$\text{s.t.} \quad g(\mathbf{x}_k) + \nabla g(\mathbf{x}_k)^\top \mathbf{d}_k = 0, \tag{2.27}$$

$$h(\mathbf{x}_k) + \nabla h(\mathbf{x}_k)^\top \mathbf{d}_k \geq 0. \tag{2.28}$$

The QP is then solved and the solution is updated for the next iterate, $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{d}_k$, and the Lagrangian multipliers are updated by recomputing stationary points of the Lagrangian. We begin by providing a random initialisation of the solution, $\mathbf{x}_0$, and iteratively formulate and solve QPs to update the solution until we converge onto a solution. Convergence analysis and practical implementation tips are provided in (Nocedal and Wright, 2006, Chap. 18).

2.3.3 Covariance Matrix Adaptation Evolution Strategy

Sequential quadratic programming requires the objective function to be twice differentiable. However, in many problems, derivatives do not exist or are not easily available. Here, we provide background for the derivative-free method Covariance Matrix Adaptation Evolution Strategy (CMA-ES), which in practice is capable of robustly handling objective functions which are non-smooth, noisy, and contain multiple local optima (Hansen et al., 2010). CMA-ES is often used in scenarios where each function evaluation is slow or expensive, such as optimising model hyper-parameters or shape optimisation of airfoils. We assume that we are faced with a black-box objective function $f : \mathbb{R}^d \to \mathbb{R}$. Here, we consider the original unconstrained formulation of CMA-ES, presented in Hansen and Kern (2004), though constrained variants of CMA-ES have been subsequently introduced (Arnold and Hansen, 2012). Black-box optimisation assumes that we have no knowledge about the local or global structure of the objective, and only have the option of evaluating the objective function. Many black-box optimisation algorithms follow a generic template (Auger and Hansen, 2012), sketched

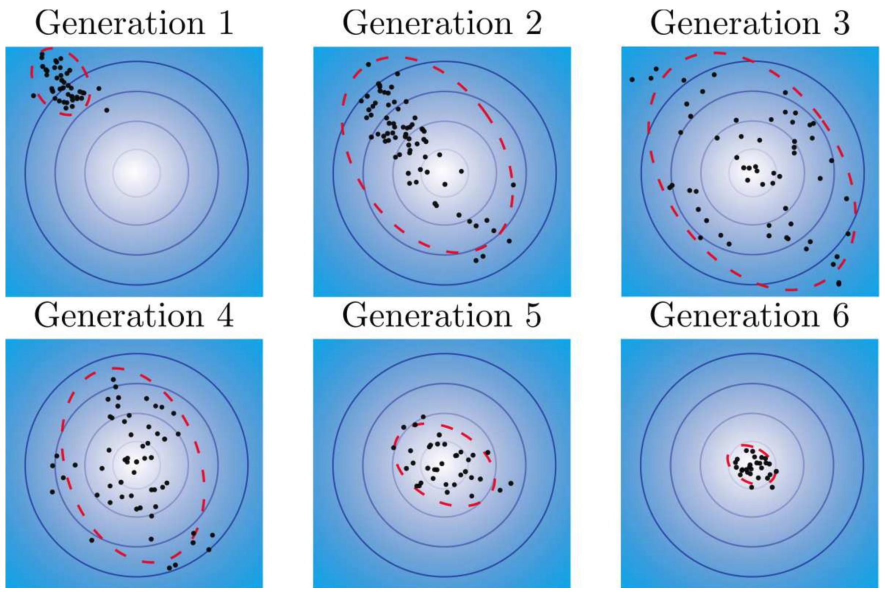


Figure 2.1: An example of CMA-ES optimising a cost function for 6 iterations (generations), where the contours are displayed, and lighter colours indicate a lower cost. Samples are given in black, and the ellipse iso-contour of the sampling distribution is in red. We observe that CMA-ES is able to rapidly locate the region with low cost. Figures from Tan et al. (2019).

in algorithm 1. According to this template, candidate solutions are iteratively sampled according to some distribution, and the function at the candidates is then evaluated. At each iteration, the candidates of the previous iteration, along with their function evaluations, are then used to update the sampling distribution. The essence of these black-box optimisation methods lies in how the sampling distribution $p(\mathbf{x}|\boldsymbol{\theta})$ is constructed and how the sampling distribution is updated.

In CMA-ES, the sampling distribution is assumed to be a multivariate Gaussian distribution. At a high level, in each iteration, CMA-ES draws λ candidates to evaluate from a Gaussian sampling distribution, then evaluates the objective function and ranks the results, finally, the Gaussian sampling distribution is updated with the candidates with the best-μ function values. Specifically, at a specific iteration k, we have $p(\mathbf{x}_k|\boldsymbol{\theta}_k) = \mathcal{N}(\mathbf{m}_k, \sigma_k^2\mathbf{C}_k)$, where $\mathbf{m}_k \in \mathbb{R}^d$ is a mean vector, $\sigma_k \in \mathbb{R}$ is a step-size parameter, and $\mathbf{C}_k \in \mathbb{S}^d_{++}$ is a positive definite covariance matrix of size $d \times d$. Additionally, we have two *evolution path* parameters $p_k^\sigma \in \mathbb{R}$ and $\mathbf{p}_k^{\mathbf{C}} \in \mathbb{R}^d$ which aid convergence. CMA-ES also requires many hyper-parameters which need to be specified. These include $\mu < \lambda \in \mathbb{N}$, which is the number of samples which we consider in the update, weights $w_1, \dots, w_\mu \in \mathbb{R}$ and $\mu_{eff} = (\Sigma_{i=1}^{\mu} w_i)^{-1}$ a variance effective

Algorithm 1: Black-box minimisation of f :

```
Initialise: Search distribution parameters $\boldsymbol{\theta}$, population size of each sampled batch
  $\lambda \in \mathbb{N}$, and solutions list $Solutions \leftarrow []$
while not terminated do
    $\{\mathbf{x}_1, \dots, \mathbf{x}_\lambda\} \sim p(\mathbf{x}|\boldsymbol{\theta})$ ;                    // Sample a batch of candidates
    Evaluate $f(\mathbf{x}_1), \dots, f(\mathbf{x}_\lambda)$;
    $solutions$.insert($\{(\mathbf{x}_1, f(\mathbf{x}_1), \dots (\mathbf{x}_\lambda, f(\mathbf{x}_\lambda))\}$) ; // Keep track of candidates
    $\boldsymbol{\theta} \leftarrow F(\boldsymbol{\theta}, \mathbf{x}_1, \dots, \mathbf{x}_\lambda, f(\mathbf{x}_1), \dots, f(\mathbf{x}_\lambda))$ ;  // Update distribution parameters
end
Return best $(\mathbf{x}, f(\mathbf{x}))$ from $Solutions$, where $f(\mathbf{x})$ is lowest.
```

selection mass factor; decay rates $c_\sigma, c_\mu, c_{\mathbf{C}}, c_1 \in \mathbb{R}$; damping factor d_σ.

At every iteration, we rank the function evaluations $f(\mathbf{x}_1), \dots, f(\mathbf{x}_\lambda)$, and select the μ samples with the best function evaluations out of the total λ samples. We now denote the selected samples as $\{\mathbf{x}_1, \dots, \mathbf{x}_\mu\}$. Then, we can use the basic update equations for our parameters, which are designed as (Hansen, 2016):

$$\mathbf{m}_{k+1} \leftarrow \sum_{i=1}^{\mu} w_i \mathbf{x}_i, \tag{2.29}$$

$$p^\sigma_{k+1} \leftarrow (1 - c_\sigma) p^\sigma_k + \sqrt{c_\sigma (2 - c_\sigma) \mu_{eff}} \mathbf{C}_k^{-\frac{1}{2}} \big(\frac{\mathbf{m}_{k+1} - \mathbf{m}_k}{\sigma_k} \big), \tag{2.30}$$

$$\sigma_{k+1} \leftarrow \sigma_k \exp \big(\frac{c_\sigma}{d_\sigma} \big(\frac{||p^\sigma_{k+1}||_2}{\mathbb{E}||N(0, I)||_2} - 1 \big) \big), \tag{2.31}$$

$$\mathbf{p}^{\mathbf{C}}_{k+1} \leftarrow (1 - c_{\mathbf{C}}) \mathbf{p}^{\mathbf{C}}_k \sqrt{c_{\mathbf{C}} (2 - c_{\mathbf{C}}) \mu_{eff}} \big(\frac{\mathbf{m}_{k+1} - \mathbf{m}_k}{\sigma_k} \big), \tag{2.32}$$

$$\mathbf{C}_{k+1} \leftarrow (1 - c_1 + c_\mu \sum_{i=1}^{\mu} x_i) \mathbf{C}_k + c_1 \mathbf{p}^{\mathbf{C}}_{k+1} {\mathbf{p}^{\mathbf{C}}_{k+1}}^\top + c_\mu \sum_{i=1}^{\mu} w_i \big(\frac{\mathbf{x}_i - \mathbf{m}_{k+1}}{\sigma_{k+1}} \big) \big(\frac{\mathbf{x}_i - \mathbf{m}_{k+1}}{\sigma_{k+1}} \big)^\top. \tag{2.33}$$

CMA-ES algorithm then conforms to the template in algorithm 1, with the samples drawn from $p(\mathbf{x}_k|\boldsymbol{\theta}_k) = \mathcal{N}(\mathbf{m}_k, \sigma_k^2 \mathbf{C}_k)$, and the update of parameters given by eqs. (2.29) to (2.33). Additional details on the selection of hyper-parameters and additional complications to the update equations can be found in Hansen (2016). An illustration of CMA-ES optimising a toy function is provided in fig. 2.1. We observe that the sampling distribution of candidate solutions quickly hones into the low cost region.

2.4 Environment Representations in Robotics

A robot navigating in an unknown environment needs to construct a representation of what it believes to be the environment – we call this representation a *map.* The robot wishes to know whether some specified coordinates of the environment are occupied or free, or the *occupancy* of the environment. In this section, we provide background on *occupancy grid maps*, a discretised representation, and *Hilbert Maps*, a more modern continuous representation of the environment.

2.4.1 Occupancy Grid Maps

Here, we briefly outline the traditional discrete map model of *occupancy grid maps* (Elfes, 1987), and its update via Baye's rule. Contributions in part I of this thesis pertain to learning-based continuous mapping methods, which are contrasted and compared against discrete approaches like occupancy grid maps.

Occupancy grid maps start by discretising the environment into a grid with some fixed resolution. We label each of the grids with an index, $i = 1, \ldots, d$, and assign a binary variable $y_i \in \{0, 1\}$, where $y_i = 1$ indicates that the event that the i^{th} cell is occupied. As the robot is moving around in the environment, it shall obtain new sensor measurements. We denote each measurement, which contains sensor data and robot position, as $s_1, s_2, \ldots, s_n$, and their combined measurements as $s_{1:n}$. For tractability, occupancy grid maps assume that each grid cell is independent, thus the joint probability of being occupied, given the sensors we have observed, is the product of its marginals:

$$p(\bigcap_{i=1}^{d} y_i = 1 | s_{i:n}) = \prod_{i=1}^{d} p(y_i = 1 | s_{i:n}). \tag{2.34}$$

where $\bigcap$ denotes the intersections of events. We are given a new sensor measurement s_{n+1} and wish to incorporate it into our existing map model. To further simplify, we assume that each sensor measurement is independent of the others. Thus, for each grid cell,

$$p(s_{1:n+1} | y_i = 1) = p(y_i = 1) \prod_{j=1}^{n+1} p(s_j | y_i = 1), \tag{2.35}$$

where $p(y_i = 1)$ is the prior, which is typically set to 0.5. Following Thrun et al. (2005), we can develop a recursive Baye's rule update equation in a stable and efficient with a *log-odds* representation. Where the odds of an event $y = 1$ is given by $\log \text{odds}(y = 1) = \log(p(y = 1)(1 - p(y = 1))^{-1})$, and the probability can be recovered by $p(y = 1) = 1 - (1 + \exp(\log \text{odds}(y = 1)))$. Incremental updates for each cell can then be expressed as:

$$\log \text{odds}(y_i = 1|s_{1:n+1}) = \log \text{odds}(y_i|s_{n+1}) + \log \text{odds}(y_i|s_{1:n}) - \log \text{odds}(y_i). \tag{2.36}$$

Here, we are assumed to have the inverse measurement model and know $p(y_i = 1|s_{n+1})$, which depends on the sensor used. Each grid cell then simply keeps track of $\log \text{odds}(y_i|s_{1:n})$. We note that the assumption each grid cell is independent may be an inaccurate one, as common obstacles would generally be expected to span multiple cells. This shortcoming is addressed by continuous representation models.

2.4.2 Hilbert Maps

Hilbert Maps (HM) (Ramos and Ott, 2016) are continuous representations of occupancy in environments, and have been shown to outperform occupancy grid maps significantly when there are fewer data points (Ramos and Ott, 2016). This is owing to the fact that natural environments are inherently continuous, and obstacles may span multiple predefined cells. HMs utilise a logistic regression classifier with projections of occupancy data into high dimensional space to obtain non-linear features. HMs are parameterised by a vector of weights with the corresponding set of features. Stochastic Gradients Descent (SGD) is then used to learn the weights of the classifier online.

We will base our discussions on features produced by projecting coordinates of interest to inducing points over spatial coordinates. These features have strong performance when used in HMs. They are known as "hinged features" as introduced in (Senanayake and Ramos, 2017), and similar to "sparse features" outlined in (Ramos and Ott, 2016). We denote the m inducing points $\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_m \in \mathbb{R}^2$ are spatially fixed inducing points. We are assumed to be given a dataset of n coordinates, and labels of whether the sensor detects the coordinate to be occupied or not. That is, we have $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where $\mathbf{x}_i \in \mathbb{R}^2$ is a coordinate in space and $y_i \in \{0, 1\}$ is a binary variable indicating whether the coordinate is occupied or not. We begin the map-building by projecting coordinates to the inducing points, using a Gaussian radial basis function, specifically,

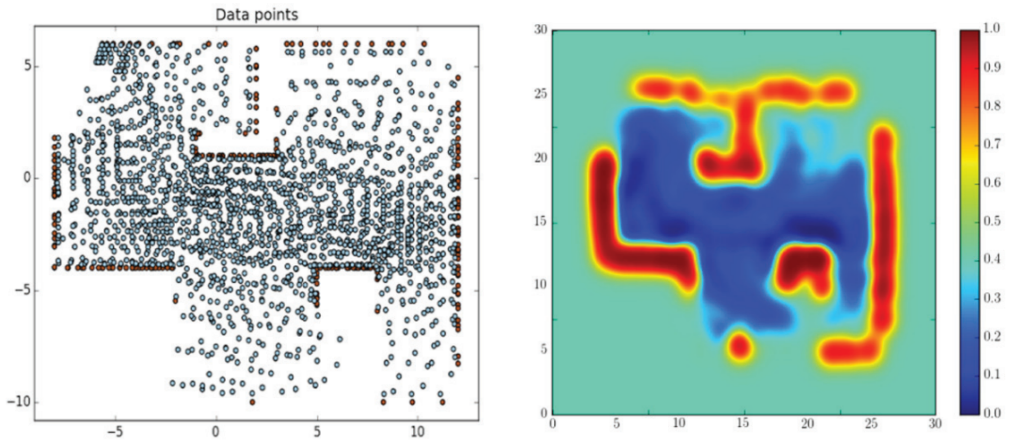


Figure 2.2: An illustration of an example Hilbert Map. (Left) Training data points used for training, blue and brown represent occupied and unoccupied points. (Right) The probability of being occupied is given by a constructed Hilbert Map. Figures from Ramos and Ott (2016).

$$\boldsymbol{\phi}(\mathbf{x}) = [\phi(\mathbf{x}, \hat{\mathbf{x}}_1), \phi(\mathbf{x}, \hat{\mathbf{x}}_2), \ldots, \phi(\mathbf{x}, \hat{\mathbf{x}}_m)], \qquad \phi(\mathbf{x}, \hat{\mathbf{x}}) = \exp(-\gamma||\mathbf{x} - \hat{\mathbf{x}}||_2^2), \tag{2.37}$$

where γ is a length-scale hyperparameter, which controls how strongly spatially-neighbouring coordinates influence one another. The probability that a coordinate $\mathbf{x}$ in the environment is unoccupied can then be expressed using a simple classifier as,

$$p(y = 1|\mathbf{x}, \mathbf{w}) = \left(1 + \exp(\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}))\right)^{-1}, \tag{2.38}$$

where $\mathbf{w}$ are weight parameters that are learnt from gathered data. The classifier can be trained by optimising a regularised binary cross-entropy loss:

$$\mathcal{L}_{\mathbf{w}}(\mathcal{D}) = \sum_{i=1}^{n} \Big[y_i \log(p(y_i = 1|\mathbf{x}, \mathbf{w})) + (1 - y_i) \log(1 - p(y_i = 1|\mathbf{x}, \mathbf{w}))\Big] + \mathcal{R}(\mathbf{w}), \tag{2.39}$$

where $\mathcal{R}(\mathbf{w})$ is the elastic-net regulariser, defined as,

$$\mathcal{R}(\mathbf{w}) = \lambda_1||\mathbf{w}||_1 + \lambda_2||\mathbf{w}||_2^2, \tag{2.40}$$

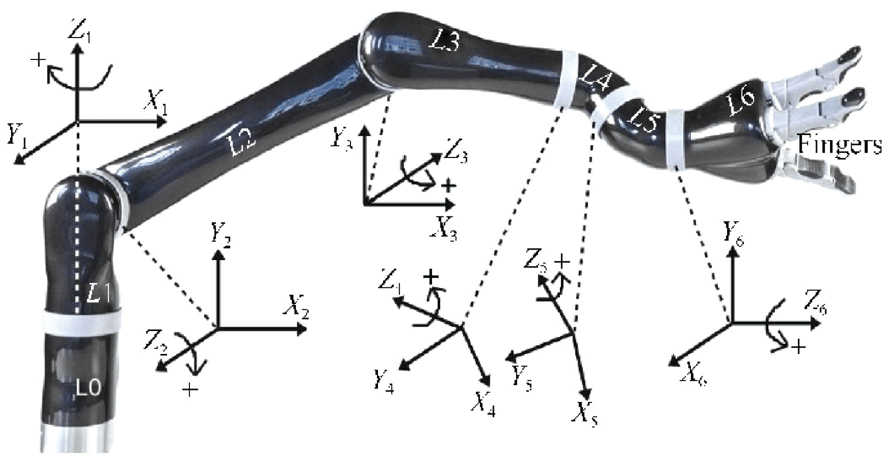


Figure 2.3: The JACO arm used in real-world experiments in chapter 8 and chapter 9 has 6 degrees of freedom (not including the gripper), one for each of 6 revolute joints. Figure adapted from Bhattacharyya et al. (2017).

where λ_1 and λ_2 are regularisation hyperparameters, which control how strongly the loss is regularised. We can train Hilbert Maps efficiently online, with streaming data, via stochastic gradient descent. When we wish to predict the probability of coordinates being occupied, we can simply project the query coordinates to the inducing points with eq. (2.37), and then compute the probabilities via eq. (2.38). An example of a built Hilbert Map along with the training data used is illustrated in fig. 2.2.

2.5 Robot Manipulator Kinematics

This section provides a brief introduction to manipulator kinematics, which is crucial to understand many of the techniques in chapter 8 and chapter 9. We begin by providing describing the configuration of a manipulator. Then, we introduce forward kinematics, mapping the joint configurations of the manipulator to the Cartesian coordinates of its end-effector.

2.5.1 Manipulator Configurations

In this thesis, the manipulators explored are restricted to be fixed base, where the root link is connected to a stationary platform, and between each link, or between the base and a link, there is a single joint. These robot manipulator systems can be modelled by n_q joints, n_q links, along with a fixed-base. We denote the displacement for each joint as q_i, $i = 1, \dots, n_q$, where each joint typically has some upper and lower displacement limits. We describe the current state of such as robot by its *configuration* $\mathbf{q}$, and the set of all feasible configurations as the *Configuration Space (C-space)*, denoted as $\mathcal{Q}$. Specifically,

$$\mathbf{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_{n_q} \end{bmatrix}, \qquad \text{where } \mathbf{q} \in \mathcal{Q}. \tag{2.41}$$

In this thesis, all of the manipulators contain revolute joints, and the configurations correspond to the rotation angles of the joints. The JACO manipulator used in the experiments of chapter 8 and chapter 9 has 6 degrees of freedom (not including the gripper), and is illustrated in fig. 2.3. In the absence of additional kinematic constraints, which are restrictions on the movements of components of the robot, the coordinates of the robot's C-space are *minimal coordinates*, specifically, that its dimension is exactly the degrees-of-freedom of the system.

Although we can specify the motion of a robot within its C-space coordinates, it can often be difficult to translate the geometry of the surrounding environment into C-space coordinates. Therefore, we would often reason about tasks in the robot's natural environment with *task-space* coordinates, the Euclidean coordinate system with respect to a point on the robot, typically the end-effector.

2.5.2 Forward Kinematics

The *forward kinematics* then refers to the mapping between the configuration of the robot to the displacement and orientation (the pose). Note that in this thesis, we may on occasion restrict ourselves to only reasoning about the displacement of the end-effector, and discard the rotation from consideration. For brevity, we shall also loosely refer to the mapping between robot configurations and end-effector displacement only as "forward kinematics". The forward kinematics $f_e(\mathbf{q})$ can often be surjective – there are many joint configurations that can result in the same end-effector displacement. Additionally, evaluating the forward kinematics is often efficient, and consists of a sequence of rigid body transformations, which only requires linear algebra and trigonometry. On the other hand, *inverse kinematics*, mapping from end-effector coordinates to joint configurations is typically much more cumbersome and may require numerical optimisation.

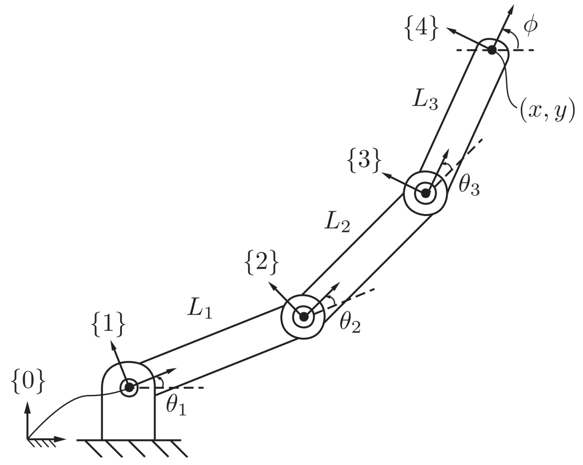


Figure 2.4: The 3 degrees of freedom planar manipulator used in the example. Example and figure from Lynch and Park (2017).

Here, we give an example (from Lynch and Park (2017)) of constructing the forward kinematics of a simple planar manipulator with 3 links. Forward kinematics can be written as a sequence of transformation matrix products, starting from the manipulator base to the end-effector. Suppose the robot has 3 links of length L_1, L_2, L_3, and 3 controllable joints $\theta_1, \theta_2, \theta_3$ at the end of each link. The fixed frame reference at the origin is labelled as $\{0\}$, three link reference frames are respectively labeled $\{1\}$, $\{2\}$, $\{3\}$, and the reference frame at the end-effector as $\{4\}$. We define the robot configuration as $\mathbf{q} = (\theta_1, \theta_2, \theta_3)$, and denote the end-effector orientation as ϕ. An illustration of the 3 degrees of freedom planar manipulator is shown in fig. 2.4. Then, we can define the transformation matrix between the reference frames as:

$$\mathbf{T}_{0,1} = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}_{1,2} = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & L_1 \\ \sin(\theta_2) & \cos(\theta_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.42}$$

$$\mathbf{T}_{2,3} = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & L_2 \\ \sin(\theta_3) & \cos(\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}_{3,4} = \begin{bmatrix} 1 & 0 & 0 & L_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.43}$$

We seek the forward kinematics, f_e, that maps our configurations $\mathbf{q}$ to positions and orientations (x, y, ϕ). We begin by finding the transformation matrix from reference frame $\{0\}$ to $\{4\}$, given by:

$$\mathbf{T}_{0,4} = \mathbf{T}_{0,1}\mathbf{T}_{1,2}\mathbf{T}_{2,3}\mathbf{T}_{3,4}. \tag{2.44}$$

The displacements in the x, y-plane of the end-effector, relative to the fixed-base origin, can be extracted from the matrix as $x = \mathbf{T}_{0,4}^{(1,4)}$ and $y = \mathbf{T}_{0,4}^{(2,4)}$, where the super-scripted tuples indicate the rows and columns of the matrix, respectively. The 2×2 upper-left sub-matrix of $\mathbf{T}_{0,4}$, i.e. $\mathbf{T}_{0,4}^{(1:2,1:2)}$, is the rotation matrix of ϕ. Hence, ϕ can be obtained via $\phi = \mathtt{atan2}(\mathbf{T}_{0,4}^{(2,1)}, \mathbf{T}_{0,4}^{(1,1)})$, where $\mathtt{atan2}$ is the 2-argument arctan function (standard 9899:1999, 1999). Here the forward kinematics f_e, with end-effector orientation, is given by:

$$\begin{bmatrix} x \\ y \\ \phi \end{bmatrix} = f_e(\mathbf{q}) = \begin{bmatrix} \mathbf{T}_{0,4}^{(1,4)} \\ \mathbf{T}_{0,4}^{(2,4)} \\ \mathtt{atan2}(\mathbf{T}_{0,4}^{(2,1)}, \mathbf{T}_{0,4}^{(1,1)}) \end{bmatrix} \tag{2.45}$$

We are also often interested in the velocities in both the task-space and the C-space. Let us denote the end-effector position coordinates as $\mathbf{x}_e \in \mathbb{R}^3$. The instantaneous velocities at a specific configuration $\mathbf{q}$ are linked via:

$$\dot{\mathbf{x}}_e = \mathbf{J}_{f_e}(\mathbf{q})\dot{\mathbf{q}} \qquad \Longleftrightarrow \qquad \dot{\mathbf{q}} = \mathbf{J}_{f_e}^{\dagger}(\mathbf{q})\dot{\mathbf{x}}_e, \tag{2.46}$$

where $\mathbf{J}_{f_e}$ denotes the Jacobian of the forward kinematics. $\mathbf{J}_{f_e}^{\dagger}$ is a *generalised inverse* of $\mathbf{J}_{f_e}$, as typically many configurations can result in the same end-effector displacement, i.e. $\mathbf{J}_{f_e}$ has more rows than columns. In particular, in this thesis, we shall use the Moore-Penrose (Penrose, 1956), which obtains the least squares solution to the over-determined system, where $\mathbf{J}_{f_e}^{\dagger} = (\mathbf{J}_{f_e}^{\top}\mathbf{J}_{f_e})^{-1}\mathbf{J}_{f_e}^{\top}$.

2.6 Dynamical Systems and Differential Equations

Throughout this thesis, we often model the motion of the robot as a dynamical system, described by an ordinary differential equation (ODE). A dynamical system represents how a *state*, i.e. a collection of values that abstract the system at the current snapshot, evolves through time. A state can be viewed as a point in *state-space*, the set of all possible configurations of a system. For example, we may wish to describe how the position of a point in the task-space of a manipulator evolves. Here, the state-space can be the 3d Cartesian coordinate system and the state a position coordinate. Beyond modelling in the task-space, we can construct dynamical systems in the configuration space of the robot manipulator, where a state of a robot is often given as its joint configurations. In particular, dynamical systems in configuration space are used to model manipulator motion in chapters 8 and 9.

2.6.1 Trajectories of Dynamical Systems

In this thesis, we investigate *continuous-time* dynamical systems, where the system does not commit to a fixed time resolution. We limit our discussion to *first-order* systems, where the order of the time derivatives is at most one. Second-order dynamical systems arise in robotic systems when taking into account acceleration, such as in chapter 9. However, these can be converted into a first-order system, by absorbing the state velocities and augmenting the state vector. A d-dimensional continuous-time first-order system with state-space $\mathcal{S} \subseteq \mathbb{R}^d$ and states $\mathbf{x} \in \mathcal{X}$ can be expressed as the initial value problem,

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), t), \qquad \mathbf{x}(0) = \mathbf{x}_0, \tag{2.47}$$

where $\dot{\mathbf{x}} \in \mathcal{TX}$ is the time derivative of the state, $t \in \mathbb{R}$ is a time variable, $\mathbf{x}_0 \in \mathcal{X}$ gives the initial state of the system, and $f : \mathcal{X} \times \mathbb{R} \rightarrow \mathcal{TX}$ is the system dynamics function. Here, we denote the tangent space of the state space as $\mathcal{TX}$, i.e. the space of possible velocities for a particle in $\mathcal{X}$. Intuitively, we can think of f as a vector field, mapping each coordinate in the state-space to a valid velocity. This vector field perspective of dynamical systems is revisited in chapter 8 when we introduce our Diffeomorphic Templates method. An example of a non-linear dynamical system along with three integrated trajectories is shown in fig. 2.5. The dynamics of the system can be visualised as a 2d vector field, illustrated in red.

Motion trajectories of a robot can be obtained from a dynamical system description of robot motion by "rollouts". Trajectories of eq. (2.47), which we denote as $\xi(t, \mathbf{x}_0)$, describe the state of the system at a given time after starting at a given initial condition and can be obtained via evaluating the integral:

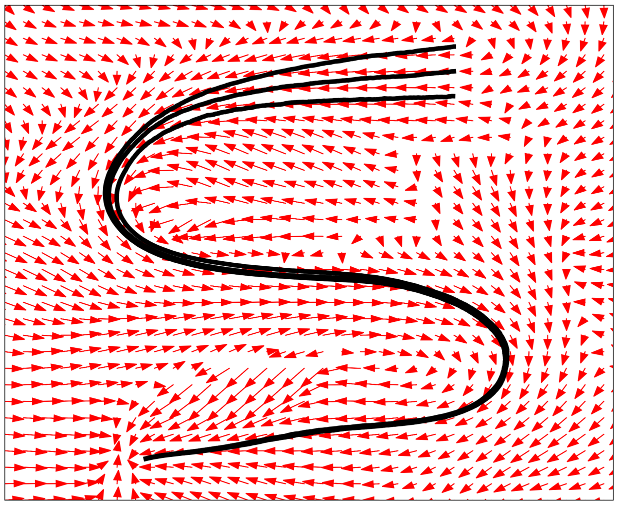


Figure 2.5: An example of a non-linear time-invariant dynamical system from chapter 8, three trajectories are integrated are shown in black, and the dynamics are visualised as a vector field in red. This system is globally asymptotically stable, and the integrated trajectories converge.

$$\xi(t, \mathbf{x}_0) = \mathbf{x}_0 + \int_0^t f(\mathbf{x}(s), s)\mathrm{d}s. \tag{2.48}$$

The tractability of this integral depends on the class of dynamical systems our system belongs to. If f is independent of time, i.e. $f(\mathbf{x}, t) = f(\mathbf{x})$ for all t, the system is known to be *time-invariant* or *autonomous*. If the dynamics are linear in $\mathbf{x}$, that is $f(\mathbf{x}, t) = \mathbf{A}(t)\mathbf{x}$, then the system is known as a linear system. Linear time-invariant systems, i.e. systems of the form $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, admit closed-form integrals. However, linear time-invariant systems are very restricted in its ability to model real-world phenomena, and the dynamical systems discussed in this thesis are generally non-linear. The use of numerical ODE integrators are needed to evaluate eq. (2.48).

Numerical ODE integrators discretise time, and recursively integrate the states to roll-out a trajectory of states. Integrators for first-order dynamical systems can be categorised as *explicit* or *implicit*: explicit methods express the state of the system in the future from the current state, while implicit methods require solving an algebraic expression which involves both the current state and the latter state. Explicit integrators are generally more efficient, but suffer from instabilities. Here, we shall outline explicit Euler's method and implicit Euler's method, which are some of the simplest examples of explicit and implicit methods.

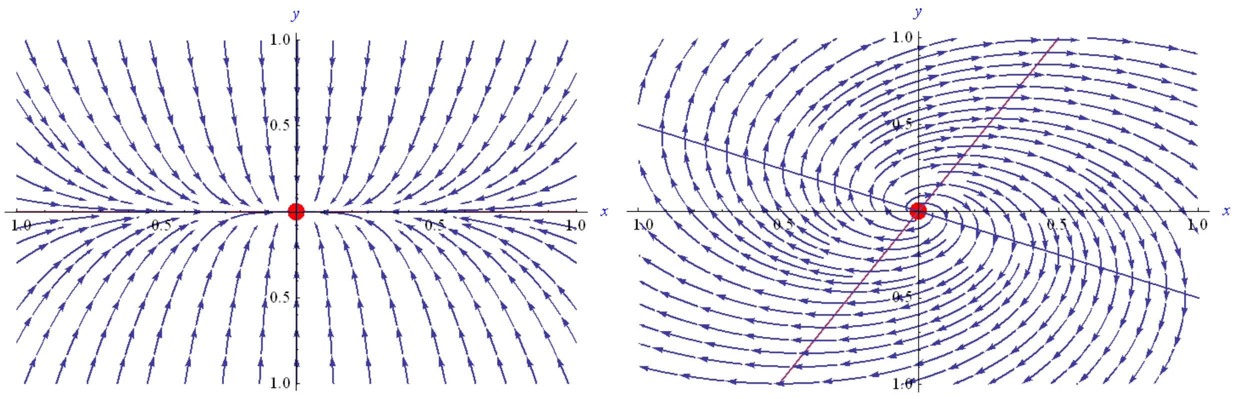


Figure 2.6: (Left) An example of a globally asymptotically stable equilibrium (an attractor); (Right) An unstable equilibrium (unstable spiral). The equilibrium points are shown in red, and dynamics are indicated by arrows. Figures from Füllsack (2013).

For both of these methods, we shall first specify a step-size Δt. In explicit Euler's method, we compute the states at the next step using the update:

$$\mathbf{x}(t + \Delta t) \approx \mathbf{x}(t) + hf(\mathbf{x}(t), t). \tag{2.49}$$

The implicit Euler's method, on the other hand, considers the velocities at the next step. Specifically, the update equation is given as:

$$\mathbf{x}(t + \Delta t) \approx \mathbf{x}(t) + hf(\mathbf{x}(t + \Delta t), t + \Delta t). \tag{2.50}$$

This eq. (2.50) requires solving an algebraic equation to recover $\mathbf{x}(t + \Delta t)$. In practice, this is done with a root-finding algorithm such as Newton-Raphson's method (Ypma, 1995).

2.6.2 Asymptotic Stability of Dynamical Systems

In the contributing chapter 8, we require our method to preserve *asymptotic stability*. Here, we shall give a brief definition on the asymptotic stability of systems. One often wishes to examine the long-term behaviour of dynamical systems – some of the questions that we may seek to answer include: When we start at some initial conditions and integrate to roll-out trajectories, what happens to the states as time progresses? Do the trajectories converge to a single point, or multiple points, or fly off and diverge? How does the initial condition impact where a trajectory ends up?

We shall describe the stability of a system with respect to equilibrium points. An equilibrium point, $\boldsymbol{x}^*$, is a point in state-space where velocity is zero, $\boldsymbol{x}^* \in \mathbb{R}^d : f(\boldsymbol{x}^*) = 0$. Additionally, a system is *locally asymptotically stable* in a region $S \subset \mathbb{R}^d$ if trajectories starting in S converge to some equilibrium $\boldsymbol{x}^* \in S$,

$$\lim_{t\to\infty} \xi(t, \boldsymbol{x}_0) = \boldsymbol{x}^*, \qquad \forall \boldsymbol{x}_0 \in S. \tag{2.51}$$

Furthermore, a system is *globally asymptotically stable* if $S = \mathbb{R}^d$, and all trajectories converge to a unique equilibrium point. Alternatively, equilibrium is known to be unstable if a small perturbation to a particle at the equilibrium shall result in the particle being repelled from the equilibrium. An example of a globally asymptotically stable equilibrium and one of an unstable equilibrium are shown in fig. 2.6. The system illustrated in fig. 2.5 is also globally asymptotically stable.

2.7 Summary

In this chapter, we have introduced some of the basic conceptions which will resurface throughout this thesis. Machine learning methods can be generally described as approaches which leverage data to make predictions or find patterns without being explicitly given the rules of how to do so. The archetype of a machine learning problem is the regression problem, which we outline in section 2.2, early on in this chapter. In section 2.2.1, we discuss linear regression, potentially used in conjunction with non-linear features, as one of the most straightforward approaches for solving regression problems. We illustrate with an example, in section 2.2.2, how dot products on non-linear features can be more efficient with the introduction of kernel functions. Next, in section 2.2.3, we give an overview of fully-connected neural networks, the most fundamental of neural network models. Modern machine learning is heavily dominated by neural network models, which are highly flexible over-parameterised models, that are exceptionally parallelisable. Then in section 2.3, we provide background on optimisation techniques used in the thesis, These include (1) ADAptive Moment estimation (ADAM), an extension of Stochastic Gradient Descent that uses momentum, and is typically used to optimise neural network models; (2) Sequential Quadratic Programming, capable of efficiently finding local solutions for constrained optimisation problems where derivatives are available; (3) Covariance Matrix Adaptation Evolution Strategy, a black-box derivative-free optimiser. We have also lightly introduced background knowledge to related robotics topics. In section 2.4, we describe methods to represent a robot's environment. These include occupancy grid maps, an early and widely-used model to represent the free and occupied space in an environment, along with Hilbert Maps, a method that avoids discretisation of the environment. Then, in section 2.5, we give some background on the essentials of robot manipulator manipulators. We elaborate on the foundational concept of a "configuration space", a vector space that consists of every possible geometric permutation of the robot. Then, we elaborate on *forward kinematics* a mapping between the configuration space and the position of the end-effector. Finally, in section 2.6, we give background on dynamical systems and differential equations, and discuss the asymptotic stability of dynamical systems.

In the rest of the thesis, we shall develop our contributed robot learning methods, and the background presented in this chapter will be revisited throughout the contributing chapters. Coming up next is our first contributing chapter, chapter 3, where we introduce our contributions to learning continuous occupancy representations and more importantly the fusion of these models.

Part I

Learning for Continuous Environment Representation

Chapter 3

Fusion of Continuous Occupancy Maps

3.1 Introduction

The deployment of multiple robots can deliver many benefits by allowing for the parallelisation of tasks. Compared with an individual robot, the deployment of multiple robots can provide speed improvements, and increase the robustness of the system, by reducing the dependency on any single robot. In particular, within a decentralised multi-robot system, there is no single central fusion center (Durrant-Whyte and Henderson, 2008), hence the impact of an individual robot malfunctioning to the system is limited. In light of these advantages, this chapter examines continuous occupancy mapping in multi-robot systems. Building reliable representations of an environment is an integral part of the exploration of environments, a fundamental problem in mobile robotics. In a multi-robot scenario, the data collected by individual robots need to be integrated into a single consistent model of the environment (Fox et al., 2006). Real-time decentralised mapping using multiple robots (Smith et al., 2012) (Darmanin and Bugeja, 2017) has various applications including agriculture (McAllister et al., 2018), environmental monitoring (Valada et al., 2014), and disaster-relief (Gregory et al., 2016) (Mendonça et al., 2016).

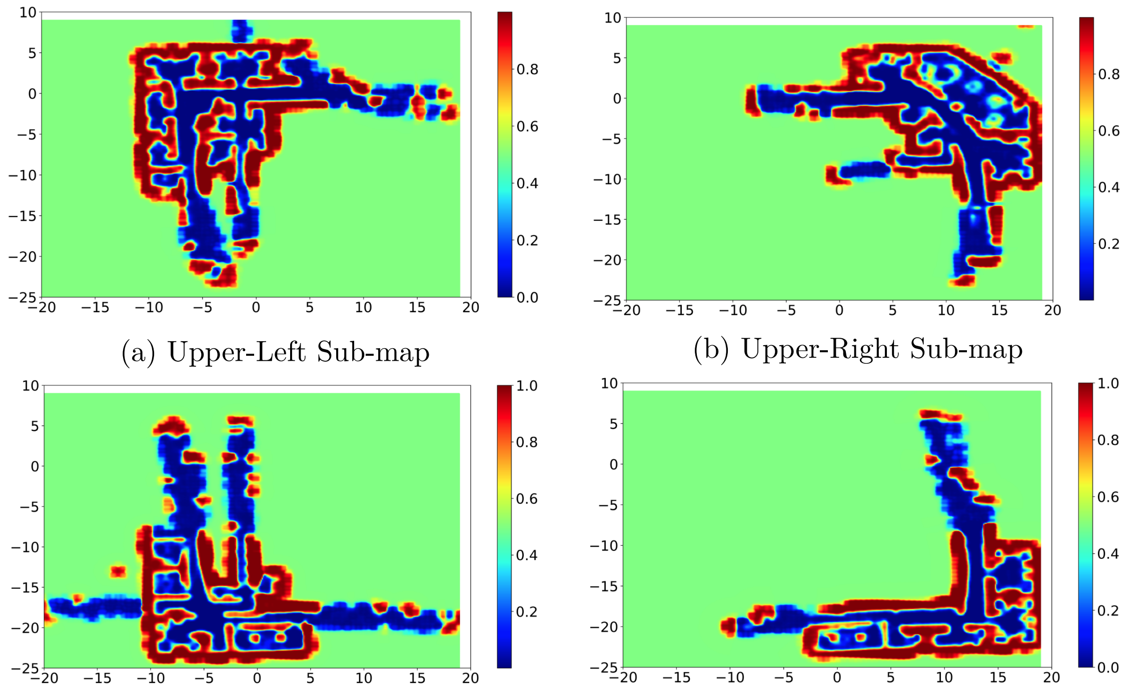
This chapter has been published in ICRA as Zhi et al. (2019a).

Historically, discrete grid maps, which assume independence between grid cells, have been used to represent the occupancy of the environment (Elfes, 1987). Although the strong assumption of independence between cells allows for efficient operations on grid maps, it ignores the spatial dependency of the environment. As occupancy in the real world is continuous and not discretised into a grid with independent cells, an occupancy grid representation may not be able to adequately capture occupancy of the real-world. Gaussian process occupancy maps (GPOM) (O'Callaghan et al., 2009) were introduced as a method to build continuous occupancy maps, by using kernels to capture spatial dependencies between occupancy data. However, GPOMs do not scale efficiently, due to its cubic time complexity on the number of all data-points used, making it impractical to use real-time online on a robot. Another framework to continuously represent the environment, the Hilbert Maps (HM) framework (Ramos and Ott, 2016), was introduced as a much faster framework to represent the environment continuously, and does not require the retention of past data points. The Hilbert Map framework was subsequently extended to Bayesian Hilbert Maps (BHM) (Senanayake and Ramos, 2017) to capture the uncertainty of parameters in the model. However, training BHMs require the inversion of a covariance matrix between parameters, a computationally expensive operation of cubic time complexity. Due to the long run-times needed for this operation, multi-robot mapping with BHMs in real time is impractical. To reduce run-time, we introduce Fast Bayesian Hilbert Maps (Fast-BHM) that remove the need to invert a covariance matrix, significantly speeding up the training.

Motivated by the advantages that continuous occupancy mapping and multi-robot mapping could provide, this chapter explores decentralised multi-robot merging of continuous occupancy maps, within variations of the Hilbert Map framework. The main contributions of this chapter are:

1. Developing Fast-BHM, a significantly sped-up variation of the Bayesian Hilbert Map model (Senanayake and Ramos, 2017).
2. Developing methods to fuse Fast-BHMs models, which can be built by individual robots, to obtain a unified Fast-BHM model.

The chapter is organised as follows. In section 3.3, we introduce the Fast-BHM model, as a significantly sped-up variation of the BHM model, mitigating the impractical run-times required to train BHMs. This is followed, in section 3.4, by the presentation of a decentralised



(a) Upper-Left Sub-map (b) Upper-Right Sub-map

(c) Lower-Left Sub-map (d) Lower-Right Sub-map

Figure 3.1: Sub-maps trained on scans from different sections of the Intel (Hahnel et al., 2003) dataset. In a multi-agent set-up, we require a method to fuse the sub-maps into a unified model.

scheme to fuse Fast-BHMs. Empirical results highlighting the effectiveness of our merging scheme, and the speed improvement of Fast-BHMs are then shown in section 3.5.

3.2 Related work

Map fusion aims to build a consistent representation of the environment based on data collected and maps built by different agents. As each individual robot moves around the environment, it builds local sub-maps based on the data it receives. These individual sub-maps can then be periodically fused to obtain a global map. Decentralised fusion methods allow each node the ability to build a global map, without relying on a centralised node (Durrant-Whyte and Henderson, 2008). Each robot can obtain a local copy of the global map, potentially providing each individual robot with information about regions that it has not yet explored, and continue to refine the map with additional data. A well-known approach to building multi-robot occupancy grid maps with known poses is through concurrent exploration and updating via Bayes filters (Thrun, 2001). Methods have also been developed for multi-robot mapping using grid maps where the pose is unknown (Howard, 2006) (Thrun and Liu, 2005).

Attempts have also been made to fuse predictions from continuous occupancy representations, under the assumption of known pose and local measurements are mapped to a global reference frame. These include fusing predictions from Gaussian Process Maps (Jadidi et al., 2014) (Kim and Kim, 2014) using Bayesian Committee Machines (Tresp, 2000) and iteratively fusing the predictions made by Hilbert Maps using an update rule (Doherty et al., 2016). These methods obtain local representations of the environment, then either pre-select sample points to query from, and combine the estimates from the individual local estimations, or represent the global model as a discretised grid (Doherty et al., 2016). Unlike these methods which aim to fuse predictions from individual models, our method merges the underlying local Fast-BHM models to arrive at a unified and compact global Fast-BHM model. This has the advantage of enabling the points to be queried to be decided after the fusion occurs, freeing users from having to define sample points *a priori* to build a grid map from queries. After this model is obtained, we can query any point in the environment.

3.3 Fast Bayesian Hilbert Maps

Bayesian Hilbert Maps (BHM) have been introduced as an extension to Hilbert Maps (Senanayake and Ramos, 2017). We refer the reader to section 2.4.2 for background on Hilbert Maps. Unlike Hilbert Maps, BHMs do not heavily depend on regularisation, eliminating the need to pick regularisation hyperparameters. Bayesian Hilbert Maps (BHM) (Senanayake and Ramos, 2017) are obtained under the assumption that weights approximately follow a multivariate normal distribution, $p(\mathbf{w}) \approx Q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. An Expectation-Maximisation (EM) (Dempster et al., 1977) -like approach is then used to iteratively learn the parameters. The update of the parameters requires the inversion of the full covariance matrix, $\boldsymbol{\Sigma} \in \mathbb{R}^{T \times T}$. This operation has a time complexity of approximately $O(T^3)$. The inversion of $\boldsymbol{\Sigma}$ hinders the usage of Bayesian Hilbert Maps in real-time on robots – it is not practical to train Bayesian Hilbert Maps in real time for large open environments. Furthermore, the full covariance matrix can also be relatively large, requiring good bandwidth to conduct multi-robot map fusion. We shall now develop Fast Bayesian Hilbert Maps (Fast-BHM), which enables the maps to be trained without the need to invert the covariance matrix.

With the aim of avoiding the usage and inversion of large matrices in mind, we propose a variation of BHMs which assumes independence between weights, Fast Bayesian Hilbert Maps (Fast-BHMs). Fast-BHMs enable us to train BHMs without constructing the full covariance matrix. As each weight in the logistic regressor, used in the original Hilbert Maps formulation (Ramos and Ott, 2016), is simply a scalar value and does not depend on one another, we assume that each weight of the Bayesian Hilbert Map can be approximated as a normal distribution and independent of one another, giving a mean-field Gaussian distribution:

$$p(\mathbf{w}) = \prod_{t=1}^{T} p(w_t) \approx \prod_{t=1}^{T} \mathcal{N}(w_t|\mu_t, \sigma_t^2), \tag{3.1}$$

where T is the number of weights present in the Bayesian Hilbert Map. Then the update equations for Fast-BHM can be derived in a manner similar to the derivation of the variational logistic regression (Bishop, 2007). It is important to note that the weights are parameters of a continuous function, assuming that these weights are independent is not the same assumption as the cell independence assumption made in occupancy grid maps. The parameters are learnt in an EM approach (Dempster et al., 1977). See the derivation of the variational logistic regression (Bishop, 2007), (Jaakkola and Jordan, 1997) for details.

We denote $\mathbf{y}$ as a vector of binary variables indicating the occupancy of coordinates. $\boldsymbol{\phi}(\mathbf{x})$ denotes a feature, obtained by applying a kernel transformation on data point $\mathbf{x}$. We take the variational logistic regression paradigm outlined in chapter 10 of Bishop (2007) and reproduce the framework here. The weights parameterising the model are denoted as $\mathbf{w}$, and a vector containing intermediate variables $\mathbf{z} \in \mathbb{R}^N$, where N is the number of data points for the training batch. Each intermediate variable (each element in $\mathbf{z}$) is used to produce a lower bound on the sigmoid function:

$$\sigma(\hat{z}) \geq \sigma(z) \exp\Big\{\frac{(\hat{z} - z)}{2} - \lambda(z)(\hat{z}^2 - z^2)\Big\}, \text{ for any } \hat{z} \in \mathbb{R}, \tag{3.2}$$

$$\text{where, } \lambda(z) = \frac{1}{2z}\big[\sigma(z) - \frac{1}{2}\big]. \tag{3.3}$$

Under the *variational inference* framework (Jaakkola and Jordan, 1997), (Blei et al., 2017), we aim to maximise a lower bound to $\log p(\mathbf{y}|\boldsymbol{\phi}(\mathbf{x}))$ by learning parameters to $h(\mathbf{w}, \mathbf{z})$, a lower bound of $p(\mathbf{y}|\boldsymbol{\phi}(\mathbf{x}), \mathbf{w})$,

$$\begin{aligned}\log p(\mathbf{y}|\boldsymbol{\phi}(\mathbf{x})) &= \log \int p(\mathbf{y}|\boldsymbol{\phi}(\mathbf{x}), \mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w} \\ &\geq \log \int h(\mathbf{w}, \mathbf{z})p(\mathbf{w})\mathrm{d}\mathbf{w}\end{aligned} \tag{3.4}$$

E-step:

$$Q(\mathbf{z}, \mathbf{z}^{old}) = \mathbb{E}\big[\log\big(h(\mathbf{w}, \mathbf{z})p(\mathbf{w})\big)\big] \tag{3.5}$$

M-step:

$$\mathbf{z} = \arg\max_{\mathbf{z}} Q(\mathbf{z}, \mathbf{z}^{old}) \tag{3.6}$$

We determine the parameters by maximising the lower bound of the marginal likelihood. We can derive the lower bound of the marginal likelihood by building on a lower bound of the sigmoid logistic, used by the authors introducing Bayesian Hilbert Maps (Senanayake and Ramos, 2017), and noted in (Bishop, 2007). Similar to the Sequential Bayesian Hilbert Maps method (Senanayake and Ramos, 2017), we wish to train the model sequentially, using scans from the current time step to train weights estimates obtained from the previous step. Following the derivation of sequentially trained BHMs, we assume that the prior can be written as $p(\mathbf{w}) = \prod_{t=1}^{T} \mathcal{N}(w_t|\mu_{t,k-1}, \sigma^2_{t,k-1})$, where k is the current time step. Closely following the derivation touched on in (Senanayake and Ramos, 2017), and detailed in (Bishop, 2007), we arrive at the update formulae for the E-step:

$$\mu_{t,k} = \sigma^2_{t,k}\big(\sigma^{-2}_{t,k-1}\mu_{t,k-1} + \sum_{n=1}^{N_k}(y_n - 0.5)\boldsymbol{\phi}(x_{n,t})\big) \tag{3.7}$$

$$\begin{aligned}\sigma^{-2}_{t,k} =& \sigma^{-2}_{t,k-1} + 2\sum_{n=1}^{N_k}\lambda(z_n)\boldsymbol{\phi}(x_{t,n})^2 \\ &\text{for all } t \in \{1, ..., T\}\end{aligned} \tag{3.8}$$

There are N_k data points in the scan of time k, and T number of weights in total. The updating formula for the M-step is obtained as:

$$z_n^2 = \sum_{t=1}^{T}\boldsymbol{\phi}(x_{n,k})^2(\sigma^2_{t,k} + \mu^2_{t,k}) \tag{3.9}$$

Where n is an index in $1, 2, ..., N_k$. These equations allow us to train Fast-BHMs. The bottleneck for Fast-BHMs in the method is matrix multiplication, which is of sub-quadratic complexity relative to the cubic complexity of matrix inversion. In the next section, we present a method to merge Fast-BHM maps in a decentralised manner.

3.4 Merging Bayesian Hilbert Maps

We develop methods to combine several individual Fast-BHMs trained on different, but potentially overlapping, data points into a single Fast-BHM. The merged Fast-BHM itself is a continuous representation of the environment. It can be trained further with new data and merged with other Fast-BHMs. Denoting the probability density functions of weights as f, the merging of several individual Fast-BHMs into a single BHM can be written as:

$$f(\mathbf{w}| \cup_{m=1}^{M} D_m) = \mathbb{F}[f(\mathbf{w}|D_1), ..., f(\mathbf{w}|D_M)], \tag{3.10}$$

where there are M individual Hilbert Maps, and there are N_m data points used to train the m^{th} Fast-BHM is expressed as $D_m = \{\mathbf{x}_i^m, y_i^m\}_{i=1}^{N_m}$. $\mathbb{F}$ denotes the method used to merge Fast-BHMs.

We assume that the individual maps being merged shared the same set of features, and therefore the size of each weight vector, $f(\mathbf{w}|D_m)$, and the merged vector, $f(\mathbf{w}| \cup_{m=1}^{M} D_m)$, will be of the same size. For the resultant merged weights to be constructed as a Bayesian Hilbert Map, we also need to maintain that the merged weights of the resultant map are normally distributed.

We can see that the difficulty of merging the weights from individual maps lies in combining local estimates of weights when the estimates differ, as there is common information between the two local estimates of the same weight (Durrant-Whyte and Henderson, 2008). We also need to maintain that the individual weights of the merged map are Gaussian. Consider the merging of two BHMs:

$$\begin{aligned} f(\mathbf{w}|D_1 \cup D_2) &= \frac{f(D_1 \cup D_2|\mathbf{w})f(\mathbf{w})}{f(D_1 \cup D_2)}, \\ \text{where } f(D_1 \cup D_2|\mathbf{w}) &\propto \frac{f(D_1|\mathbf{w})f(D_2|\mathbf{w})}{\underbrace{f(D_1 \cap D_2|\mathbf{w})}_{\text{Common Information}}}. \end{aligned} \tag{3.11}$$

3.4.1 Conflation

We will now introduce the concept of Conflation (P. Hill, 2008) as a method to fuse distributions of random variables. The Conflation operation can be used to consolidate results from independent experiments that estimate the same quantity. Conflation is defined in Definition 1. The original authors prove Conflation minimises the maximum loss in Shannon Information and is the best linear unbiased estimate. Before the first fusion of any sub-maps, we assume that the valid local estimates of a particular weight are conditionally independent only on the global weight. This is based on the assumption that the data points retrieved from the sensor are assumed to be independent, and are free from any biases of the sensor. This assumption holds fairly well, as demonstrated by the results of conflation in our experiments. After the initial merger of separate Hilbert Maps, we may wish to continue training on copies of the merged map. In this scenario, we assume that the global map from the previous merge is the common information between the different updated copies in the next merge.

Definition 1 ((P. Hill, 2008)) *Let* $X_1, X_2, ..., X_n$ *have PDFs* $f_1, f_2, ..., f_n$ *satisfying* $0 < \int_{-\infty}^{\infty} \prod_{i=1}^{n} f_i(x) \mathrm{d}x < \infty$*. Then the conflation* $\&(X_1, X_2, ..., X_n)$ *is continuous with density:*

$$f(x) = \frac{f_1(x) f_2(x) ... f_n(x)}{\int_{-\infty}^{\infty} f_1(y) f_2(y) ... f_n(y) \mathrm{d}y} \tag{3.12}$$

As the product of the probability density functions (PDF) of normal random variables is the PDF of another normal distribution, $f_1(y)f_2(y)...f_n(y)$ is the PDF of a normal distribution, and the integral with respect to y is equal to 1. Therefore, the conflation of N normally distributed random variables is simply the product of the PDF of the variables, $f_{Merged}(x) \approx \&(f_1, ..., f_N) = \prod_{n=1}^{N} f_n(x)$, which is also a PDF of a normal distribution. This property allows multiple cycles of training and merging to occur.

The fusion to arrive at the global estimate of a particular weight w_{global} from n valid local estimates, independent conditioned on the true value of the global weight, can be written as:

$$w_{global} \approx \&(w_1, .., w_n) \sim \mathcal{N}\left(\frac{\sum_{i=1}^{n} \frac{\mu_i}{\sigma_i^2}}{\sum_{i=1}^{n} \sigma_{i=1}^{-2}}, \frac{1}{\sum_{i=1}^{n} \sigma_i^{-2}} \right) \tag{3.13}$$

Conflation assumes that the estimates were computed independently, and using it directly

Algorithm 2: Fast-BHM Sequential Fusion

```
Input:
$N$ sub-maps to fuse, $\mathbf{w}_1, ..., \mathbf{w}_N$.;
Number of merges before stopping, $M$ Map Fusion at Robot $n$, with map $\mathbf{w}_n$
for ( $m \in \{1, ..., M\}$ ) {
    for ( $i \in \{1, ..., N\}$ ) {
        $\mathbf{w}_i^{old} \leftarrow \mathbf{w}_i$;
        $\mathbf{w}_i \leftarrow Train(\mathbf{w}_i)$;
        if $m = 1$ or $i = n$ then
            $\Delta\mathbf{w}_i \leftarrow \mathbf{w}_i$;
        end
        $\Delta\mathbf{w}_i \leftarrow GetIncrement(\mathbf{w}_i, \mathbf{w}_i^{old})$;
    }
    $\{\mathbf{w}_1, ..., \mathbf{w}_N\} \leftarrow Combine(\{\Delta\mathbf{w}_1, ..., \Delta\mathbf{w}_N\})$;
}
Output: $\{\mathbf{w}_1, ..., \mathbf{w}_N\}$
```

without removing the common information may result in double counting. We assume that the global estimate from the previous merge is the only common information, and empirically demonstrate this assumption holds in section 3.5.

From the theory of conflations for normal distributions (P. Hill, 2008), the combined estimate of the weights, w, given independent datasets D_1 and D_2 can be written as:

$$f(w|D_1 \cup D_2) \approx \&[f(w|D_1), f(w|D_2)] = f(w|D_1)f(w|D_2) \tag{3.14}$$

Suppose any resultant weight from additional training after the map merge (with PDF $f(w|D_M \cup D_N)$) can be approximated as the conflation between the previous weight of the merged map (with PDF $f(w|D_M)$), and a normally distributed increment, with PDF $f(w|D_N)$, resulting from the new data.

$$\begin{aligned} f(w|D_M \cup D_N) &\approx \&[f(w|D_M), f(w|D_N)] \\ &= f(w|D_M)f(w|D_N) \end{aligned} \tag{3.15}$$

From eq. (3.13) and eq. (3.15), we can arrive at the parameters for the weight increment estimates, given the new data as:

$$\Delta w \sim \mathcal{N}\left(\frac{(\sigma_M^2 + \sigma_N^2)\mu_{M\cup N} - \sigma_N^2 \mu_M}{\sigma_M^2}, \frac{\sigma_{M\cup N}^2 \sigma_M^2}{\sigma_M^2 - \sigma_{M\cup N}^2} \right) \tag{3.16}$$

Where $\Delta w \sim \mathcal{N}(\mu_N, \sigma_N)$ is the estimate of the increment of weight from the new data, $w_M \sim \mathcal{N}(\mu_M, \sigma_M)$ is the global weight estimate from the previous merge, and $w_{M\cup N} \sim \mathcal{N}(\mu_{M\cup N}, \sigma_{M\cup N})$ is the local estimate after further training on a copy of the previous global estimate.

Unless an agent has not used its sub-map for a merge before, it stores a copy of the global weights from the previous fusion until the next fusion. At the next fusion, each local estimate is compared to the copy of the previous global estimate, and estimates of the increment of weight are found and used in the merging. Algorithm 2 outlines the repeated merging process, where *GetIncrement* returns the estimated increment of weight, *Train* trains the map model with additional data starting with the inputted parameters, and *Combine* merges the weights assuming there is no common information.

3.4.2 Filtering Out Uncertain Local Estimates of Weights

As there are physical coordinates associated with the features, if there are no Lidar scans close to the feature coordinates used to train the Fast-BHM, the weights assigned to those features will have parameters very close to the initial values. We can filter out local estimates of weights that are very uncertain, by defining a threshold for the variance of a weight. When several sub-maps are combined together, local estimates of weights with variances below the threshold are used for the fusing to obtain the global estimates. For any particular weight in the vector of weights, if there is only one reasonably confident local estimate, the global estimate of that weight will have the same mean and variance of the local estimate with acceptable variance. If none of the local estimates for a particular weight is below the threshold, the global estimate for the weight will be the default mean and variance values.

3.5 Experiments

We conduct experiments to empirically study:

1. The relative performance, and training time required, for Bayesian Hilbert Maps (BHM) and Fast-BHM

2. The quality of Fast-BHMs trained on separate scans of the environment, then fused together using the scheme described in section 3.4
3. Whether errors that occur during fusion will accumulate as local maps are repeatedly fused,
4. The performance of Fast-BHMs relative fusion methods that require discretisation of a continuous representation, when the size of information transmitted is constrained.

Experiments are conducted on the Intel Dataset (Hahnel et al., 2003), the Freiburg Campus Dataset (Howard and Roy, 2003), and a simulated dataset of two robots in an indoor environment. Unless stated otherwise, we use 5600, 9408, and 4900 features respectively when experimenting with the aforementioned datasets. We use the Area under Receiver Operating Curve (AUC) as the performance measure (Bradley, 1997).

3.5.1 Fast-BHM vs BHM

Experiments were conducted to investigate the relative performances of Fast Bayesian Hilbert maps (Fast-BHM) and Bayesian Hilbert Maps. The results of the performance and run-time, of a Python implementation of both models, are tabulated in table 3.1.

Table 3.1: Each map representation with the performance measure, and the time needed to train on 90% of the scans in the Intel Dataset (Hahnel et al., 2003)

	Fast-BHM	**BHM**
AUC	0.95±0.04	0.96±0.03
Training Time	7 min	339 min

We see that the time required to train up a Fast-BHM is significantly less than that needed for a BHM, while the performance of the Fast-BHM is only marginally weaker, as measured by the area under the Receiver Operating Characteristic (ROC) curve. The training times were achieved by running python implementations on a typical desktop computer, and demonstrate the significant relative speed-up Fast-BHMs achieve. Being able to create maps quickly is crucial for autonomous robots in the field. As a covariance matrix of weight parameters is no longer kept, the size of information needed to be communicated required between each agent to define the model is also smaller for Fast-BHMs.

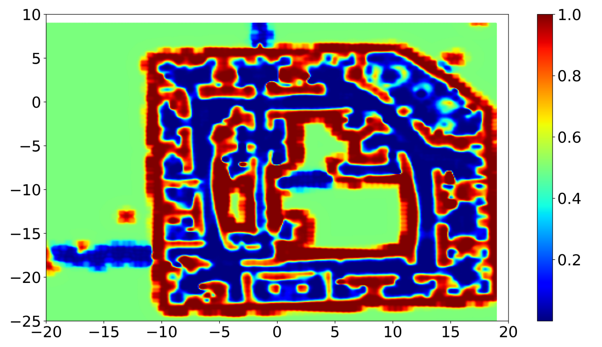
3.5.2 Fusion of Fast-BHM

Experiments were conducted to evaluate the performance of our fused global map. A baseline to compare the performance of our fusion scheme is a single Fast-BHM trained on all the data gathered by the individual robots. This baseline value would be the best possible result achievable, indicating that no information was loss or misinterpreted during the fusion. As assumptions and approximations were made during the fusion process, we would expect a slightly lower performance. We also wish to compare a global map obtained from multiple fusions, with one from just a single fusion, and determine whether the errors due to approximations and assumptions made during each merge would accumulate, or be corrected by subsequent training.

We split the scans in the Intel Dataset (Hahnel et al., 2003) into training and test sets at a ratio of 9:1. Using the training set, the average x and y coordinates of each data point in the scan is calculated. Based on the average coordinates, each scan is categorised to be in one of the following quadrants: Upper-Left; Upper-Right; Lower-Left; Lower-Right. The split for the Intel dataset (Hahnel et al., 2003) is illustrated in fig. 3.1. This is equivalent to running individual robots in those four quadrants. A separate Fast-BHM for each quadrant is then trained. As the split was made considering the average of all the data-points in a scan, there will be data-points that overlap with scans belonging in other quadrants. In the repeated fusion experiment, sub-maps of each quadrant are merged together upon completing training on 25%, 50%, 75% and 100% of the training data. For obtaining a map after a single merge, the sub-maps of each quadrant are merged when 100% of training has been used. The Freiburg Campus dataset (Howard and Roy, 2003) was treated similarly, but with training and testing sets split at a ratio of roughly 4:1. The training and testing sets for the simulated dataset were split at a roughly 9:1 ratio. Scans from the simulated did not need to be divided into quadrants, as the simulated scenario was two robots collecting scans.

Table 3.2 compares the Area Under the Curve (AUC) metric of the different global map models obtained by querying the test set. The accuracies of the fused maps were high across all datasets examined, indicating that assumptions and approximations made during fusion were sound, and the fusion method proposed is well suited to fuse local Fast-BHM models. If we compare the performances of a global map obtained from a single merge and that from repeated merges. We see that repeated merging results in a global map that performs approximately as well as a map from only one merge. This indicates that errors from merges do not accumulate, as further training on a global map will help amend errors in the global estimates.

Figures 3.2, 3.3, and 3.4 show the resultant merges of local sub-maps and Fast-BHMs trained on data from all the agents. There are close to no visual differences in the merged and trained global Fast-BHMs. It can be seen that our method to conduct map fusion can produce a global map very similar to a map obtained by training the union of all the separate data scans. The conflation of local weight estimates has proven to be a reasonable approximation for the global weight values.


(a) Global Fast-BHM from merger of sub-maps

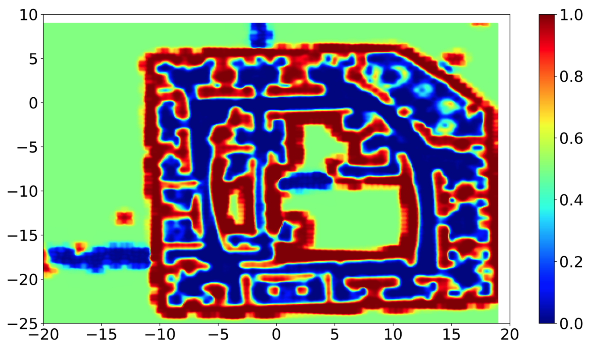
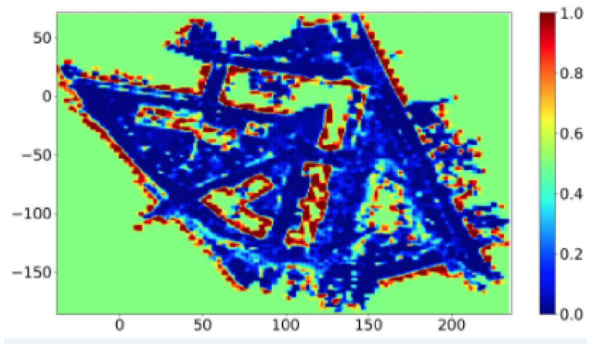

(b) Single Fast-BHM trained on training set

Figure 3.2: Merged vs entirely trained Global Fast-BHM representations (Intel Dataset)


(a) Global Fast-BHM from merger of sub-maps

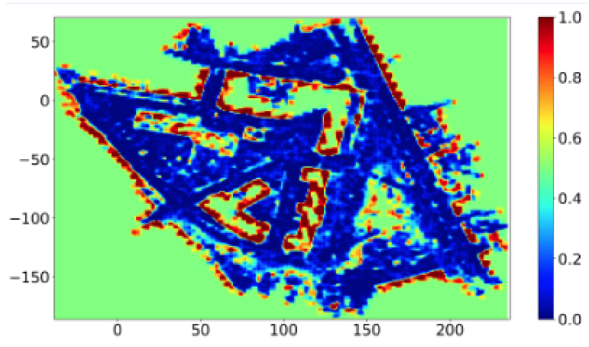
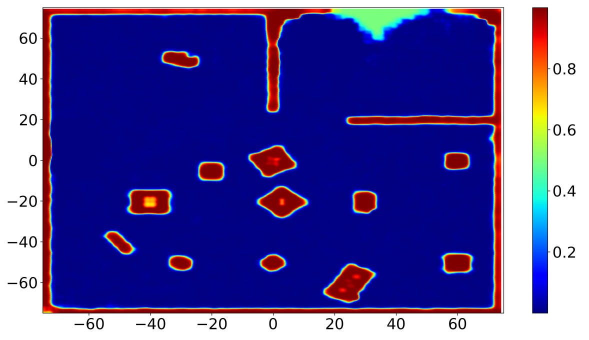

(b) Single Fast-BHM trained on training set

Figure 3.3: Sub-maps trained on scans from the Freiburg Campus dataset

3.5.3 Continuous vs Discrete Fused Map Representation

The result of our fusion scheme is a continuous Fast-BHM model, that can be queried at any valid coordinate. Whereas, previously proposed methods (Kim and Kim, 2014) (Jadidi


(a) Global Fast-BHM from merger of sub-maps

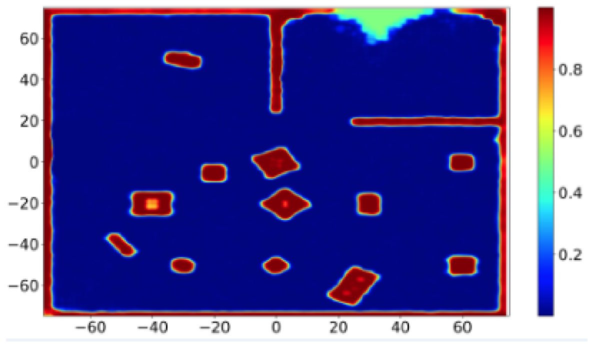

(b) Single Fast-BHM trained on training set

Figure 3.4: Merged vs entirely trained Global Fast-BHM representations (Simulated Environment)

Table 3.2: Area under ROC curve of both global maps obtained from different methods, and different datasets

Dataset	**Fusion Type**	**AUC**
Intel	No Fusion	0.95±0.03
	Fuse Once	0.94±0.04
	Repeatedly Fused	0.94±0.03
Freiburg Campus	No Fusion	0.97±0.02
	Fuse Once	0.89±0.090
	Repeatedly Fused	0.90±0.03
Simulated	No Fusion	0.99±5e-4
	Fuse Once	0.99±5e-4
	Repeatedly Fused	0.99±5e-4

et al., 2014) (Doherty et al., 2016) looked into merging the output predictions from separate models rather than fusing the model itself. Such methods require an *a priori* selection of points to query. In particular, the method proposed by Doherty et al. (2016) requires a grid map to be defined and used to sample a continuous occupancy representation, such as a Hilbert Map, discretising the global map, and a fusion update equation is used to merge outputs of Hilbert Maps. We shall compare this approach to our map fusion approach.

Continuous Fast-BHM models are more compact than grid cell representations. We want to explore the performances of continuous and discretised occupancy maps when the amount of information allowed to be communicated to each model is restricted. Using the Intel (Hahnel et al., 2003) dataset, We incrementally build a map representation with the

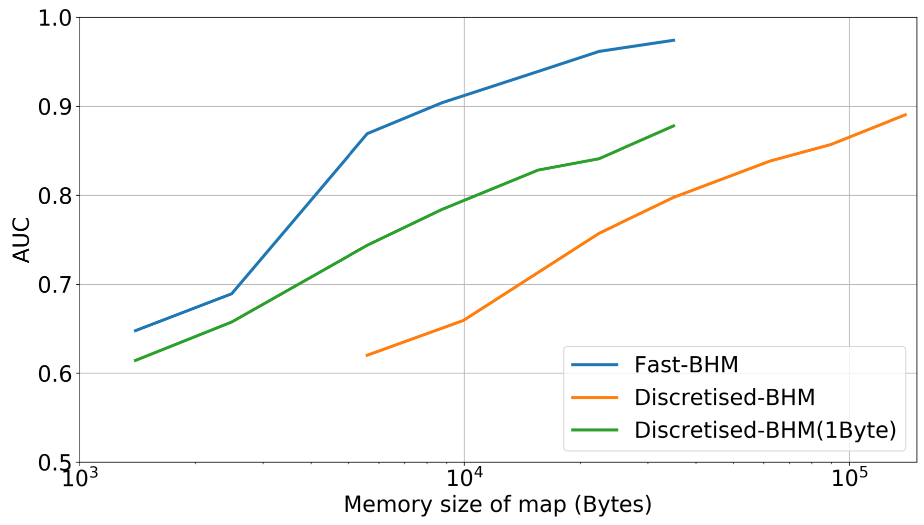


Figure 3.5: AUC of the global estimate with constrained communication (Intel Dataset)

Table 3.3: The performance of our proposed method on merging sub-maps using the Intel (Hahnel et al., 2003) dataset, using 5600 features, and the performance of method from (Doherty et al., 2016), with a square cell size of 10cm

	AUC	**Precision**	**Recall**
Proposed Method	0.936±0.03	0.819±0.03	0.919±0.03
Method from (Doherty et al., 2016)	0.916±0.02	0.864±0.03	0.866±0.15

fusion method described in (Doherty et al., 2016) on the outputs from Fast-BHMs, each using 10000 features, then discretising with differing cell sizes. We also explore using our proposed method to fuse sub-maps of Fast-BHMs, with different numbers of features. We can then plot the size of the map representation against the AUC measure of the fused map, as shown in fig. 3.5. We can see that Fast-BHMs outperform the discretised grid maps at any given memory size considered. This is true for both occupancy grid cells that used a floating point to denote the probability, and an efficient implementation of grid maps (Quigley et al., 2009), which stores an approximation of the probability in 1 Byte. The compactness of Fast-BHMs is particularly of use when the amount of information we can communicate is limited.

The performance of both methods on merging sub-maps using the Intel (Hahnel et al., 2003) dataset is detailed in table 3.3. Although the method from (Doherty et al., 2016) has a higher precision, our method has a higher recall and a higher overall AUC measure. Although in this case, both methods have similar performance, Fast-BHM is roughly 44.8

KB in size, while the grid map has a size of 560 KB.

3.6 Summary

This chapter introduces Fast-BHMs (Fast-BHMs) as a significantly sped-up version of Bayesian Hilbert Maps (BHMs), and presents a method to merge Fast Bayesian Hilbert Maps in a multi-robot scenario. Our merging method utilises the distributions of parameters to produce a unified global model, which is itself a Fast-BHM. We empirically demonstrate that Fast-BHMs are significantly faster than BHMs. We then demonstrate that our map fusion method is able to produce accurate global estimates, which are compact in size and can be easily transmitted when communication is limited.

In the next chapter, we extend beyond representing the occupancy of the environment and develop learning methods to continuously represent the distribution of movement directions in an environment over time. This representation allows us to expand beyond static environments, and reason about how dynamic objects in the environment move.

Chapter 4

Continuous Spatiotemporal Maps of Motion Directions

4.1 Introduction

Robots need a representation of the environment to operate autonomously. Building static maps that capture the occupancy of an environment without considering the dynamic objects is well-studied in robotic mapping. However, simply mapping occupancy in a static environment, using for example standard occupancy grid maps (Elfes, 1987), does not adequately capture the dynamics in real-world environments. Robots, in particular autonomous vehicles, often need to operate in urban environments with moving objects, such as crowds of people and other moving vehicles. The understanding of movement directions can shed light on representing the trajectories of these dynamic objects in the environment. This work addresses the problem of understanding the trajectories of dynamic objects by building a directional probabilistic model that is continuous over space and time. The method provides a probability distribution over possible directions, which includes angular uncertainty, making our method a valuable tool for safe decision-making. With the proposed method, we can predict the possible directions a dynamic object would take in a given location at a given time. Then, we can use the predicted directional uncertainty to generate trajectories

This chapter has been published in IEEE RA-L as Zhi et al. (2019b).

that dynamic objects typically take. Knowledge of such typical trajectories sheds light on long-term dynamics in an environment, improves path planning (Chiang et al., 2017), and can plausibly be applied in autonomous driving in cities (Pomerleau et al., 2014).

Recent advances in building maps for trajectory directions (Senanayake and Ramos, 2018) have been able to capture the multi-modality of uncertainty distributions associated with predictions of directions. However, grid map based methods, such as the recently presented method of (Senanayake and Ramos, 2018), require the environment to be discretised into independent grid cells specified *a priori*, and interaction occurs between data points in neighbouring cells. This assumption is unrealistic and a significant impediment to the performance, as in the real world directional flows are largely continuous. In this work, we present a method to build continuous spatiotemporal maps, overcoming these limitations. This is a more realistic model of the real world, as in the natural world, movement directions are not split into uniform grids. The continuous representation provides significant improvements in performance. Taking inspiration from advances in continuous occupancy mapping (Ramos and Ott, 2016), we utilise approximate kernel matrices to project data into a high-dimensional Reproducing Kernel Hilbert Space (RKHS) to build scalable continuous maps of directional uncertainty. Our proposed method does not require discretisation of the environment, and is able to be queried at arbitrary resolution after training. Most previous attempts to build maps of directions have assumed that the distribution of directions at a selected coordinate in space is stationary in time, or model how point values change over time. There have been methods to model periodic spatiotemporal changes in occupancy (Krajník et al., 2017), or model temporal patterns with directions as discrete values (Mellado et al., 2018). Our method extends spatiotemporal representations to beyond a scalar value, to allow for multi-modal distributions as outputs.

Motivated by the limitations discussed above, we develop a method to build a map of directional uncertainty that addresses the drawbacks mentioned. We generate spatiotemporal kernel features, then pass the features into a Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) network, and use the hidden representation to train a mixture density network (Bishop, 1994) of directional von Mises distributions. The proposed method builds a model with the following desirable properties:

1. models the probability distribution of movement directions over a valid support of $[-\pi, \pi)$, and can be multi-modal;

2. represents the environment in a continuous manner, without assuming the discretisation of the environment into a grid of fixed resolution, with independent cells;

3. takes into account how the probability distribution of directions changes over time, extending the map to become spatiotemporal.

4.2 Related Work

The method provided in our work is aimed at learning the long-term dynamics of an environment. There have been attempts (Arbuckle et al., 2002) to extend occupancy mapping beyond static environments by storing occupancy signals over time, and then building a representation along time in each grid. However, these methods are memory intensive and assume that time can be discretised into slices. Further extensions (Kucner et al., 2013; Ambrus et al., 2014; Tanzmeister et al., 2014) of learning how occupancy changes over time have also included various improvements to predicting changing occupancy maps over time.

Instead of modelling a changing occupancy map over time, methods (O'Callaghan et al., 2011; McCalman et al., 2013) have also been developed to understand long-term occupancy by modelling trajectories. O'Callaghan et al. (2011) uses Gaussian Process learning to infer the direction an object takes in different areas of the environment. A major drawback of this method was its inability to represent multiple directions. McCalman et al. (2013) uses filtering with Kernel Bayes Rule, and determines the components of a mixture of Gaussian distributions. Both of these methods do not "wrap" around a circle, and can predict differing probability densities at $-\pi$ and π. As O'Callaghan et al. (2011) and McCalman et al. (2013) apply non-parametric models, the scalability of these methods is limited. Recently, a parametric directional grid maps approach (Senanayake and Ramos, 2018) was proposed to model directional uncertainty in grid cells as a mixture of von Mises (VM) distributions, which capture multi-modal distributions, and have a support of $[-\pi, \pi)$. The authors showed that the directional grid maps method outperforms the Gaussian Process method presented in O'Callaghan et al. (2011). The directional grid maps method (Senanayake and Ramos, 2018) relies on *a priori* discretisation of the environment into grid cells. This method allows the directional uncertainty in each grid cell to be learnt, which is assumed to take into account trajectories over different periods of time. However, the performance of this method may be hindered by the assumption of independent grid cells, and the assumption that the

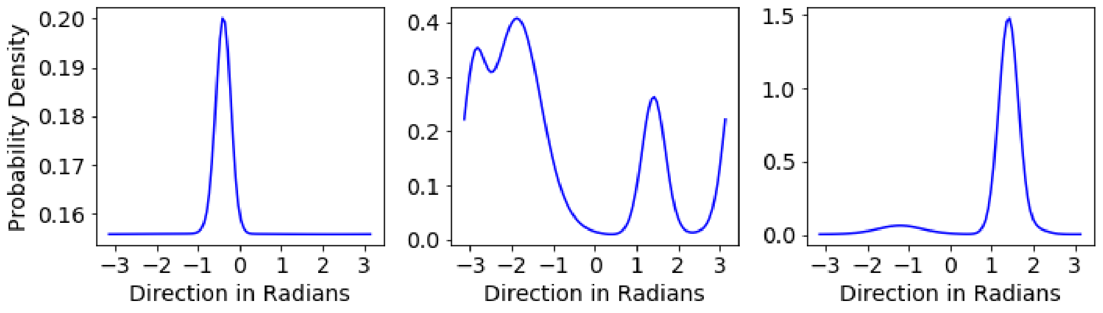


Figure 4.1: Example distributions of directions in different coordinates in space and time. The directions are in radians.

uncertainty is stationary over time. Both of these assumptions are relaxed in our proposed method.

A similar method of fitting semi-wrapped Gaussian mixture models using EM is also presented in Kucner et al. (2017). This method also attempts to achieve a "dense representation" that is continuous over space, by combining predictions from several cells via imputation methods. There have also been some recent attempts to model scalar values which can be periodic in time, such as occupancy (Krajník et al., 2017) or a direction category (Mellado et al., 2018), using the Fourier Transform. Our spatiotemporal model differs from these methods in that allows for multi-modal distributions as outputs.

4.3 Methodology

4.3.1 Problem Formulation

We aim to build continuous directional maps over space and time. Let us consider a dataset, denoted as $\mathcal{D} = \{(x_m, y_m, t_m, \theta_m)\}_{m=1}^{M}$, where x_m and y_m denote the longitude and latitude of the data point in space, t_m denotes the time step at which the m^{th} observation was made, and $\theta_m \in [-\pi, \pi)$ denotes the angular direction of a trajectory at this point. Given specific coordinates in space and time: (x^*, y^*, t^*), we want the model to give us the probability distribution of the angular directions $p(\theta|x^*, y^*, t^*)$. Essentially, our problem is to learn a mapping between a coordinate in space-time and a multi-modal distribution. Some example output distributions are shown in fig. 4.1.

4.3.2 Overall Architecture of Proposed Model

A brief overview of our model is given in this subsection, and important components of the model are discussed in the following subsections. Fig. 4.2 summarises the main architecture of our model. Given input data and a set of coordinates, known as inducing points, we calculate high-dimensional kernel features. The features are then treated as a sequence and passed through a LSTM network. The hidden representation of the LSTM network is inputted to a mixture density network (MDN) to capture a multi-modal distribution. Fully connected layers in the MDN jointly learn the parameters α, μ, and κ, and approximate a mixture of von Mises distributions. We can then evaluate the probability distribution function of trajectory direction uncertainty given an arbitrary coordinate. Note that the coordinates inputted are a 3-dimensional vector, that provides a coordinate in space and time. Previous methods of mapping direction, such as those in Senanayake and Ramos (2018) and O'Callaghan et al. (2011), assume that the target distributions of directions do not change in the map over time. If we make the same assumption, and only require a continuous directional map in space independent of time, the high-dimensional features over a set of coordinates, known as inducing points, over space are inputted directly into the MDN for training. In section 4.4, we will demonstrate the benefits of a spatiotemporal model over a purely spatial model.

4.3.3 Generating high-dimensional Features Over Space

high-dimensional kernel features over space encode the influence each training data point holds on predictions in neighbouring points in space. This allows us to achieve maps that are continuous over space. We generate a large number of features from the inputted data that approximate the covariance matrix between N data points, known as the kernel matrix, which is commonly used in kernel machines (Hofmann et al., 2008). In this work, we approximate a kernel by projecting data to a set of inducing points, similar to the sparse Gaussian Process method (Snelson and Ghahramani, 2006). Inducing points can be thought of as pseudo-inputs which input coordinates are compared against. Inducing points are the centre of basis functions which are used to construct the continuous function of output parameters. We operate on the covariance between our input data points and our inducing points, rather than the covariance between each data point.

We start by defining a set of M_s inducing points over space, denoted by $\bar{\mathbf{x}}_1 \ldots \bar{\mathbf{x}}_{M_s}$. In our

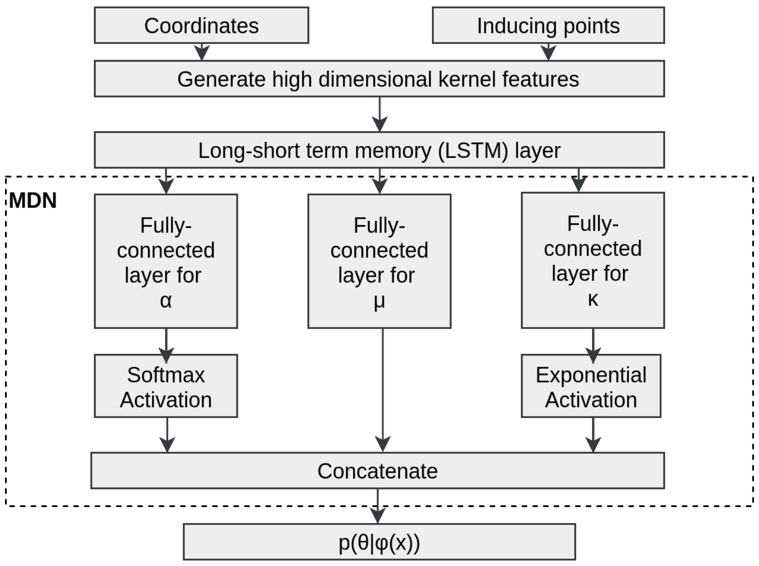

Coordinates
Inducing points
Generate high dimensional kernel features
Long-short term memory (LSTM) layer
MDN
Fully-connected layer for α
Fully-connected layer for μ
Fully-connected layer for κ
Softmax Activation
Exponential Activation
Concatenate
p(θ|φ(x))

Figure 4.2: The architecture of our model. high-dimensional approximate kernel features, generated from an inputted coordinate with predetermined inducing points, are passed through a LSTM network, followed by a MDN.

experiments, we place the inducing points with fixed spacing along longitude and latitude. $\mathbf{x} \in \mathbb{R}^2$ and $\bar{\mathbf{x}} \in \mathbb{R}^2$. We generate high-dimensional feature vectors from inputs by evaluating a kernel function, $k(\mathbf{x}, \bar{\mathbf{x}})$, between the input and each inducing point. We denote the high-dimensional features, with spatial inducing points, as $\boldsymbol{\phi}_s(\mathbf{x}) \in \mathbb{R}^{M_s}$.

Similar to Ramos and Ott (2016), the Radial Basis Function (RBF) kernel (Scholkopf and Smola, 2001) is chosen to generate features in space. This kernel function is defined as

$$k(\mathbf{x}, \bar{\mathbf{x}}) = \exp\left(-\frac{\|\mathbf{x} - \bar{\mathbf{x}}\|_2^2}{2\ell^2}\right), \tag{4.1}$$

where ℓ is the length scale hyperparameter of the kernel. Intuitively, ℓ controls how far the influence of a single training data point extends. Low ℓ values indicates "close". In the next section, we will extend our high-dimensional features to include time.

4.3.4 From Spatial to Spatiotemporal

This subsection is motivated by the need to extend our continuous spatial model into a continuous spatiotemporal model. There are two steps involved in extending or model to spatiotemporal:

1. Generate high-dimensional features in the space and time domains. This allows us to construct a sequence of spatial features over time;
2. Pass the spatiotemporal features into a LSTM network to learn a more compact representation of temporal variations and use this compact representation in our training

Spatiotemporal kernel features are needed to generate high-dimensional features that account for the influence of each training example in both space and time. A natural way to build spatiotemporal kernels, described by Flaxman (2015), is to multiply a spatial kernel with a temporal one. Similar to our method for spatial mapping, we assume M_t inducing points in time, $\bar{t}_1, \dots, \bar{t}_{M_t}$, which are evenly-spaced across a predefined range of time. The temporal kernels describe the correlation between the data and inducing points, from which we can generate a time series. The series can then be inputted into a LSTM network to learn further temporal dependencies. Our spatiotemporal kernel function is given by:

$$k\big((\mathbf{x}, t), (\bar{\mathbf{x}}, \bar{t})\big) = \underbrace{k_s(\mathbf{x}, \bar{\mathbf{x}})}_{\text{Spatial}} \underbrace{k_t(t, \bar{t})}_{\text{Temporal}}, \tag{4.2}$$

where $\mathbf{x}$ and t represent a coordinate in space and time, and $\bar{\mathbf{x}}$ and $\bar{t}$ are inducing points in space and time respectively. We can calculate the temporal features over each data point as $\boldsymbol{\phi}_t \in \mathbb{R}^{M_t}$. We can then obtain the spatiotemporal feature of each data point as, $\boldsymbol{\phi} = \boldsymbol{\phi}_s \otimes \boldsymbol{\phi}_t$, where $\boldsymbol{\phi} \in \mathbb{R}^{M_s \times M_t}$. We also have different length scales for the spatial kernel function and the temporal kernel function, denoted by ℓ_s and ℓ_t respectively.

We input the spatiotemporal features into a LSTM network (Hochreiter and Schmidhuber, 1997), then the outputted hidden representations of the LSTM network. Neural networks with LSTMs have proven to be effective for learning problems related to sequential data, in particular time series (Gers et al., 1999). The key idea behind LSTMs is the use of memory cells which maintain their states over time and non-linear gating units that control information flowing in and out of these cells. The gating units typically include the input gate, output gate, forget gate, and memory cell. Each gate contains a non-linear transformation and a set of weights and biases. During training, the weights and biases of the gates are learnt from the data. Further details on LSTM networks can be found in Greff et al. (2017). In our method, the LSTM is able to learn a compact hidden representation of the spatiotemporal kernel features, which is a vector of the dimensionality of the specified output of the LSTM. A mixture density network is subsequently trained on the hidden representation of the LSTM.

4.3.5 Mixture Density Networks

We expect the output of our model to be a probability distribution of movement directions, with each mode in this distribution indicating a probable trajectory direction. A probability distribution over the directions also sheds light on the uncertainty and the relative frequencies of occurrence of each associated direction. We can achieve such outputs using mixture density networks (MDN) (Bishop, 1994). A MDN allows our model to output parameters of a multi-modal mixture of distributions. Fig. 4.1 shows the probability distributions of directions at three different points in space and time. Note that as the distribution is bounded between $-\pi$ and π radians, while the integral of the probability density function is equal to 1, the probability density at a given point can have a value greater than 1.

The probability density of the target direction is represented as a convex combination of R individual probability density functions, which we will refer to as components. We denote the r^{th} component as $p_r(\theta|\boldsymbol{\phi}(\mathbf{x}, t))$, where $\boldsymbol{\phi}(\mathbf{x}, t)$ denotes the high-dimensional spatiotemporal feature vector, and θ represents the target angular directions. We can then take a linear combination and write the likelihood as

$$p(\theta|\boldsymbol{\phi}(\mathbf{x}, t)) = \sum_{r=1}^{R} \alpha_r p_r(\theta|\boldsymbol{\phi}(\mathbf{x}, t)). \tag{4.3}$$

The parameters α_r, known as mixing coefficients, are the weights of each component in the MDN. For a dataset with N inputted data points, $\{(\mathbf{x}_n, t_n, \theta)\}_{n=1}^{N}$, which are assumed to be independent and identically distributed, we can write the joint likelihood as the product of the likelihood for each data point. The joint likelihood is

$$\begin{aligned} p(\theta_1, ..., \theta_N|\{\boldsymbol{\phi}(\mathbf{x}_n, t_n)\}_{n=1}^{N}) &= \prod_{n=1}^{N} p(\theta_n|\boldsymbol{\phi}(\mathbf{x}_n, t_n)) && (4.4)\\ &= \prod_{n=1}^{N} \sum_{r=1}^{R} \alpha_r p_r(\theta_n|\boldsymbol{\phi}(\mathbf{x}_n, t_n)). && (4.5) \end{aligned}$$

We assume that the probability density function of angular directions of trajectories at a particular coordinate can be approximated by a mixture of von Mises distributions (Mardia, 1975). For the r^{th} component, the distribution has parameters $\mu_r\big(\boldsymbol{\phi}(\mathbf{x},t)\big)$ and $\kappa_r\big(\boldsymbol{\phi}(\mathbf{x},t)\big)$. These are known as the mean direction, and concentration parameters respectively. For brevity, we henceforth denote the mean direction and concentration of the r^{th} component, with respect to the inputted high-dimensional features, as simply μ_r and κ_r. The probability of a single component distribution is then

$$p_r(\theta|\boldsymbol{\phi}(\mathbf{x},t)) = \mathcal{VM}(\theta|\mu_r,\kappa_r) = \frac{\exp\big(\kappa_r \cos\big(\theta-\mu_r\big)\big)}{2\pi\mathcal{J}_0\kappa_r}, \tag{4.6}$$

where $\mathcal{J}_0$ is the 0^{th} order modified Bessel function (Mardia, 1975).

The von Mises distribution can be viewed as an approximation of a wrapped Gaussian distribution, around the range $[-\pi,\pi)$. The mean direction of the von Mises is analogous to the mean of the Gaussian distribution, and the concentration can be viewed as the reciprocal of the variance in a Gaussian distribution.

4.3.6 Learning Details

The loss function of the MDN is the average negative log-likelihood (ANLL) over all the training examples, given by:

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}\log\Bigg(\sum_{r=1}^{R}\alpha_r\mathcal{VM}(\theta_n|\mu_m,\kappa_m)\Bigg). \tag{4.7}$$

We can then use a neural network to optimise α_r, μ_r, and κ_r, by minimising the loss function. The mixing coefficients must satisfy $\sum_{r=1}^{R}\alpha_r = 1$, which can be enforced by applying a softmax activation function,

$$\alpha_r = \frac{\exp(z_r^{\alpha})}{\sum_{r=1}^{R}\exp(z_r^{\alpha})}, \tag{4.8}$$

where z_r^{α} denotes the network output corresponding to α_r. To enforce κ to be non-negative, an exponential activation function $\kappa_r = \exp(z_r^{\kappa})$ is applied to the network outputs.

In practice, the number of components, R, is determined prior to training. The larger the R chosen, the more expressive the approximated distribution becomes. However, a larger R also results in a larger number of parameters to learn, requiring more training data.

Due to the relatively high-dimensional outputs of MDNs, overfitting may easily occur when there are relatively few training examples. Overfitting in MDNs is discussed in detail in Hjorth and Nabney (1999). A straightforward method to combat overfitting in MDNs is to regularise the weights in the MDN, using, for instance, $L2$ regularisation. We can rewrite our regularised loss function as,

$$\mathcal{L}_{reg} = \mathcal{L} - \lambda_\alpha \|\mathbf{w}_\alpha\|_2^2 - \lambda_\mu \|\mathbf{w}_\mu\|_2^2 - \lambda_\kappa \|\mathbf{w}_\kappa\|_2^2, \tag{4.9}$$

where λ_α, λ_μ, $\lambda_\kappa \geq 0$ are regularisation coefficients, and $\mathbf{w}_\alpha$, $\mathbf{w}_\mu$, $\mathbf{w}_\kappa$ are neural network weights of the mixing coefficients, mean directions, and concentrations respective.

The neural network architecture used in our experiments, with the specific number of units of each layer is shown in table 4.1. M_s and M_t denote the number of spatial and temporal inducing points respectively. This is dependent on the dataset used. R denotes the number of von Mises distribution components in our output and was set to 5 for all the experiments.

Table 4.1: The architecture of the spatiotemporal network used in our experiments. LSTM refers to a long short-term memory layer and FC refers to a fully connected layer. M_s and M_t are hyperparameters which denote the number of spatial and temporal inducing points respectively. R denotes the number of von Mises distribution components in the output.

Input (Size = $M_s \times M_t$)		
LSTM (Units = $40R$)		
FC (Units = R, α)	FC(Units = R, μ)	FC (Units = R, κ)
Softmax Activation		Exponential Activation
Concatenate		

4.4 Experiments and Discussions

In this section, we investigate the behaviour of our proposed model on both simulated and real trajectory datasets. To evaluate the performance of our method, we run 10-fold cross-validation, splitting our data into training and test sets at a 90:10 ratio for each fold. We shall make use of Minimum Rotation Error and Average Likelihood metrics:

Minimum Rotation Error (MRE): This metric gives the difference, in radians, between the ground truth direction and the mean of the nearest von Mises (VM) component. MRE gives a simple and relatable metric for real-world problems. However, it does not consider the mixing coefficients or the uncertainties of the VM distributions, nor does it consider components other than the nearest one to the ground truth. MRE over all of our test data is given by:

$$MRE = \frac{1}{N'} \sum_{n=1}^{N'} \min_{r} |\theta_n - \mu_r^*|, \tag{4.10}$$

where θ_n denotes the label of test example n in a test set of N' points in total, and α_r^*, μ_r^*, and κ_r^* are predicted parameters of component r in the mixture of R distributions. The lower the value, the smaller the error between the ground truth and the mean of the nearest VM component.

The Average Likelihood (AL): This metric is able to capture the uncertainty and multi-modality in angular distributions. This is equivalent to the *Average Probability Density* metric in Senanayake and Ramos (2018). When measuring the performance of models, models with similar MRE can have significantly differing AL. Over all of our test data, AL is given by,

$$\text{AL} = \frac{1}{N'} \sum_{n=1}^{N'} \sum_{r=1}^{R} \alpha_r^* \mathcal{VM}(\theta_n | \mu_r^*, \kappa_r^*)), \tag{4.11}$$

The higher the AL value, the better the test data fit the approximated mixture. Note that the probability density at a point can be greater than 1.

In all of our experiments, the inducing points are positioned in a grid manner in space and time. Neighbouring inducing points in each axis have a fixed distance between them, and all training and test data points lie within a regular grid of inducing points. In our experiments, the regularisation coefficients are chosen to be $\lambda_\alpha = 0.001$, $\lambda_\mu = 0$, $\lambda_\kappa = 0.001$. We use a standard desktop computer with an Intel Core i7 microprocessor and 32 GB of RAM to conduct all experiments. The datasets used in our experiments are listed in Table 4.2.

For the Pedestrian 1 and 2 datasets, we conduct experiments using different numbers of inducing points. If we convert the temporal axis of all time-stamped datasets to use 0.1 hours as units, we position inducing points in a $1 \times 1 \times 1$ lattice when using Pedestrian 3, a $5 \times 5 \times 1$ when using Edinburgh, and a $20 \times 20 \times 2$ for both Lankershim and Peach Tree St datasets.

Table 4.2: Description of datasets

Dataset	Temporal	Description of the dataset
Pedestrian 1 (Senanayake and Ramos, 2018)	No	Simulated unimodal.
Pedestrian 2 (Senanayake and Ramos, 2018)	No	Simulated multimodal.
Pedestrian 3	Yes	Simulated multimodal.
Edinburgh (Majecka, 2009)	Yes	Real world pedestrian trajectories collected on 24 August.
Lankershim (Federal Highway Administration, 2007a)	Yes	A large real world traffic dataset on Lankershim Boulevard, USA.
Peach Tree St (Federal Highway Administration, 2007b)	Yes	A large real world traffic dataset collected on Peach Tree Street.

4.4.1 Continuous and Discrete Map Representations

A major benefit of the proposed method is that the resulting maps are continuous over space and time. Removing the need to discretise the environment into independent cells is not only a more natural way to represent directions, it also facilitates building directional maps at arbitrary resolution. To investigate the benefits of continuous mapping, we conducted two experiments:

1. We compare the performance of the discrete mapping methods, and dense variants of these methods, against our method at various resolutions (Fig. 4.3);
2. the performance of the discrete mapping methods against our method when we only have 5% of the original dataset used (Table 4.3).

We compare the performance of our continuous mapping method with the two discrete directional grid map methods presented in Senanayake and Ramos (2018). One of the methods presented fits a von Mises distribution to each cell individually, the other fits a mixture of von Mises distributions after finding reasonable initialisations via clustering.

Similar to our continuous method, dense representations similar to the method described in Kucner et al. (2017) also allow for arbitrary resolution of the map, by providing a smooth transition from one cell to the next. We make comparisons against dense variants of grid-based methods, similar to the method presented in Kucner et al. (2017), but using mixtures of VM distributions instead of the Wrapped Gaussian Mixture Model (Kucner et al., 2017). Rather than segmenting the environment into independent cells, we want to lay overlapping circular cells over the environment. The same data point may lie within the radius of several

Table 4.3: Results on pedestrian datasets 1 (P1) and 2 (P2), trained on the full dataset (100%) and a random subset containing 5% of the data. MM and UM stand for the unimodal and multimodal discrete mapping techniques in Senanayake and Ramos (2018).

Data	Metric	Method	100%	5%	Change
P1	AL	Cont-	1.69±0.23	1.15±0.41	-0.54
		MM-Dis	1.54±0.04	0.60±0.14	-0.94
		UM-Dis	1.50±0.03	0.87±0.08	-0.63
	MRE	Cont-	0.08±0.01	0.16±0.09	+0.08
		MM-Dis	0.16±0.01	0.27±0.11	+0.11
		UM-Dis	0.16±0.02	0.24±0.09	+0.08
P2	AL	Cont-	1.44±0.38	1.14±0.45	-0.30
		MM-Dis	1.17±0.03	0.57±0.19	-0.60
		UM-Dis	0.86±0.02	0.46±0.02	-0.40
	MRE	Cont-	0.13±0.02	0.19±0.10	+0.06
		MM-Dis	0.29±0.02	0.74±0.18	+0.46
		UM-Dis	0.49±0.06	0.87±0.08	+0.38

cells, and be used in the training of several VM mixtures. When we query a coordinate, we impute the distribution by taking the linear combination of the VM mixtures within a fixed radius, with the weights determined by computing the RBF function applied on distances between the queried point and the centre of each relevant cell. This setup is similar to that in Kucner et al. (2017). In our experiments, the radius is the distance between the centre of one cell to neighbouring cells.

The objective of the first experiment is to understand how the proposed continuous mapping method compares to discrete mapping methods. The results of the first experiment are shown in Fig. 4.3. The number of grid cells / inducing points used is 20, 35, 130, 700, 2800, and 11200. The data density of each independent non-empty cell at each resolution is shown in Fig. 4.4. For the continuous method, we need to adjust the length scales, as good values for the length scale is dependent on the distance between inducing points. For the continuous method, we use a length scale, ℓ_s, of 3.0, 2.2, 1.0, 0.6, 0.4, and 0.3 respectively for both datasets with the aforementioned resolutions. We see that our continuous representation of directional maps outperforms previously proposed grid-based methods, when there is an equal number of inducing points and grid cells. All the methods perform well at 700 grid cells/inducing points. The benefit of continuous mapping is prominent when there is a small number of grid cells or inducing points representing the map. When there are only a

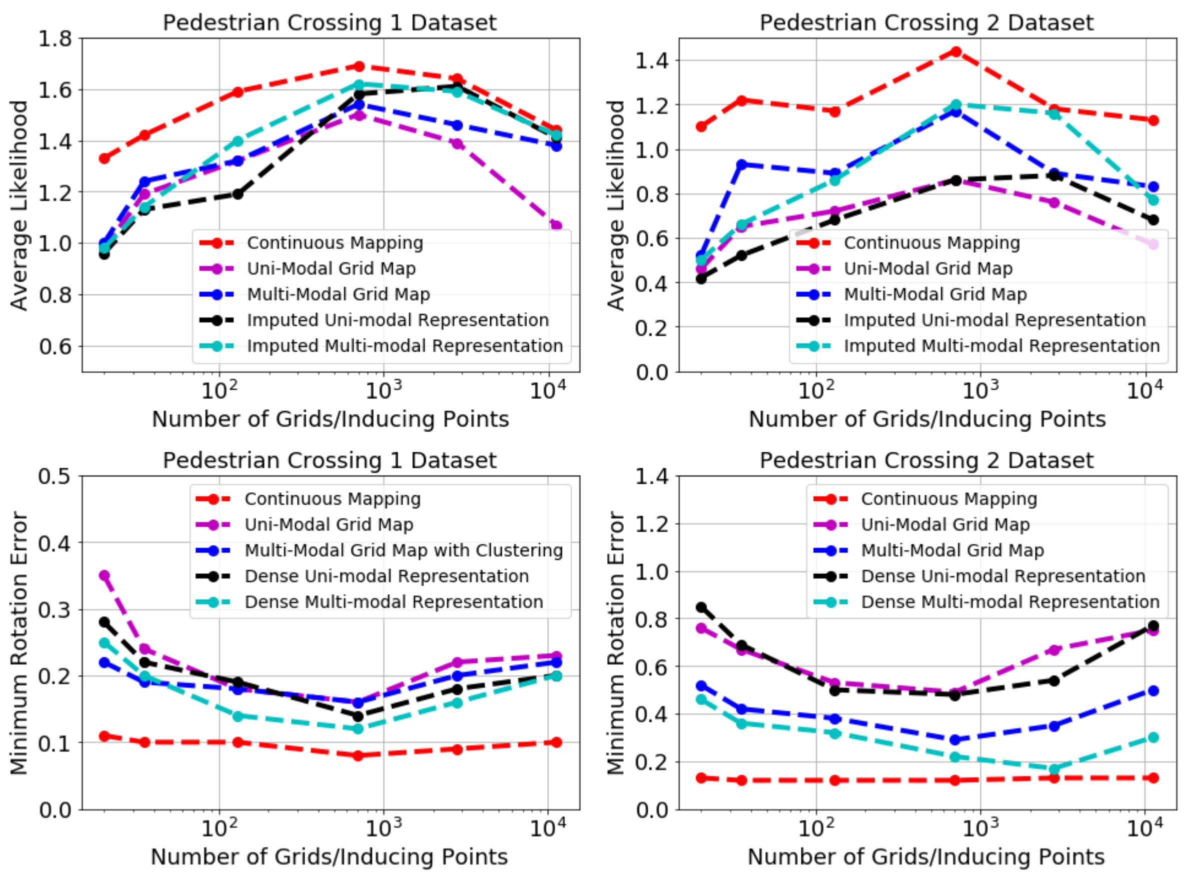


Figure 4.3: The performance of our continuous mapping method compared with the uni-modal grid-based method, and the multi-modal grid-based method with clustering, described in Senanayake and Ramos (2018), as well as dense variants similar to Kucner et al. (2017). We examine the effect of increasing the number of grid cells (for the discrete methods) and inducing points (for the continuous method).

few cells, the cell resolution is relatively low, and there may be directional data in several directions. However, the grid-based methods give each training example within the same cell equal weighting, and do not take into account the position of the data points within the cell. For example, a training data point with coordinates at the edge of cell holds the same weight as a data point positioned at the centre of the cell. The dense representations attempts to achieve outputs which are continuous, and provide slight improvements in performance relative to discretised grid cells. It factors in the predictions from several cells, and the distances between the training data points and the cell centres. However, during training it still suffers from not weighing in these distances. The continuous representation considers the distance

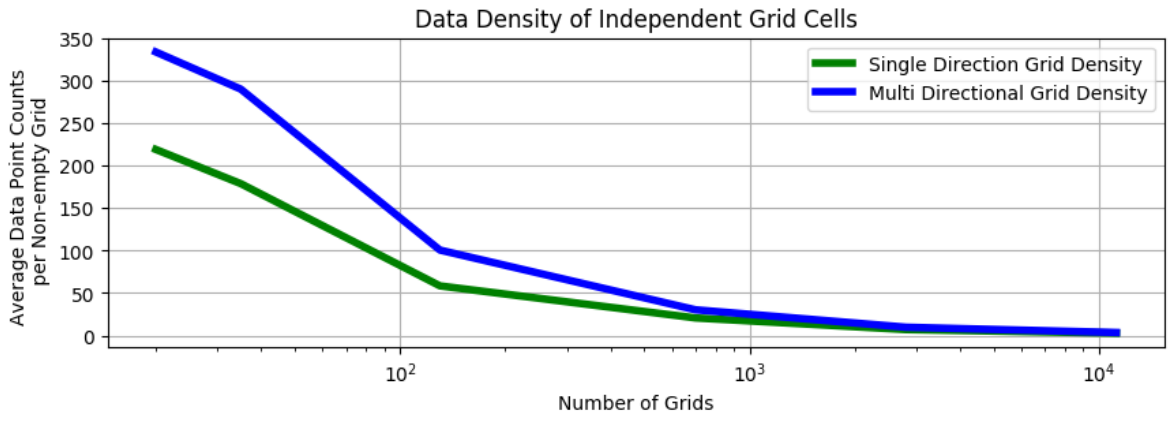
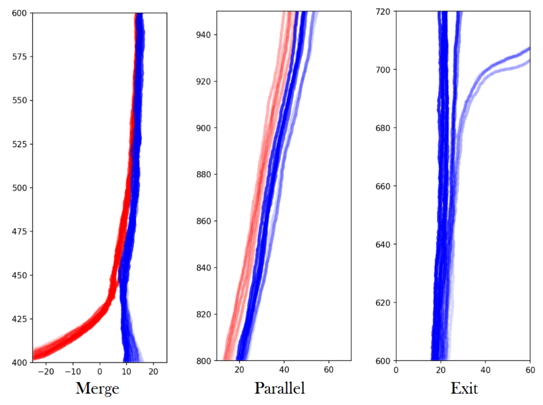


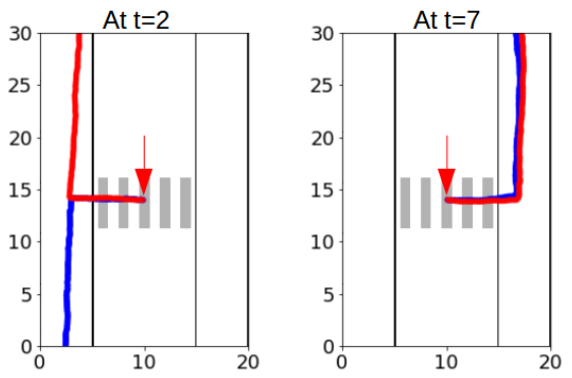
Figure 4.4: The average number of data points in every none empty independent grid cell, at different resolutions and P1, P2 datasets.

between each inducing point and the training coordinate, whereas both simple grid map and imputed dense models give equal weighting to data points in a certain range. The drawback of this is apparent – a training point very close to the cell centre is considered to have the same weighting as a point on the fringes of the cell. The continuous representation takes these distances into account via the RBF distances between a data point and predefined inducing points. As the number of grid cells increase over 700, the number of data points in each grid cell decreases such that there are too few data points in each cell. Likewise with the continuous method, when the number of inducing points becomes too large, the number of parameters in the neural network increase to a level that the performance decreases.

The second experiment evaluates the performance of discrete and continuous methods when there are few data points. The results of the second experiment which shows the sensitivity of each method to the amount of data are tabulated in Table 4.3. We trained each model at the resolution of, 700 inducing points/grid cells. We see that with 95% of the data points randomly removed and ten-fold cross-validation used to obtain the performance, the performance for all three methods deteriorate as expected. However, we see that the change of the continuous method is lower compared to the discretised methods. This is due to the continuous method considering all data points to have a degree of influence in the surrounding region. This makes more efficient use of training data, as a point that is considered in another cell in the discrete method would be considered during the training of the continuous model. This also results in the continuous method being more robust to outliers. In the discrete methods, when there are very few data points in a single grid cell, the effect of an outlier is significant, but the continuous model would be informed by data points that the discrete model treats as independent, and thus would have information from more data points.



(a) Generated trajectories from a model trained on traffic over sections of highway in Lankershim are shown. The vehicles move from bottom to top, with 16 such trajectories in each plot. (Left) vehicles enter the boulevard in two different directions and merge; (Centre) two separate lanes (red and blue) of the boulevard; (Right) trajectories diverge.



(b) Predicted trajectories at times $t = 2$ (left) and $t = 7$ (right). At each of the two times, we generate 2 trajectories which originate at the middle of the pedestrian crossing, indicated by the red arrows. Trajectories both start at the same location at the centre of the crossing, but at different times, they take different directions.

4.4.2 Generating Trajectories in Space

This subsection briefly outlines a method to visualise directional predictions from our model, by generating trajectories. As the time between steps is negligible relative to the time recorded in the data, we disregard the temporal variations to generate a long-term directional map which represents the "typical traffic patterns" at a given position in the environment. We can generate trajectories that originate from a particular coordinate anywhere in space by recursively sampling the distribution of directions and taking small steps in the sampled direction. Visualisation of trajectories, especially on traffic datasets, allows us to qualitatively evaluate the predictions of our model.

Given a particular starting coordinate (x_0, y_0), we query our continuous map to obtain $p(\theta|x_0, y_0)$. This answers the question of in which direction should an agent take next if the current position is (x_0, y_0). We sample this distribution to obtain a direction to move in, θ_0. We can then update the coordinates by taking an increment step, d, in the direction of

θ_0. We can take an Euler integration approach, and recursively predict the current position by sampling directions and computing $x_{n+1} = x_n + d\cos(\theta_n)$ and $y_{n+1} = y_n + d\sin(\theta_n)$ for N time steps. This results in a trajectory with coordinates $(x_0, y_0), \ldots, (x_N, y_N)$. We can generate different trajectories from (x_0, y_0) by taking various samples from $p(\theta|x_n, y_n)$. Plots of such generated trajectories with a model trained on the Lankershim dataset are shown in Fig. 4.5a. It can be seen that the trajectories consistently follow sensible paths, and the model is able to generate realistic trajectories that enter and exit the boulevard. Such predictions can plausibly be used for risk-aware decision-making in urban environment.

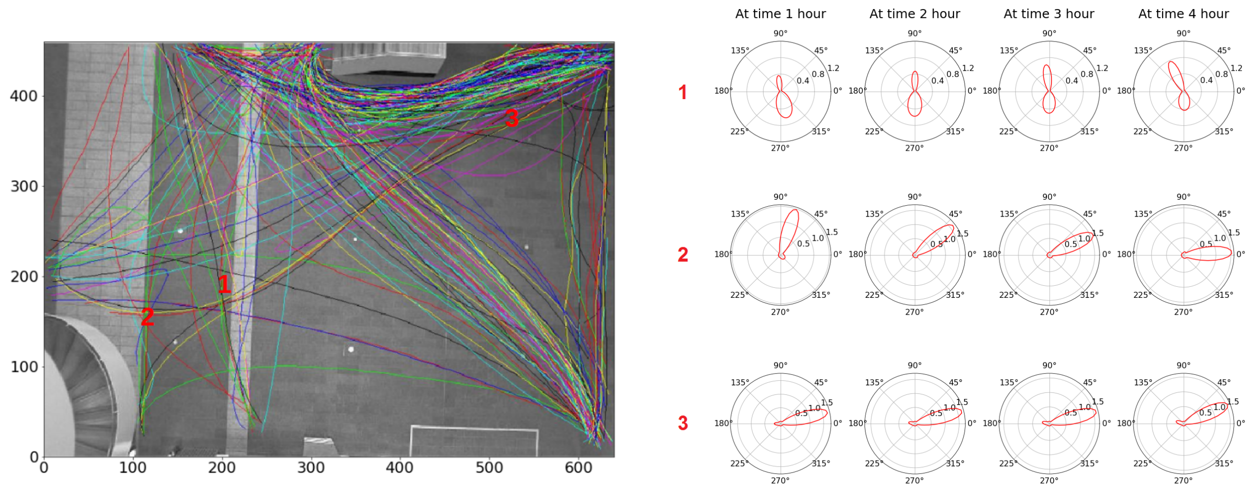


Figure 4.6: The left figure shows trajectories people take in the atrium of the Edinburgh university (Majecka, 2009). We attempt to model how the directional distribution change over time in three different locations. The predicted probability distributions over four time-steps at the three locations are shown on the right.

4.4.3 The Spatiotemporal Representation

Trajectories generated in Section 4.4.2 represent the typical paths dynamic agents such as vehicles and pedestrians take. Previous methods, such as those in O'Callaghan et al. (2011) and Senanayake and Ramos (2018), assume that the typical paths of objects do not change over time. However, in real-world environments, the general directions of trajectories change not only with space but also with time. An obvious example would be the different trajectory directions of people entering and exiting a building before and after an event.

The kernel function used for temporal features affects how smooth the interpolated functions between inducing points are. After experimenting with common stationary kernels, the RBF kernel was found to have good performance for generating temporal features. Fig. 4.5b shows trajectories generated at different points in time using a model trained on the pedestrian crossing 3 dataset, at the same starting coordinates $(10, 15)$. We see that at different times, generated trajectories will either end at the left footpath or the right. If the temporal trends are not considered, the distribution of directions on the pedestrian crossing would be largely bi-modal, with modes representing the left-to-right and right-to-left movements simultaneously. Such a representation is incorrect. In reality, as captured by the proposed spatiotemporal model, trajectories generated starting from the centre of the pedestrian crossing could end on the left side and right side of the road, depending on the time (Fig. 4.5b). Fig. 4.6 also illustrates directional probability density function change in time, predicted by a model trained on Aug 24 of the Edinburgh dataset, at three fixed points in space at four different time-steps.

Table 4.4: The AL values on several datasets, using both the spatial and spatiotemporal methods.

Dataset	Spatial	Spatiotemporal
Pedestrian Crossing 3	1.09±0.06	2.20±0.23
Edenburgh	0.38±0.02	0.38±0.02
Peach Tree St	1.34±0.04	1.40±0.09
Lankershim	1.70±0.08	1.75±0.15

To obtain a quantitative evaluation, we compare the performance of our continuous spatiotemporal model with a solely spatial continuous model on a range of datasets, and the results are tabulated in Table 4.4. The spatial continuous model directly trains a MDN on the kernel features over space. We see that our spatiotemporal models give respectable improvements in performance on most of the tested datasets, and perform at least as well as the spatial models on all of the tested datasets, despite having a lower density of training data points due to the additional time axis. The greatest improvement in performance was observed on the pedestrian 3 dataset, as it contains trajectories of completely different directions at different times (illustrated in the plots in Fig. 4.5b). We only use the AL metric, as MRE only considers the VM component with μ closest to the ground truth direction, even if it has a low α. Therefore, it does not penalise inaccurate parameter estimates in VM components other than the closest component to ground truth, nor does it consider the α values of each component. MRE is incapable of capturing nuanced changes in the concentration and mixing coefficients, which usually occur in the time domain. For example, the mean directions of pedestrian direction distribution on a footpath would largely remain unchanged throughout the day, but the mixing coefficients would vary significantly over time. MRE would not be able to accurately reflect the performance in this scenario.

4.5 Summary

We present a novel spatiotemporal model to learn the distribution of directions of moving objects. The proposed method is continuous in both space and time, and can be queried at arbitrary resolution. We make use of high-dimensional kernel features, LSTM networks, and a MDN of von Mises distributions. Our method is capable of capturing subtle temporal changes in the multi-modal directional distribution that cannot be adequately modelled with existing techniques. Although the datasets we used are limited to urban environments, the proposed continuous spatiotemporal model is exceedingly generic, and can be applied to other domains that involve mapping coordinates in space and time to a potentially multi-modal distribution. Additional improvements on the proposed method can revolve around conditioning the probability distribution of movement directions on the history of the object for path planning in dynamic environments.

We shall next move to part II, and explore learning for anticipatory navigation. Up next in chapter 5, we develop a framework which produces probabilistic predictions of how neighbouring agents move and accounts for these predictions when controlling the ego-robot.

Part II

Learning for Anticipatory Navigation

Chapter 5

Stochastic Process Anticipatory Navigation

5.1 Introduction

Robots coexisting with humans often need to drive towards a goal while actively avoiding collision with pedestrians and obstacles present in the environment. To safely manoeuvre in crowded environments, humans have been found to anticipate the movement patterns in the environment and adjust accordingly (Higuchi, 2013). Likewise, we aim to endow robots with the ability to adopt a probabilistic view of their surroundings and consider likely future trajectories of pedestrians. Static obstacles in the environment are encoded in an occupancy map, and avoided during local navigation. We present the Stochastic Process Anticipatory Navigation (SPAN) framework, which generates local motion trajectories that takes into account predictions of the anticipated movements of dynamics objects.

SPAN leverages a probabilistic predictive model to predict the movements of pedestrians. Motion prediction for dynamic objects, in particular for human pedestrians, is an open problem and has typically been studied in isolation without direct consideration of its use in a broader planning framework. Many recent methods use learning-based approaches (Alahi et al., 2016; Zhi et al., 2019; Gupta et al., 2018) to generalise motion patterns from collected

This chapter has been published in ICRA as Zhi et al. (2021b).

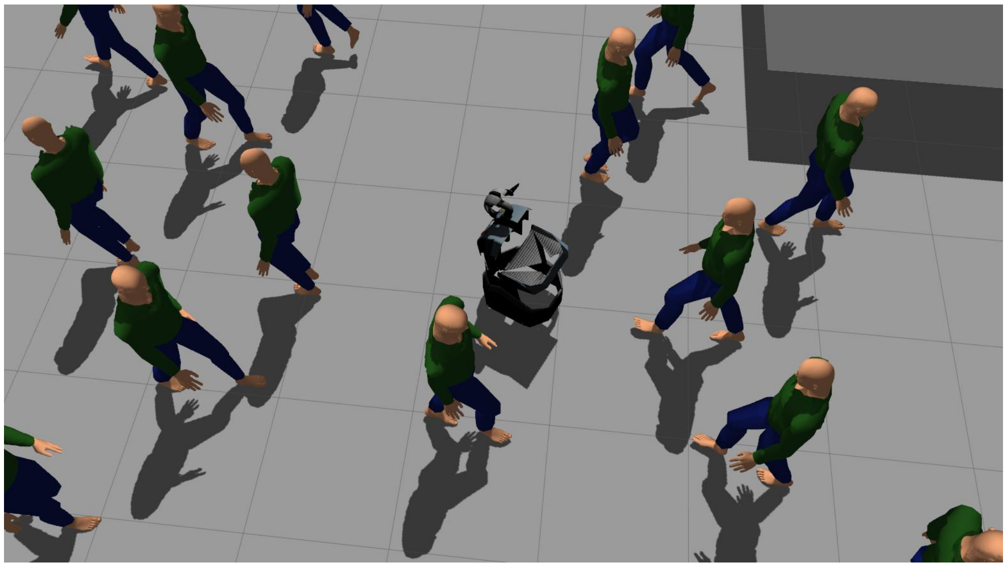


Figure 5.1: A non-holonomic robot drives towards a goal while interacting with pedestrians and static obstacles, and actively adapts to probabilistic predictions of surrounding movements.

data. To adequately capture the uncertainty of these predictions, representations of future motion patterns of pedestrians need to be probabilistic. Furthermore, collision-checking with future pedestrian positions needs to be done at a relatively high time resolution. To address these requirements, we represent uncertain future motions as continuous-time stochastic processes. Our representation captures prediction uncertainty and can be evaluated at an arbitrary time-resolution. We train a neural network to predict parameters of the stochastic process, conditional on the observed motion of pedestrians in the environment. Navigating through an environment requires online generation of collision-free and kinematically-feasible trajectories. We combine learned probabilistic anticipations of future motion and an occupancy map into a non-holonomic control problem formulation. To account for environment interactions a time-to-collision term (Hayward, 1972) is integrated into the control problem. The formulated problem is non-smooth, and a derivative-free constrained optimiser is used to efficiently solve the problem to obtain control actions. A receding horizon approach is taken and the problem is re-optimised at a fixed frequency to continuously adapt to pedestrian prediction updates.

SPAN is novel in jointly integrating data-driven probabilistic predictions of pedestrian motions and static obstacles in a control problem formulation. The technical contributions of this chapter include: (1) A continuous stochastic process representation of pedestrian futures, that is compatible with neural networks, and allows for fast chance-constrained collision-checking, at flexible resolutions; (2) formulation of a control problem that utilises predicted stochastic processes as anticipated future pedestrian positions, to navigate through crowds. We demonstrate that the optimisation problem can be solved efficiently to allow the robot to navigate through crowded environments. SPAN also opens opportunities for advances in neural network-based motion prediction methods to be utilised for anticipatory navigation. The stochastic process representation of pedestrian motion is compatible with many other neural network models as the output component of the network.

5.2 Related Work

SPAN actively predicts the motion patterns of pedestrians in the surrounding area and avoids collision while navigating towards a goal. Simple solutions to motion prediction include constant velocity or acceleration models (Rudenko et al., 2020b). Recent methods of motion prediction utilise machine learning to generalise motion patterns from observed data (Zhi et al., 2019b; Alahi et al., 2016; Gupta et al., 2018; Zhi et al., 2019). These methods have a particular focus on capturing the uncertainty in the predicted motion, as well as on conditioning on a variety of environmental factors, such as the position of neighbouring agents. Generally, past literature examines the motion prediction in isolation, without addressing issues arising from integrating the predictions into robot navigation, where the anticipation of pedestrian futures is valuable.

We seek to generate collision-free motion, accounting for predicted motion patterns. Planning collision-free paths is well-studied, with methods at differing levels of locality. At one end of the spectrum, probabilistically-complete planning methods (Lavalle, 1998) explore the entire search-space to find a globally optimal solution. At the other end, local methods (Khatib, 1985; Cheng et al., 2018; Ratliff et al., 2009) aim to perturb collision-prone motion away from obstacles, trading-off global optimality guarantees for faster run-time. Our work focuses on the local aspect, where we aim to quickly obtain a sequence of control commands to navigate through dynamic environments. Various non-linear controllers (Bruno Brito and Alonso-Mora, 2019; Neunert et al., 2016; Williams et al., 2016; Zhou et al., 2018) also generate control sequences to avoid collisions.

Local collision-avoidance methods for navigation for dynamic environments have been

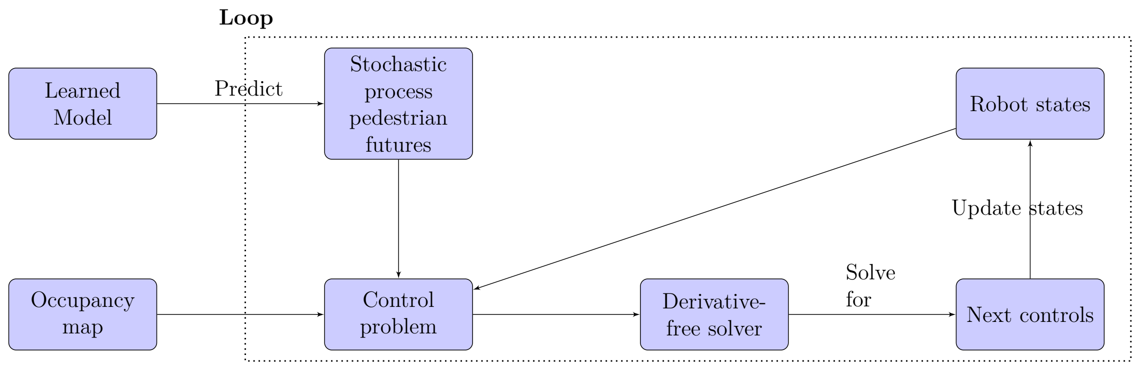


Figure 5.2: An overview of SPAN.

explored. Methods such as velocity obstacles (VO) (Fiorini and Shiller, 1998) and optimal reciprocal collision avoidance (ORCA) (Berg et al., 2009) find a set of feasible velocities for the robot, considering the current velocities of other agents. Partially observable Markov decision process (POMDP) solvers have been used in (Luo et al., 2018; Cai et al., 2020) to control linear acceleration for autonomous vehicles, with steering controls obtained from a global planner. Methods based on OCRA that indirectly solve for control sequences, such as (Bareiss and van den Berg, 2015), are known to be overly conservative. Less conservative methods have also been developed to incorporate human intent to identify unlikely human action (Bajcsy et al., 2021). An approach was proposed in Davis et al. (2020), where a subgradient-based method utilises a time-to-collision value (Hayward, 1972), under constant velocity assumptions. Similarly, SPAN also optimises against time-to-collision, and uses derivative-free constrained optimisation solvers. We extend the usage of time-to-collision to probabilistic predictions of pedestrian positions, provided by learning models, as well as integrating static obstacles captured by occupancy maps.

5.3 Problem Overview

A general overview of SPAN is provided in fig. 5.2. (1) First, we train a model offline to predict probabilistic future pedestrian positions, conditioning on previous positions; (2) The predictive model and occupancy map are incorporated in an anticipatory control problem; (3) The control problem is solved online, in a receding horizon manner, with derivative-free solvers to obtain controls for the next states. This process is looped, with the learned model is continuously queried, and the problem continuously formulated and solved to obtain the next states.

5.3.1 Problem Formulation

This chapter addresses the problem of controlling a robot, to navigate towards a goal $\boldsymbol{g}$, in an environment with moving pedestrians and static obstacles. The state of the robot at a given time t is given by $\mathbf{x}(t) = [x(t)\ y(t)\ \theta(t)]^{\top}$, denoting the robot's two-dimensional spatial position and an additional angular orientation. We focus our investigation on wheeled robots, with the robot following velocity-controlled non-holonomic unicycle dynamics, where the system dynamics are given by:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) = [v\cos(\theta), v\sin(\theta), \omega]^{\top}, \qquad \mathbf{u} = [v, \omega]^{\top}, \tag{5.1}$$

where $\dot{\mathbf{x}}$ denotes the state derivatives. The robot is controlled via its linear velocity v and angular velocity ω. We denote the controls as $\mathbf{u}$ and the non-linear dynamics as $f(\mathbf{x}, \mathbf{u})$.

Information about *static obstacles* in the environment is provided by an occupancy map, denoted as $f_m(\cdot)$, which represents the probability of being occupied $p(\text{Occupied}|\hat{\boldsymbol{x}}) \in [0, 1]$ at a coordinate $\hat{\boldsymbol{x}} \in \mathbb{R}^2$. The set of positions of N pedestrian obstacles, at a given time t from the current time, is denoted by the set $\mathcal{O} = \{\boldsymbol{o}^1(t), \ldots, \boldsymbol{o}^N(t)\}$. We are assumed to have movement data, allowing for the training of a probabilistic model to predict future positions of pedestrians.

5.4 Probabilistic and Continuous Prediction of Pedestrian Futures

In this section, we introduce a continuous *stochastic process* representation of future pedestrian motion, and outline how the representation integrates into a neural network learning model.

5.4.1 Pedestrian Futures as Stochastic Processes

Representations of future pedestrians positions need to capture uncertainty. Additionally, the time-resolution of the pedestrian position forecasts needs to synchronise with the frequency of collision checks. These factors motivate the use of a continuous-time stochastic process (SP) for future pedestrian positions. A SP can be thought of as a distribution over functions. Furthermore, the continuous nature allows querying of future pedestrian position at an arbitrary time, rather than at fixed resolution, without additional on-line interpolation.

We start by considering a deterministic pedestrian motion trajectory, before extending to a *distribution over motion trajectories.* A trajectory is modelled by a continuous-time function, given as the weighted sum of m basis functions. We write the n^{th} pedestrian $\boldsymbol{o}^n \in \mathcal{O}$ coordinate at t as:

$$\mathbf{o}^n(t) = \mathbf{W}^{n\top}\mathbf{\Phi}(t), \qquad \mathbf{\Phi}(t) = [\boldsymbol{\phi}(t, \mathbf{t}'), \boldsymbol{\phi}(t, \mathbf{t}')], \tag{5.2}$$

where $\mathbf{W}^{n\top} \in \mathbb{R}^{m\times 2}$ is a matrix of weights that defines the trajectory. $\mathbf{\Phi}(t)$ contains two vectors of basis function evaluations, $\boldsymbol{\phi}(t, \mathbf{t}')$, one for each coordinate axis. $\boldsymbol{\phi}(t, \mathbf{t}') = [\phi(t, t'_1), \ldots, \phi(t, t'_m)]^\top$ is a vector of basis function evaluations at m evenly distanced time points, $\mathbf{t}' \in \mathbb{R}^m$. We use the squared exponential basis function given by $\phi(t, t') = \exp(-\gamma\|t-t'\|^2)$, where γ is a length-scale hyper-parameter.

To extend our representation from a single trajectory to a distribution over future motion trajectories, we assume that the weight matrix is not deterministic, but random with a matrix normal (MN) distribution:

$$\mathbf{W}^n \sim MN(\mathbf{M}^n, \mathbf{U}^n, \mathbf{V}^n). \tag{5.3}$$

The matrix normal distribution is a generalisation of the normal distribution to matrix-valued random variables (Dutilleul, 1999), parameters mean matrix $\mathbf{M}^n \in \mathbb{R}^{m\times 2}$ and scale matrices $\mathbf{U}^n \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{2\times 2}$. $\mathbf{M}^n$ is analogous to the mean of a normal distribution, and $\mathbf{U}^n, \mathbf{V}^n$ capture the covariance. A log-likelihood of the matrix normal distribution is given later in eq. (5.6). Predicting a SP requires predicting $\mathbf{M}^n, \mathbf{U}^n, \mathbf{V}^n$.

5.4.2 Learning Pedestrian Motion Stochastic Processes

This subsection outlines how to train a model to predict SP representations of pedestrian motion. In the following subsection, we denote the weight matrix as $\mathbf{W}$ and parameters of the matrix normal distribution as $\mathbf{M}, \mathbf{U}, \mathbf{V}$, without referring to an obstacle index of a specific pedestrian. We aim to condition on a short sequence of recently observed pedestrian positions, up to the present $\{\widehat{\mathbf{o}}_1, \ldots, \widehat{\mathbf{o}}_p\}$, and predict $\mathbf{M}, \mathbf{U}, \mathbf{V}$, which define a SP of the motion

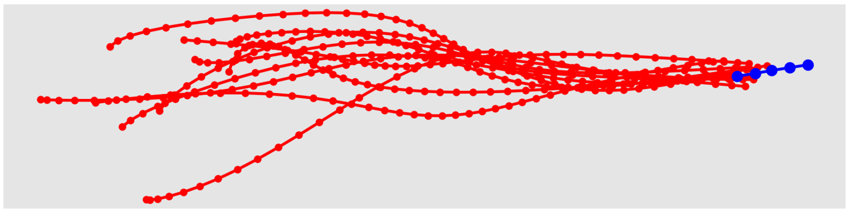


Figure 5.3: We can visualise the predicted stochastic process by sampling trajectories (red) from a predicted distribution, conditional on previously observed pedestrian coordinates (blue).

thereafter. A predictive model is trained offline with collected data, and then queried online.

From timestamped data to continuous functions

Pedestrian motion data is typically in the form of a discrete sequence of n timestamped coordinates $\{t_i, \widehat{\mathbf{o}}_i\}_{i=1}^{n}$, where t_i gives time and $\widehat{\mathbf{o}}_i \in \mathbb{R}^2$ gives coordinates. Timestamped coordinates can be converted into a continuous function by finding a deterministic $\mathbf{W}$. For a pre-defined basis function $\mathbf{\Phi}(\cdot)$, we solve the ridge regression problem:

$$\min_{\mathbf{W}} \left\{ \sum_{i=1}^{n} (\widehat{\mathbf{o}}_i - \underbrace{\mathbf{W}^{\top}\mathbf{\Phi}(t_i)}_{\mathbf{o}(t_i)})^2 + \lambda \|\mathrm{vec}(\mathbf{W})\|^2 \right\} \tag{5.4}$$

where λ is a regularisation hyper-parameter, and vec$(\cdot)$ is the vectorise operator. By converting timestamped data to continuous functions, we can obtain a dataset, $\mathcal{D}$, with N_D pairs, $\mathcal{D} = \{(\{\widehat{\mathbf{o}}_1, \dots, \widehat{\mathbf{o}}_p\}_d, \mathbf{W}_d)\}_{d=1}^{N_D}$, where $\{\widehat{\mathbf{o}}_1, \dots, \widehat{\mathbf{o}}_p\}_d$ are previously observed coordinates, and $\mathbf{W}_d$ defines the continuous trajectory from $\widehat{\mathbf{o}}_p$ onwards. $\mathbf{W}_d$ is obtained from timestamped coordinate data via eq. (5.4). This dataset is used to train the predictive model.

Training a Neural Network

We wish to learn a mapping, F, from a sequence of observed pedestrian coordinates, $\{\widehat{\mathbf{o}}_1, \dots, \widehat{\mathbf{o}}_p\}$, to the parameters over distribution $\mathbf{W}$, that defines future motion beyond the sequence. We can use a neural network as,

$$\overbrace{(\mathbf{M}, \mathbf{U}, \mathbf{V})}^{\text{defines SP over future motion}} = \underbrace{F}_{\text{neural network function approximator}}(\overbrace{\{\widehat{\mathbf{o}}_1, \ldots, \widehat{\mathbf{o}}_p\}}^{\text{past coordinates}}) \tag{5.5}$$

and train via the negative log-likelihood (Dutilleul, 1999) of the matrix normal distribution, over N_D training samples of dataset $\mathcal{D}$. The loss function of the network is given as:

$$\mathcal{L} = -\sum_{d=1}^{N_D} \log \frac{\exp\{-\frac{1}{2}\mathrm{tr}[\mathbf{V}^{-1}(\mathbf{W}_d - \mathbf{M})^\top \mathbf{U}(\mathbf{W}_d - \mathbf{M})]\}}{(2\pi)^m |\mathbf{V}||\mathbf{U}|^{\frac{m}{2}}}, \tag{5.6}$$

where $\mathrm{tr}(\cdot)$ is the trace operator. In our experiments, a simple 3-layer fully-connected neural network with width 100 units, and an output layer giving vectorised $\mathbf{M}, \mathbf{U}, \mathbf{V}$ is sufficiently flexible to learn F. In this work, we focus on outlining a representation of motion able to be learned and used for collision avoidance. More complex network architectures can be used to learn F, including recurrent network-based models (Alahi et al., 2016).

Querying the Predictive Model

After training, the neural network outputs $\mathbf{M}, \mathbf{U}, \mathbf{V}$, conditional on observed sequences. $\mathbf{M}, \mathbf{U}, \mathbf{V}$ define random weight matrix $\mathbf{W}$, and the SP model via eq. (5.2). Individual trajectories can be realised from the SP by sampling $\mathbf{W}$. Figure 5.3 shows sampled trajectories from a predicted SP.

5.5 Anticipatory Navigation with Probabilistic Predictions

5.5.1 Time-to-Collision Cost for Control

This subsection introduces a control formulation which makes use of probabilistic anticipations of future pedestrian positions. Human interactions have been found to be governed by an inverse power-law relation with the time-to-collision (Karamouzas et al., 2014), and can be integrated in robot control (Davis et al., 2020). To this end, we formulate an optimal control problem, with the inverse time-to-collision in the control cost:

$$\min_{\mathbf{u}} \quad \left\{ \|\hat{\boldsymbol{x}}(T) - \boldsymbol{g}\|_2 + \frac{\kappa}{\tau(\mathbf{x}_0, \mathbf{u}, \boldsymbol{o}, f_m)} \right\} \tag{5.7a}$$

$$\text{s.t.} \quad \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}), \quad t \in [0, T], \tag{5.7b}$$

$$\mathbf{x}(0) = \mathbf{x}_0, \tag{5.7c}$$

$$\mathbf{u}_{lower} \leq \mathbf{u} \leq \mathbf{u}_{upper}, \tag{5.7d}$$

where T is the time horizon; $\hat{\boldsymbol{x}}$, $\boldsymbol{g}$, $\mathbf{x}$, $\mathbf{u}$ are the robot coordinates, goal coordinates, states, and target controls; τ is the time-to-collision value; κ is a weight hyper-parameter controlling the collision-averseness; $\boldsymbol{o}$ denotes the position of pedestrians, f_m denotes an occupancy map; $\mathbf{u}_{upper}, \mathbf{u}_{lower}$ denote the control limits, and f refers to the system dynamics given in eq. (5.1). Like (Davis et al., 2020), we define time-to-collision as the first time a collision occurs under constant controls over the time horizon, enabling efficient optimisation.

5.5.2 Chance-constrained Collision with Probabilistic Pedestrian Movements

This subsection outlines a method to check for collisions between the robot and SP representations of pedestrian movement, to evaluate a time-to-collision value. Collision occurs when the distance between the robot and a pedestrian is below their collision radii. The set of coordinates where collision occurs, at time t, with the n^{th} pedestrian is given by:

$$\mathcal{C}_t^n = \left\{ \boldsymbol{x} \in \mathbb{R}^2 \;\middle|\; \|\boldsymbol{x} - \boldsymbol{o}^n(t)\|_2 < r_{\hat{\boldsymbol{x}}} + r_{o_n} \right\}, \tag{5.8}$$

where $r_{\hat{\boldsymbol{x}}}$, $r_{\boldsymbol{o}_n}$ denote the radii of the robot and the pedestrian, and $\boldsymbol{o}^n(t)$ the pedestrian position. The position of pedestrians at a given time is uncertain, and given by the stochastic process defined in eq. (5.2). Therefore, we specify collision at t in a chance constraint manner, where a collision between the robot and a pedestrian is assumed to occur if the probability of the robot position, $\hat{\boldsymbol{x}}(t)$, being in set $\mathcal{C}_t^n$ is above threshold ϵ, i.e. $p(\hat{\boldsymbol{x}}(t) \in \mathcal{C}_t^n) > \epsilon$. Time-to-collision is taken as the first instance that a collision occurs, i.e., when the probability of collision with pedestrians exceeds ϵ. Thus, for the chance-constraint formulation, the set of robot motions colliding at t is given by,

$$\mathcal{C}_t = \{\hat{\boldsymbol{x}}(t) | \max_n \{p(\hat{\boldsymbol{x}}(t) \in \mathcal{C}_t^n)\} > \epsilon\}. \tag{5.9}$$

As the weight parameters, $\mathbf{W}$, of the stochastic process is a matrix normal distribution, the position distribution at a specified time is a multivariate Gaussian. By considering the random weight matrix distribution of the n^{th} pedestrian, $\mathbf{W}^n$ and eq. (5.2), we have:

$$\boldsymbol{o}^n(t) \sim \mathcal{N}(\mathbf{M}^{n\top}\boldsymbol{\Phi}(t), \mathbf{V}^n\boldsymbol{\Phi}^\top(t)\mathbf{U}^n\boldsymbol{\Phi}(t)), \tag{5.10}$$

where $\mathbf{M}^n, \mathbf{U}^n, \mathbf{V}^n$ are the parameters of the distribution of weight matrix $\mathbf{W}^n$, and $\boldsymbol{\Phi}(t)$ contains basis function evaluations. The position of the robot $\hat{\boldsymbol{x}}(t)$ is deterministic, thus $\hat{\boldsymbol{x}}(t) - \boldsymbol{o}^n(t)$ is also multi-variate Gaussian, with a shifted mean. The probability of collision with the n^{th} pedestrian is the integral of a Gaussian over a circle of radius $r_{\hat{\boldsymbol{x}}} + r_{\boldsymbol{o}_n}$. Let $\boldsymbol{d}^n = \hat{\boldsymbol{x}}(t) - \boldsymbol{o}^n(t)$, we have the expressions:

$$p(\hat{\boldsymbol{x}}(t) \in \mathcal{C}_t^n) = \int_{\|\boldsymbol{d}^n\|_2 < r_{\hat{\boldsymbol{x}}} + r_{\boldsymbol{o}_n}} p(\boldsymbol{d}^n) \mathrm{d}\boldsymbol{d}^n, \tag{5.11}$$

$$p(\boldsymbol{d}^n) = \mathcal{N}(\hat{\boldsymbol{x}}(t) - \mathbf{M}^{n\top}\boldsymbol{\Phi}(t), \mathbf{V}^n\boldsymbol{\Phi}^\top(t)\mathbf{U}^n\boldsymbol{\Phi}(t)). \tag{5.12}$$

This expression is intractable to evaluate analytically, and an upper bound to integrals of this kind is given in (Zhu and Alonso-Mora, 2019) by:

$$p(\hat{\boldsymbol{x}}(t) \in \mathcal{C}_t^n) \leq \frac{1}{2}\left\{1 + erf\left(\frac{r_{\hat{\boldsymbol{x}}} + r_{\boldsymbol{o}_n} - \boldsymbol{a}^\top\boldsymbol{d}^n(t)}{\sqrt{2\boldsymbol{a}^\top\boldsymbol{\Sigma}^n(t)\boldsymbol{a}}}\right)\right\}, \tag{5.13}$$

$$\boldsymbol{\Sigma}^n(t) = \mathbf{V}^n\boldsymbol{\Phi}^\top(t)\mathbf{U}^n\boldsymbol{\Phi}(t), \qquad \boldsymbol{a} = \frac{\boldsymbol{d}^n(t)}{\|\boldsymbol{d}^n(t)\|_2}, \tag{5.14}$$

where $\boldsymbol{\Sigma}^n(t) \in \mathbb{R}^{2\times 2}$ is the covariance of the Gaussian $\boldsymbol{o}^n(t)$, and $erf(\cdot)$ is the error function. Using the defined eq. (5.13), we can efficiently evaluate the collision probability upper-bound between the robot and each pedestrian, and take the first time probability upper-bound exceeds ϵ as the time-to-collision with pedestrians. By eqs. (5.9) and (5.13) a collision occurs at t if:

$$\max_{n=1,\ldots,N}\left\{\frac{1}{2}\left[1 + erf\left(\frac{r_{\hat{\boldsymbol{x}}} + r_{\boldsymbol{o}_n} - \boldsymbol{a}^\top\boldsymbol{d}^n(t)}{\sqrt{2\boldsymbol{a}^\top\boldsymbol{\Sigma}^n(t)\boldsymbol{a}}}\right)\right]\right\} > \epsilon. \tag{5.15}$$

That is, collisions arise if the probability of collision with any pedestrian is above threshold ϵ. The robot-pedestrian time-to-collision, $\tau_{\boldsymbol{o}}$, is the earliest time collision with a pedestrian occurs, namely $\tau_{\boldsymbol{o}} = \min_{t\in[0,T]}(t)$, subject to eq. (5.15).

5.5.3 Chance-constrained Collision with Occupancy Maps

Static obstacles in the environment are often captured via occupancy maps. Time-to-collision values accounting for occupancy maps can be integrated into SPAN. An occupancy map can be thought of as a function, $f_m(\hat{\boldsymbol{x}})$, mapping from position, $\hat{\boldsymbol{x}} \in \mathbb{R}^2$, to the probability the position is occupied, $p(\text{Occupied}|\hat{\boldsymbol{x}})$. We ensure the probability of being occupied at any coordinate on the circle of radius, $r_{\hat{\boldsymbol{x}}}$, around our robot position is below the chance constraint threshold, ϵ. By considering the occupancy map as an implicit surface, a collision occurs at time t if,

$$\max_{\phi\in[-\pi,\pi)} \left\{ f_m\left(\hat{\boldsymbol{x}}(t) + r_{\hat{\boldsymbol{x}}} \begin{bmatrix} \sin(\phi) \\ \cos(\phi) \end{bmatrix} \right) \right\} > \epsilon. \tag{5.16}$$

Maximising the independent variable, ϕ, sweeps a circle around the robot to check for collisions. As the domain of ϕ is relatively limited and querying from occupancy maps is highly efficient, taking under 10^{-6}s each query, a brute force optimisation can be done. The time-to-collision due to static obstacles is then the first time-step where a collision occurs, namely $\tau_m = \min_{t\in[0,T]}(t)$, subject to eq. (5.16). We take the overall time-to-collision value for the optimisation in eq. (5.7a) as the minimum of the time-to-collision for both static obstacles and pedestrians, specifically $\tau = \text{minimum}(\tau_{\boldsymbol{o}}, \tau_m)$.

5.5.4 Solving the Control Problem

We take a receding horizon control approach, continuously solving the control problem defined in eq. (5.7a) - eq. (5.7d), and updating the controls for a single time-step. Algorithm 3 outlines our framework, bringing together the predictive model, the formulation and solving of control problem. A predictive model of obstacle movements is trained offline with motion data. The predictive model and a static map of the environment are provided to evaluate the time-to-collision, τ, in the control problem. We take a "single-shooting" approach and evaluate the integral in the control cost via an Euler scheme with step-size $\Delta t = 0.1s$. At each time step we check for collision by eqs. (5.15) and (5.16), and τ is the first time-step

Algorithm 3: Stochastic Process Anticipatory Navigation

```
Input: $\mathbf{x}_0, \dot{\mathbf{x}}_0, \mathbf{g}, \mathcal{D}, f_m, \Delta t$
Offline: Train neural network, $F$, to predict with dataset $\mathcal{D}$ (section 5.4.2)
Initialise: $t \leftarrow 0$; $\mathbf{x}(0) \leftarrow \mathbf{x}_0$; $\dot{\mathbf{x}}(0) \leftarrow \dot{\mathbf{x}}_0$
  while goal not reach do
      Observe $N$ pedestrians' coordinates, within a past time window,
       $\{\{\widehat{\mathbf{o}}_1, \dots, \widehat{\mathbf{o}}_p\}\}_{i=0}^{N}$;
      $\mathcal{O} \leftarrow \emptyset$ ;                                   // collects predictions
      Predict $\{\mathbf{M}^n, \mathbf{U}^n, \mathbf{V}^n\}_{n=0}^{N} = F(\{\widehat{\mathbf{o}}_1, \dots, \widehat{\mathbf{o}}_p\}_i)$;
      for ( $n = 1$ to $N$// For all peds.
       ) {
          // Construct pedestrian futures as SP
          $\boldsymbol{o}^n(t) = \mathbf{W}^{n\top}\mathbf{\Phi}(t)$, $\mathbf{W}^n \sim MN(\mathbf{M}^n, \mathbf{U}^n, \mathbf{V}^n)$;
          $\mathcal{O} \leftarrow \mathcal{O} \cup \{\boldsymbol{o}^n(t)\}$;
      }
      // optimise for control with pred.
      $\mathbf{u} \leftarrow$ Solve eqs. (5.7a) to (5.7d), with $\mathcal{O}, f_m, \mathbf{x}(t)$ to get controls;
      $\mathbf{x}(t+\Delta t) \leftarrow \text{NextState}(\mathbf{x}(\text{t}), \mathbf{u})$ // Execute controls
      $t \leftarrow t + \Delta t$;
      if $\|\hat{\boldsymbol{x}}(t) - \mathbf{g}\|_2 < \epsilon_{goal}$ then
          Terminates;                                              // goal reached
      end
  end
```

a collision occurs. The control problem is non-smooth, so we use the derivative-free solver COBYLA (Powell, 1994) to solve the optimisation problem, and obtain controls **u**. As the control-space is relatively small, the optimisation can be computed quickly. We can re-solve with several different initial solutions in the same iteration to refine and evade local minima. Controls are updated at 10 Hz.

5.6 Experimental Evaluation

We empirically evaluate the ability of SPAN to navigate in environments with pedestrians and static obstacles. Results and desirable emergent behaviour are discussed below.

5.6.1 Experimental Setup

Both simulated pedestrian behaviour and real-world pedestrian data are considered in evaluations. We use a power-law crowd simulator (Karamouzas et al., 2014, 2017) with pedestrians' behaviour set as aggressive, and real-world indoor pedestrian data from (Rudenko et al., 2020a). We construct two simulated environments, *Sim 1* and *Sim 2*, both with 24 pedestrians. *Sim 1* models a crowded environment with noticeable congestion; *Sim 2* models a relatively open environment with less frequent pedestrian interaction. The simulated pedestrians have a maximum speed of $1m/s$. Pedestrian trajectories collected at different times in dataset are overlaid, increasing the crowd density. The maps used in our experiments are shown in fig. 5.4. We navigate the robot from starting point to a goal. Gazebo simulator (Koenig and Howard, 2004) was used for visualisation. The following metrics are used:

1. Time-to-goal (TTG): The time, in seconds, taken for the robot to reach the goal;
2. Duration of collision (DOC): The total time, in seconds, the robot is in collision.

We evaluate SPAN against the following methods:

- Closed-loop Rapidly-exploring Random Trees* (CL-RRT*). An asymptotic optimal version of (Kuwata et al., 2008) with a closed-loop prediction and a nonlinear pure-pursuit controller (Amidi and Thorpe, 1991). (Kuwata et al., 2008) demonstrates an ability to handle dynamic traffic, and is a DARPA challenge entry. At each iteration, a time budget of $0.1s$ is given to the planner to find a free trajectory towards the goal, with pedestrians represented as obstacles. If a feasible solution is not found, the robot will remain at its current pose.
- A reactive controller. The reactive controller continuously solves to find optimal collision-free controls for the next time-step, without anticipating future interaction.

In our experiments, we set the time-to-collision weight, κ=100; the trajectory length-scale hyper-parameter, $\gamma = 0.01$; the control limits on linear/ angular velocity as $-1 \leq \boldsymbol{v} \leq 1$ and $-1 \leq \omega \leq 1$. Each iteration we solve for a "look-ahead" time horizon of $T = 4.0s$. The collision threshold is set at $\epsilon = 0.25$, and the radii for the robot and the pedestrians are given as $r_{\hat{\boldsymbol{x}}}, r_{\boldsymbol{o}} = 0.4m$. In the simulated setups, the predictive model is trained with additionally generated data from the crowd simulator; in the dataset setup, 80% of the data

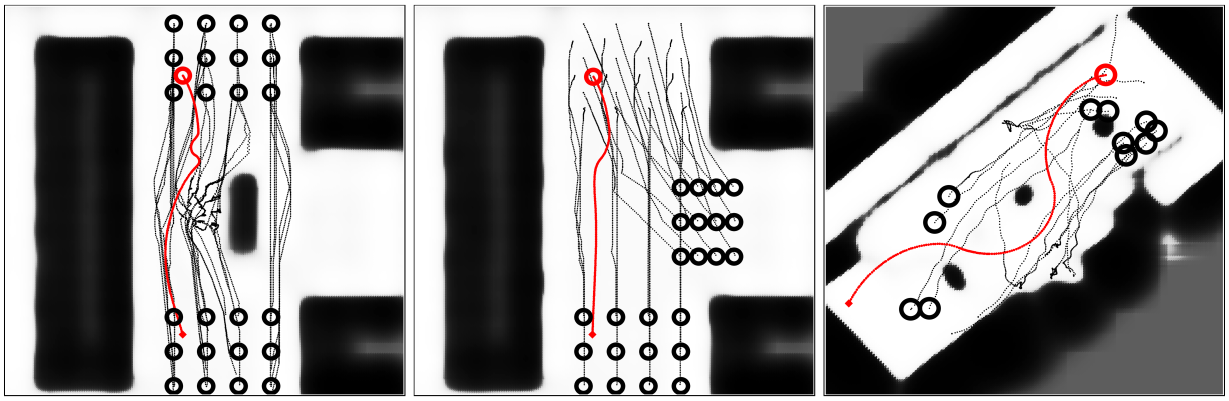


Figure 5.4: Motion of pedestrians (black) and robot (red) overlaid on an occupancy map of the environments for Sim1 (left), Sim2 (center), and Dataset (right). Note that crossing trajectories does not necessarily indicate a collision, as moving agents could be at the same position at different times. The starting position of the pedestrians and robot are indicated by circles. The robot navigates smoothly and sensibly, remaining collision-free.

was used to train the predictive model, with the remainder for evaluation. We condition on observed pedestrian movements in the last 0.5s, to predict the next 4.0s. Occupancy maps provided are implemented as continuous occupancy maps (Zhi et al., 2019a; Ramos and Ott, 2016), with the method in (Ramos and Ott, 2016). Controls were solved with different initialisations 40 times per iteration. The predictive model is implemented in Tensorflow (Abadi et al., 2016), control formulation and COBYLA (Powell, 1994) are in FORTRAN, interfacing with Python2. Videos of a controlled non-holonomic robot navigating through simulated aggressive crowds and real pedestrian data can be found in `https://youtu.be/bLDCnE5Lo6Q` and `https://youtu.be/ZJ5Uh0JgMdA`.

5.6.2 Experimental Results

The resulting motion trajectories by the controlled robot in the experiments are illustrated in fig. 5.4, where the motion trajectories of the robot is marked in red, while those of pedestrians are marked in black. At a glance, we see that the controlled robot navigates smoothly and sensibly. The evaluated metrics are tabulated in table 5.1. The average computational times of an entire iteration of SPAN, including prediction and obtaining controls, are 27ms, 31ms, 33ms in *Dataset*, *Sim 1*, *Sim 2* setups respectively, with no iteration taking more than 50ms in any setup.

SPAN is able to control the robot to reach the goal, without collisions, in all of the exper-

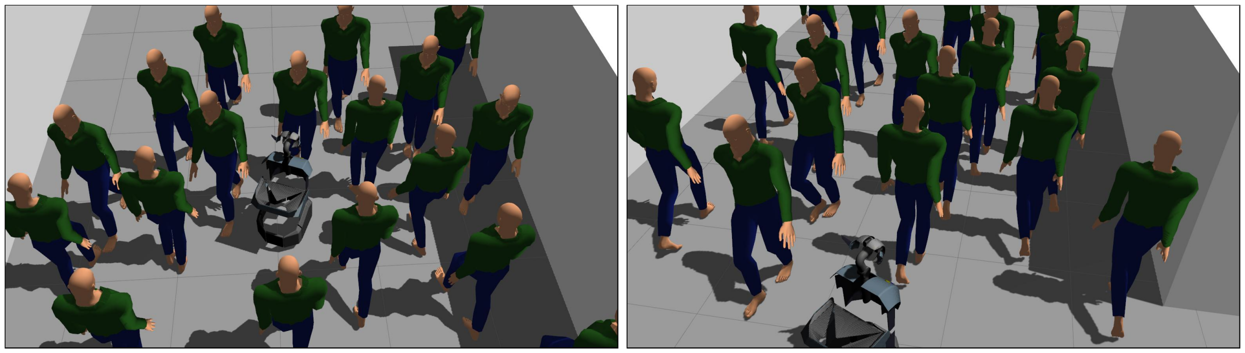


Figure 5.5: *Sim 1* setup. (Left): The reactive controller often drives head-on into the crowd, leading the robot to become stuck and resulting in minor collisions. A considerable amount of time is needed to resolve the congestion. (Right): The robot controlled by SPAN evades congestion when possible, and follows behind pedestrians that are moving in the same direction.

iment environments. In particular, relative to the compared methods, SPAN demonstrates markedly safer navigation through the *Sim 1* and *Dataset* environment setups. Both of these environments are relatively cluttered, with regions of congestion arising. We observe that both CL-RRT* and the reactive controller has a greater tendency to freeze amongst large crowds, while SPAN gives more fluid navigation amidst crowds. Figure 5.5 shows how the reactive controller (left figure) and SPAN (right figure) navigate the robot through crowds in the dense *Sim 1* environment. The reactive controller often controls the robot to drive towards incoming pedestrians head-on. This behaviour often results in driving into bottleneck positions and congestion arising, with the robot then unable to escape from a trapped position. On the other hand, the robot controlled by SPAN avoids getting stuck in crowds, as the future position of pedestrians are anticipated. Incoming pedestrians are actively evaded, while the robot follows other pedestrians through the crowd.

In the *Dataset* setup, both the compared CL-RRT* and the reactive controller produce very collision-prone navigation trajectories. This is due to the completely dominate pedestrian behaviour, which follows collected trajectory data, whereas the simulated pedestrian crowds also attempts to avoid collision with the robot. Both CL-RRT* and the reactive controller give a relatively straight trajectory towards the goal and are in collision for more than $3s$, colliding with three different pedestrians. Meanwhile SPAN produces a safe and meandering trajectory. The trajectories of the controlled robot by CL-RRT* and SPAN is illustrated in fig. 5.7, along with plots of how the distance to goal changes in time. We argue,

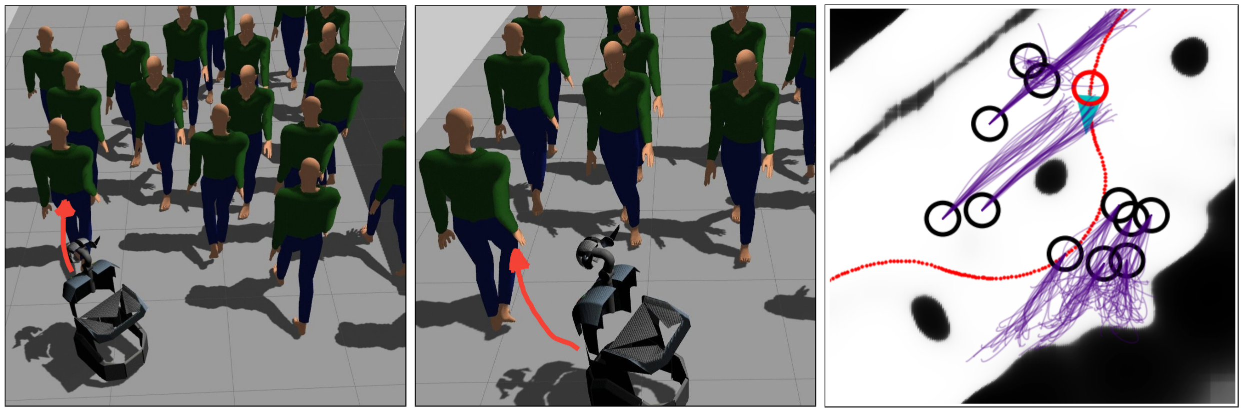


Figure 5.6: Anticipatory behaviour of our controlled robot can be observed. (Left, center) The robot follows (red arrow) pedestrians anticipated to move ahead, while preemptively avoiding incoming pedestrians. (Right) Robot as red circle and pedestrians as black. Sampled trajectories (in purple, next 4.0s) visualise predicted SP, and the robot trajectory is in red. The robot evades an incoming group of pedestrians preemptively, and moves toward pedestrians predicted to move away.

in the *Dataset* setup, although the time-to-goal of SPAN is marginally longer, the navigated path is much safer with no collisions, while the compared methods have more than 3s of collision. Additionally, all three of the evaluated methods perform similarly in the comparatively open *Sim 2* environment, producing collision-free and efficient trajectories. This is due to the ample space in the environment, with relative little congestion occurring.

Method	Metrics	Dataset	Sim 1 (crowded)	Sim 2 (open)
SPAN (ours)	TTG	21.7	25.9	17.4
	DOC	0.0	0.0	0.0
CL-RRT*	TTG	19.0	36.8	17.8
	DOC	3.7	1.1	0.0
Reactive	TTG	19.1	43.5	17.1
Controller	DOC	3.4	0.4	0.0

Table 5.1: The evaluations of SPAN against compared methods. TTG gives the time (in seconds) taken by the robot to reach the goal, and DOC gives the total duration (in seconds) the robot is in collision. SPAN finds a collision-free solution in all three evaluated set-ups.

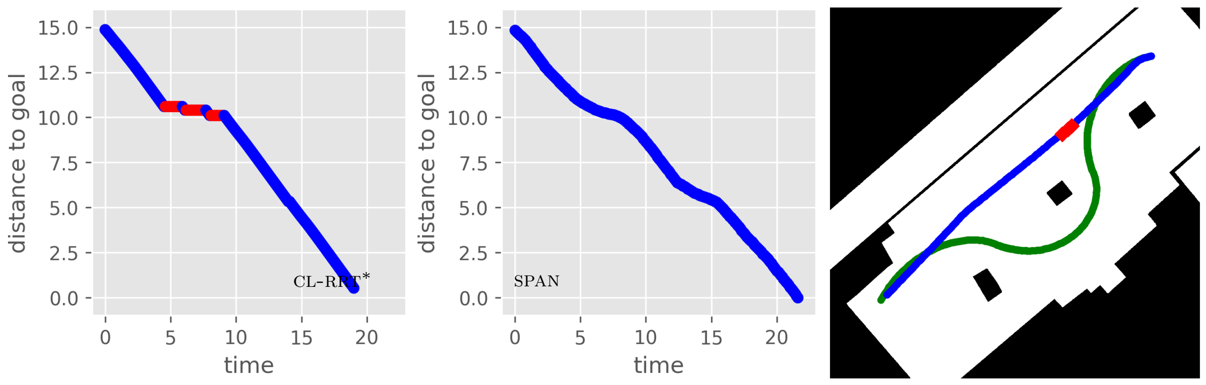


Figure 5.7: (Left, Center) shows time against the distance between the robot and goal, for the CL-RRT* and SPAN respectively. The times that are in collision are given in red, while the collision-free times are in blue. (Right) shows the motion of the robot as controlled by CL-RRT* in blue, and that as controlled by ours in green. Time-steps where the robot is in collision are in red. We see that SPAN is able to provide a safe path for the controlled robot, while CL-RRT* provides an aggressive path, resulting in over $3s$ of collision.

5.6.3 Emergent Behaviour

The formulated control problem in eqs. (5.7a) to (5.7d) does not explicitly optimise for the following of other agents. However, we observe following behaviour, behind other pedestrians, also moving in the goal direction, through crowds. Figure 5.6 (left, center) show the controlled robot following moving pedestrians through crowds, while avoiding those moving towards the robot. This is also detailed in fig. 5.6 (right), where predicted future positions are visualised by sampling and overlaying trajectories (in purple) from the SP model. The entire navigated motion trajectory taken by the robot is outlined in red. We observe the robot taking a swerve away from the group of incoming pedestrians preemptively, and turning to follow the other group predicted to move away. At that snapshot, the pedestrians in the lower half of the map blocks the path through to the goal, but are all predicted to move away, giving sufficient space for our robot to follow and pass through. The emergence of the crowd-following behaviour facilitates smooth robot navigation through crowds.

5.7 Summary

We introduce Stochastic Process Anticipatory Navigation (SPAN), a framework for anticipatory navigation of non-holonomic robots through environments containing crowds and static obstacles. We learn to anticipate the positions of pedestrians, modelling future positions as continuous stochastic processes. Then predictions are used to formulate a time-to-collision control problem. We evaluate SPAN with simulated crowds and real-world pedestrian data. We observe smooth collision-free navigation through challenging environments. A desired emergent behaviour of following behind pedestrians, which move in the same direction, arises. The stochastic process representation of pedestrian motion is compatible as an output for many neural network models, bringing together advances in motion prediction literature with robot navigation. Particularly, conditioning on more environmental factors to produce higher quality predictions within SPAN, is a clear direction for future work.

In the upcoming chapter 6, we shall develop a framework to predict how dynamic agents in an environment should move based on the environment structure. Humans have good intuition of what movement patterns in an environment when provided a floor-plan of it. We seek to endow robots with the same capability, by learning from experience.

Chapter 6

Trajectory Generation in New Environments from Past Experiences

Understanding movement trends in dynamic environments is critical for autonomous agents, such as service robots and delivery vehicles, to achieve long-term autonomy. This need is highlighted by the increasing interest in developing mobile robots capable of coexisting and interacting safely and helpfully with humans. Anticipating likely motion trajectories allows autonomous agents to anticipate future movements of other agents and thus navigate safely as well as plan socially compliant motions by imitating observed trajectories.

Many methods predicting motion extrapolate partially observed trajectories, without incorporating knowledge of the environment. Example of such methods include constant acceleration(Rong Li and Jilkov, 2003), filtering methods (Zhang et al., 2008), and auto-regressive models (Agarwal and Triggs, 2004). Advances in machine learning have lead to the development of methods which learn the motion behaviours from a dataset of trajectories. By training on motion data collected in the same environment, some learning-based methods can learn the general flow of movement (Zhi et al., 2019b; Kucner et al., 2017; Senanayake and Ramos, 2018), which implicitly accounts for environmental geometry. However, such methods are typically environment-specific, requiring predictions to be made in the same environment as that the training data was collected. Further methods, such as (Sadeghian et al., 2019) attempt to encode local parts of the environment along with trajectory extrap-

This chapter has been published in IROS as Zhi et al. (2021a).

olation.

Through experience, humans have developed the ability to anticipate movement patterns based on the layout of the environment. We hypothesise that the structure of environments contains information about how objects move within the environment. By considering 2D environment floor plans, given as occupancy grid maps, it is possible to transfer motion patterns from training environments to new environments where no motion observations have been made. We propose a probabilistic generative model, Occupancy-Conditional Trajectory Network (OTNet), capable of generating motion trajectories for new unseen environments by generalising motion trajectories from previously observed environments, as shown in Figure 6.1. We empirically demonstrate that our model is capable of generating motion trajectories in simulated and real-world environments by motion behaviour observed in other environments. We can additionally refine our predicted trajectory patterns, such as fixing its start point. We envision OTnet to provide priors about long-term trajectory patterns which can be integrated downstream in trajectory prediction models which reason about the immediate movement of surrounding agents, such as the predictive model in SPAN, outlined in chapter 5.

6.1 Related Work

6.1.1 Motion Trajectory Prediction

Estimating likely motion trajectories has been studied for a long time. Early simple methods to predict motion are often dynamics-based methods which extrapolate based on the laws of physics, such as constant velocity and constant acceleration models (Rong Li and Jilkov, 2003), with more complex Dynamics-based methods are utilised in (Zernetsch et al., 2016; Kooij et al., 2019). These models typically requires making a priori assumptions about agent motions, leaving little room to learn from observations. Other attempts at modelling motion trajectories include building dynamic occupancy grid maps based on occupancy data over time (Arbuckle et al., 2002; C. Mitsou and Tzafestas, 2007; Tanzmeister et al., 2014), inverse motion planning methods (Ziebart et al., 2009), and hidden Markov methods (Vasquez et al., 2009). Recent developments in machine learning have led to an interest in data-driven models (Gupta et al., 2018), which make fewer assumptions but rely heavily on observed data. A class of learning-based methods focus on learning a map of motion in a specific environment

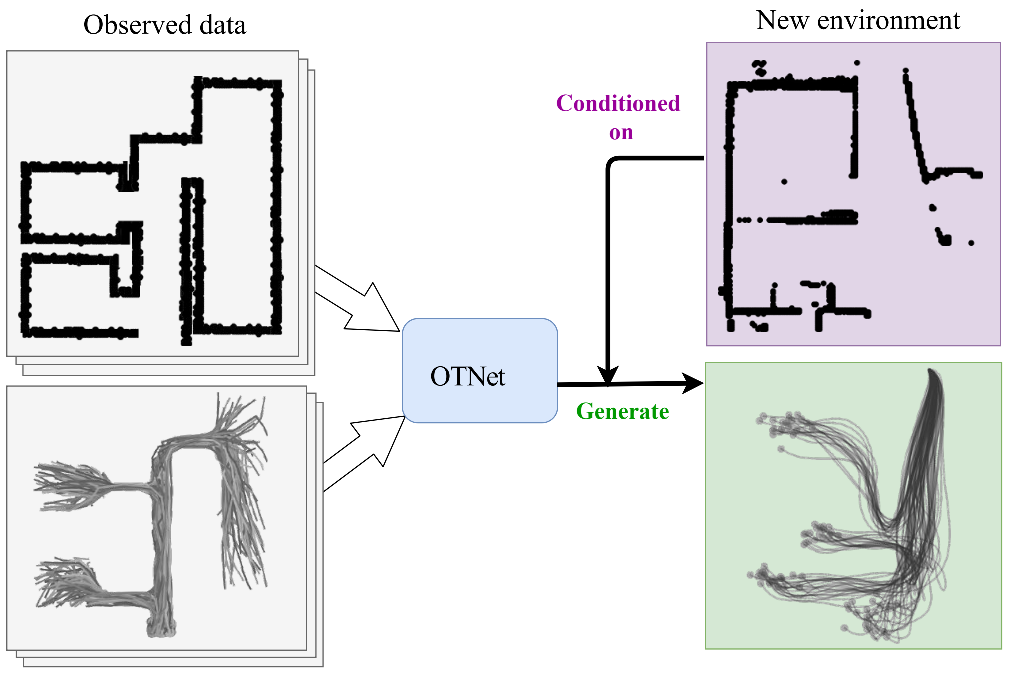


Figure 6.1: We use OTNet to transfer motion patterns from maps where motion has been observed to new environments where motion has not been observed, by conditioning on the occupancy map of the new environment.

to implicitly capture map-aware motion patterns in the environment (Zhi et al., 2019b; Senanayake and Ramos, 2018; Kucner et al., 2017; Mellado et al., 2018).

6.1.2 Trajectory Generation

Neural network based generative models have been used in motion prediction, where they typically generate complete trajectories conditioned on an incomplete one. Generative adversarial networks (GANs) (Goodfellow et al., 2014) are a popular class of generative models that for trajectory generation (Li et al., 2019; Gupta et al., 2018). Conditional Variational Autoencoders (CVAE) (Sohn et al., 2015) are another class of generative models (Ivanovic et al., 2018; Ivanovic and Pavone, 2019) that have been utilised in the same problems. Similar to this work, ideas for transferring motion patterns in specific transport scenarios are considered in (Jaipuria et al., 2018). Methods using neural networks to generate trajectories based on observed trajectories have also gained attention in the motion planning community. A recent work, Motion Planning Network (MPNet) (Qureshi et al., 2019), aims to generate trajectories by learning from a dataset of optimal trajectories in various simulated environments.

6.2 Methodology

6.2.1 Problem Formulation

This paper addresses the problem of generating motion trajectories in new environments, by transferring trajectories data observed in other training environments. Figure 6.1 gives a high level idea of the problem at hand: We have observed data in the form of motion trajectories and maps for training (left), and provided an occupancy map of a novel environment (upper right), we wish to generate likely motion patterns (lower right).

We assume to have a training dataset consisting of occupancy representations of diverse environments and a collection of trajectories observed within each map. We denote the dataset as $\mathcal{D} = \{\mathcal{M}_n, \{\xi_p\}_{p=1}^{P_n}\}_{n=1}^{N}$, where there are N maps and corresponding sets of observed trajectories, $\mathcal{M}_n$ is the n^{th} occupancy map, and $\{\xi_p\}_{p=1}^{P_n}$ is the set of P_n trajectories collected in the corresponding environment. Note that the number of trajectories in each environment may differ.

We now have a new map, $\mathcal{M}^*$, where no trajectory observations have been made, and seek to generate new likely motion trajectories on $\mathcal{M}^*$, based on patterns we see in our training dataset. Key requirements of a solution to the problem include: (1) the ability to generate trajectories are diverse enough to capture multiple trajectory patterns; (2) the ability to generate trajectories that start from a specified coordinate.

6.2.2 Overview of OTNet

On a high level, our method encodes maps as similarities between themselves and a predefined collection of maps, and represents trajectories as parameters of a function. A mixture density network (Bishop, 1994) is then used to predict distributions over function parameters, conditional on an encoded map.

The training process is illustrated in Figure 6.2, and can be summarised as:

1. Construct feature vectors of similarities, $\boldsymbol{\phi}$, from each training map. Intuitively, we similarly shaped floor-plans to have similar motion patterns. Our map encoding explicitly

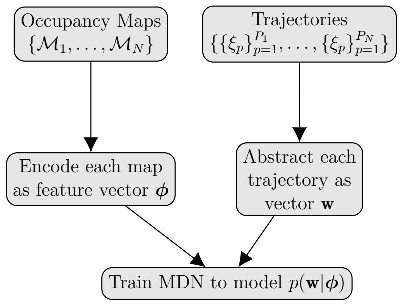

Occupancy Maps $\{\mathcal{M}_1, \ldots, \mathcal{M}_N\}$
Trajectories $\{\{\xi_p\}_{p=1}^{P_1}, \ldots, \{\xi_p\}_{p=1}^{P_N}\}$
Encode each map as feature vector $\boldsymbol{\phi}$
Abstract each trajectory as vector $\mathbf{w}$
Train MDN to model $p(\mathbf{w}|\boldsymbol{\phi})$

Figure 6.2: Process of learning model to generate $p(\mathbf{w}|\boldsymbol{\phi})$

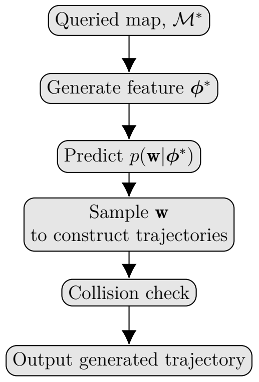

Queried map, $\mathcal{M}^*$
Generate feature $\boldsymbol{\phi}^*$
Predict $p(\mathbf{w}|\boldsymbol{\phi}^*)$
Sample $\mathbf{w}$ to construct trajectories
Collision check
Output generated trajectory

Figure 6.3: Process of generating trajectories

introduces the notion of map similarity to the learning process. Details in Section 6.2.3.

2. Following the continuous trajectory representation outlined in chapter 5, We represent trajectory data as vectors of weight parameters, $\mathbf{w}$, of a function. This allows us to operate on on trajectory data sequences which contain differing numbers of waypoints. Each trajectory can typically be represented with much fewer parameters than waypoint coordinates.

3. We use a mixture density network (MDN) (Bishop, 1994) to learn the distribution over weight parameters conditioned on the map feature vectors, $p(\mathbf{w}|\boldsymbol{\phi})$. Details in Section 6.2.4.

A brief overview of the generative process is illustrated in Figure 6.3.

After the MDN has been trained, we construct a feature vector $\boldsymbol{\phi}^*$ of a new map $\mathcal{M}^*$, and query the MDN to obtain $p(\mathbf{w}|\boldsymbol{\phi}^*)$. Vectors of $\mathbf{w}$ can be sampled from $p(\mathbf{w}|\boldsymbol{\phi}^*)$, and each sample $\mathbf{w}$ can be used to generate a new trajectory. As there are no explicit constraints in the MDN to prevent trajectories from overlapping with occupied regions, and we can efficient sample check for collisions, we accept collision-free trajectories to output. If we are only interested in trajectories that start at a certain point, we can generate trajectories conditional on a specified start-point.

6.2.3 Encoding of Environmental Occupancy

We represent occupancy maps as a vector of similarities between the given environment and a representative database of other maps. Intuitively, we can think of this as operating in the *space of maps*, where we pin-point the map of interest by its relation with other maps, where maps similar to one another are closer together in the space of map. We expect similar maps to have similar motion trajectory patterns. Related ideas have been explored in the context of pseudo-inputs for sparse Gaussian process (Snelson and Ghahramani, 2006).

The Hausdorff distance is a widely used distance measure to shapes and images (Huttenlocher et al., 1993), and can be efficiently computed in linear time. The Hausdorff distance measures the distance between two finite sets of points, and allows us to make comparisons between our maps. We place the *Hausdorff distance* into a *distance substitute kernel* (Haasdonk and Bahlmann, 2004), to obtain our similarity function.

Given two sets of points $A = \{a_1, a_2, \dots, a_n\}$ and $B = \{b_1, b_2, \dots, b_m\}$, and in general n and m are not required to be equal, the one-sided Hausdorff distance between the two sets is defined as:

$$\hat{\delta}_H(A, B) = \max_{a \in A} \min_{b \in B} ||a - b||. \tag{6.1}$$

The one-sided Hausdorff distance is not symmetric, we enforce symmetry by taking the average of $\hat{\delta}_H(A, B)$ and $\hat{\delta}_H(B, A)$, i.e.:

$$\delta_H(A, B) = \frac{1}{2}(\hat{\delta}_H(A, B) + \hat{\delta}_H(B, A)). \tag{6.2}$$

We can then define a similarity function between two sets A and B, analogous to a distance substitute kernel described in (Haasdonk and Bahlmann, 2004), as:

$$S_H(A, B) = \exp\Big\{ - \frac{\delta_H(A, B)^2}{2\ell_H} \Big\}, \tag{6.3}$$

where ℓ_H is a length scale hyper-parameter.

We extract occupied edge points of binary grid maps and evaluate the similarity function between each map in the representative database. We can select a set of maps within our training data to be in the representative database, or when the number of maps is low, we can consider all the training maps to be in the database. A simple method of selecting the set of examples to be in a representative database are described in (Zhi et al., 2019b). If we have N training maps and M representative maps in the database, feature vector for the n^{th} map, $\boldsymbol{\phi}_n$, is:

$$\begin{bmatrix} \boldsymbol{\phi_1} \\ \vdots \\ \boldsymbol{\phi_N} \end{bmatrix} = \begin{bmatrix} S_H(\mathcal{M}_1, \mathcal{M}_1) & \dots & S_H(\mathcal{M}_1, \mathcal{M}_M) \\ \vdots & \ddots & \vdots \\ S_H(\mathcal{M}_N, \mathcal{M}_1), & \dots, & S_H(\mathcal{M}_N, \mathcal{M}_M). \end{bmatrix}. \tag{6.4}$$

For every map $\mathcal{M}$ in our dataset, there is a corresponding vector of similarities $\boldsymbol{\phi} \in \mathbb{R}^M$. The m^{th} element in the feature vector $\boldsymbol{\phi}_n$ denotes the similarity between the n^{th} occupancy map in the training dataset, and the m^{th} occupancy map in the database.

6.2.4 Learning a Mixture of Stochastic Processes

Like section 5.4.2, we begin by modelling motion as continuous trajectories. We differ slightly from section 5.4.2, by considering the x and y dimensions independently. This produces fewer parameters we shall need to eventually learn. We define a normalised timestep parameter $\tau \in [0, 1]$. A continuous trajectory can be modelled function, $\Xi(\tau) = [x(\tau), y(\tau)]$, which maps τ to the x and y coordinates of the trajectory. We model $x(\tau)$ and $y(\tau)$ as weighted sums of M fixed radial basis functions centred on evenly spaced τ values, with weight vectors $\mathbf{w}_x$ and $\mathbf{w}_y$. These weight parameters can be found from discrete trajectory data ξ via solving regression equations akin to . We can then query the trajectory coordinates at τ^* by evaluating $x(\tau^*) = \mathbf{w}_x^T \mathbf{k}(\tau^*)$ and $y(\tau^*) = \mathbf{w}_y^T \mathbf{k}(\tau^*)$. We denote the concatenation of $\mathbf{w}_x$ and $\mathbf{w}_y$ as $\mathbf{w} = [\mathbf{w}_x, \mathbf{w}_y]^T \in \mathbb{R}^{2M}$.

Recall that from Section 6.2.3, each occupancy representation is encoded as a vector of similarities $\boldsymbol{\phi}$. Our goal is now to estimate $p(\mathbf{w}|\boldsymbol{\phi})$, the conditional distribution over trajectory parameters. There may exist many distinct groupings of trajectories in each environment. The distribution of trajectories is expected to be multi-modal, and we need to use a model capable of predicting multi-modal conditional distributions over the weights, $p(\mathbf{w}|\boldsymbol{\phi})$. Mixture density networks (MDN) (Bishop, 1994) are a class of neural networks capable of representing conditional distributions. We slightly modify the classical MDN described in (Bishop, 1994) to learn a mixture of vectors of conditional distributions, corresponding to the conditional distribution for each element in $\mathbf{w}$. We model the conditional distribution $p(\mathbf{w}|\boldsymbol{\phi})$ as a mixture of Q vectors of distributions, which we call mixture components. To reduce the number of parameters to predict, we make the mean-field Gaussian assumption

(Bishop, 2007) on the weights giving,

$$p(\mathbf{w}|\boldsymbol{\phi}) = \sum_{q=1}^{Q} \alpha_q p_q(\mathbf{w}|\boldsymbol{\phi}) = \sum_{q=1}^{Q} \alpha_q \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_q, \boldsymbol{\Sigma}_q), \tag{6.5}$$

where $p_q(\mathbf{w}|\boldsymbol{\phi})$ denotes the q^{th} component, and α_q is the associated component weight. From the mean-field Gaussian assumption, each mixture component has mean vector, $\boldsymbol{\mu}_q = [\mu_{q,1}, \mu_{q,2}, \ldots, \mu_{q,2M}]^T$, and diagonal covariance, $\text{diag}(\boldsymbol{\Sigma}_q) = \boldsymbol{\Sigma}_q^2 = [\boldsymbol{\Sigma}_{q,1}^2, \boldsymbol{\Sigma}_{q,2}^2, \ldots, \boldsymbol{\Sigma}_{q,2M}^2]^T$. We can write each component of the conditional distribution as:

$$p_q(\mathbf{w}|\boldsymbol{\phi}) = \prod_{m=1}^{2M} \frac{1}{\sqrt{2\pi\boldsymbol{\Sigma}_{q,m}^2}} \exp\Big\{ - \frac{(w_m - \mu_{q,m})^2}{2\boldsymbol{\Sigma}_{q,m}^2} \Big\}, \tag{6.6}$$

giving us the negative log-likelihood loss function over N maps, and P_n trajectories observed in the environment corresponding to the n^{th} map in the dataset as:

$$\mathcal{L}(\boldsymbol{\theta}) = -\log \Big[\prod_{n=1}^{N} \prod_{p=1}^{P_n} \sum_{q=1}^{Q} \alpha_q p_q(\mathbf{w}|\boldsymbol{\phi}) \Big], \tag{6.7}$$

where we denote the set of parameters to optimise as $\boldsymbol{\theta} = \{\alpha_q, \boldsymbol{\mu}_q, \boldsymbol{\Sigma}_q\}_{q=1}^{Q}$. Using a neural network to minimise the loss function defined in Equation (6.7), we can learn a model that maps from the feature vector of similarities $\boldsymbol{\phi}$ to the parameters required to construct $p(\mathbf{w}|\boldsymbol{\phi})$. The neural network is relatively simple with a sequence of fully-connected layers.

The standard MDN constraints are applied using the activation functions highlighted in (Bishop, 1994). This includes:

1. $\sum_{q=1}^{Q} \alpha_q = 1$, such that component weights sum up to one, by applying the softmax activation function on associated network outputs;

2. $\boldsymbol{\Sigma}_{q,m} \geq 0$, by applying an exponential activation function on associated network outputs.

As a distribution is estimated for each of the elements in weight vector, $\mathbf{w}$, the predicted $p(\mathbf{w}|\boldsymbol{\phi})$ results in a mixture of discrete processes, where each realisation is a vector of $\mathbf{w}$. $\mathbf{k}$ denotes the basis functions.

6.2.5 Trajectory Generation and Conditioning

After we complete the training of our MDN model, we can generate trajectories in environments with no observed trajectories. We generate the feature vector of similarities, $\boldsymbol{\phi}^*$, from the map of interest, $\mathcal{M}^*$, and into the MDN. We obtain parameters, that define conditional distributions over $\mathbf{w}$. Realisations of $\mathbf{w}$ can be sampled randomly from the predicted $p(\mathbf{w}|\boldsymbol{\phi}^*)$, and a possible continuous trajectory, Ξ, can be found by evaluating

$$\Xi(\tau) = [x(\tau), y(\tau)]^\top = [\mathbf{w}_x\mathbf{k}(\tau), \mathbf{w}_y\mathbf{k}(\tau)]^\top, \tag{6.8}$$

where $\mathbf{k}(\tau)$ gives evaluations of the basis functions for the continuous trajectories. As there are no explicit constraints in the MDN to prevent the generation of trajectories which overlap with occupied regions, we apply collision checking. The trajectories can be generated and checked very efficiently, as it involves randomly sampling a mixture of Gaussian distributions and checking a map.

In order to predict how agents observed at a known position move, we are often interested in generating likely trajectories which begin at a certain start-point. To achieve this, let us consider the distribution of trajectories x, y-coordinates. Intuitively, rather than the distribution on weight parameters $\mathbf{w}$, we want to consider the joint distribution between trajectory coordinates. We evaluate the continuous trajectories at a set of L times of interest, $\boldsymbol{\tau} = [\tau_1, \tau_2 \dots \tau_L]$,

$$p(\Xi(\boldsymbol{\tau})) = p\Big(\begin{bmatrix}\mathbf{w}_x^T\mathbf{K}(\boldsymbol{\tau})\\ \mathbf{w}_y^T\mathbf{K}(\boldsymbol{\tau})\end{bmatrix}\Big) = \sum_{q=1}^{Q}\alpha_q\begin{bmatrix}\mathcal{N}(\hat{\boldsymbol{\mu}}_q^x, \hat{\boldsymbol{\Sigma}}_q^x)\\ \mathcal{N}(\hat{\boldsymbol{\mu}}_q^y, \hat{\boldsymbol{\Sigma}}_q^y)\end{bmatrix}, \tag{6.9}$$

where $\mathbf{K}(\boldsymbol{\tau}) = [\mathbf{k}(\tau_1), \mathbf{k}(\tau_2), \dots, \mathbf{k}(\tau_L)]^T$ is a matrix containing basis function evaluations at L time points of interest $\boldsymbol{\tau}$. We can now find mean and covariance parameters $(\hat{\boldsymbol{\mu}}_q^x, \hat{\boldsymbol{\mu}}_q^y, \hat{\boldsymbol{\Sigma}}_q^x, \hat{\boldsymbol{\Sigma}}_q^y)$ for the trajectory coordinates at times of interest as,

$$\hat{\boldsymbol{\mu}}_q^x = \boldsymbol{\mu}_q^{xT}\mathbf{K}(\boldsymbol{\tau}), \qquad \hat{\boldsymbol{\Sigma}}_q^x = \mathbf{K}(\boldsymbol{\tau})\boldsymbol{\Sigma}_q^x\mathbf{K}(\boldsymbol{\tau})^T, \tag{6.10}$$

$$\hat{\boldsymbol{\mu}}_q^y = \boldsymbol{\mu}_q^{yT}\mathbf{K}(\boldsymbol{\tau}), \qquad \hat{\boldsymbol{\Sigma}}_q^y = \mathbf{K}(\boldsymbol{\tau})\boldsymbol{\Sigma}_q^y\mathbf{K}(\boldsymbol{\tau})^T, \tag{6.11}$$

The covariances of the q^{th} component between the elements in weight parameter vectors $\mathbf{w}_x$ and $\mathbf{w}_y$ are denoted as $\boldsymbol{\Sigma}_q^x$ and $\boldsymbol{\Sigma}_q^y$ respectively. The component covariance $\boldsymbol{\Sigma}_q$ between all the weights $\mathbf{w} = [\mathbf{w}_x, \mathbf{w}_y]$ and $\mathbf{\Sigma}^x_q$, $\mathbf{\Sigma}^y_q$ are $\text{diag}(\mathbf{\Sigma}_q) = [\text{diag}(\mathbf{\Sigma}^x_q), \text{diag}(\mathbf{\Sigma}^y_q)]$. Note that although $\mathbf{\Sigma}_q$ is diagonal, the covariance of Gaussian components between times of interest $\hat{\mathbf{\Sigma}}^x_q, \hat{\mathbf{\Sigma}}^y_q$ are typically fully-connect.

For any component, we can then enforce locations that the trajectories must pass through (x^*, y^*) at τ_i, by finding the distributions of coordinates the times of interest conditioned on the fixed point. Consider when we wish to condition on the start point, i.e. $\tau_1 = (x^*, y^*)$, a single component in the mixture can be written as:

$$p_q(\Xi(\boldsymbol{\tau}_{1:L})|\Xi(\boldsymbol{\tau}_1) = [x^*, y^*]) = \alpha_q \begin{bmatrix} \mathcal{N}(\bar{\boldsymbol{\mu}}^x, \bar{\mathbf{\Sigma}}^x) \\ \mathcal{N}(\bar{\boldsymbol{\mu}}^y, \bar{\mathbf{\Sigma}}^y) \end{bmatrix}. \tag{6.12}$$

If we consider elements in mean vectors $\hat{\boldsymbol{\mu}}^x$, $\hat{\boldsymbol{\mu}}^y$ and covariance matrices $\hat{\mathbf{\Sigma}}^x$, $\hat{\mathbf{\Sigma}}^y$ of the unconditioned distribution as:

$$\hat{\boldsymbol{\mu}}^z = \begin{bmatrix} \hat{\boldsymbol{\mu}}^z_1 \\ \hat{\boldsymbol{\mu}}^z_{2:L} \end{bmatrix}, \qquad \hat{\mathbf{\Sigma}}^z = \begin{bmatrix} \hat{\mathbf{\Sigma}}^z_{1,1} & {\hat{\mathbf{\Sigma}}^z_{1,2:L}}^T \\ \hat{\mathbf{\Sigma}}^z_{1,2:L} & \hat{\mathbf{\Sigma}}^z_{2:L,2:L} \end{bmatrix}, \qquad \text{for } z = \{x, y\}, \tag{6.13}$$

then for $z = \{x, y\}$, we can express the parameters of the conditional distribution as (Bishop, 2007):

$$\bar{\boldsymbol{\mu}}^z = \hat{\boldsymbol{\mu}}^z_{2:L} + \hat{\mathbf{\Sigma}}^z_{1,2:L} {\hat{\mathbf{\Sigma}}^z_{1,1}}^{-1}(z^* - \hat{\boldsymbol{\mu}}_1) \tag{6.14}$$

$$\bar{\mathbf{\Sigma}}^z = \hat{\mathbf{\Sigma}}^z_{2:L,2:L} - \hat{\mathbf{\Sigma}}^z_{1,2:L} {\hat{\mathbf{\Sigma}}^z_{1,1}}^{-1} {\hat{\mathbf{\Sigma}}^z_{1,2:L}}^T. \tag{6.15}$$

We can condition each mixture component to start at a certain point, or employ strategies to only condition selected mixture components, such as the nearest component to the conditioned coordinate, and only generate trajectories belonging to the conditioned mixture component.

6.3 Experimental Results and Discussion

6.3.1 Dataset and Metrics

Training an OTNet requires a dataset containing occupancy maps of multiple environments along with observed trajectories in each environment. To the best of our knowledge, there exists no real-world dataset of sufficient size with different occupancy maps and trajectories observed in each of the different environments. Therefore, we conduct our experiments with our simulated dataset, *Occ-Traj120* (Lai et al., 2019). This dataset contains 120 binary occupancy grid maps of indoor environments with rooms and corridors, as well as simulated motion trajectories. We aim to utilise our method to learn transfer the motion to new environments. The dataset is split with a 80-20 ratio for training and testing respectively. Our aim is to transfer the trajectory behaviour, from a training subset to unseen test environments.

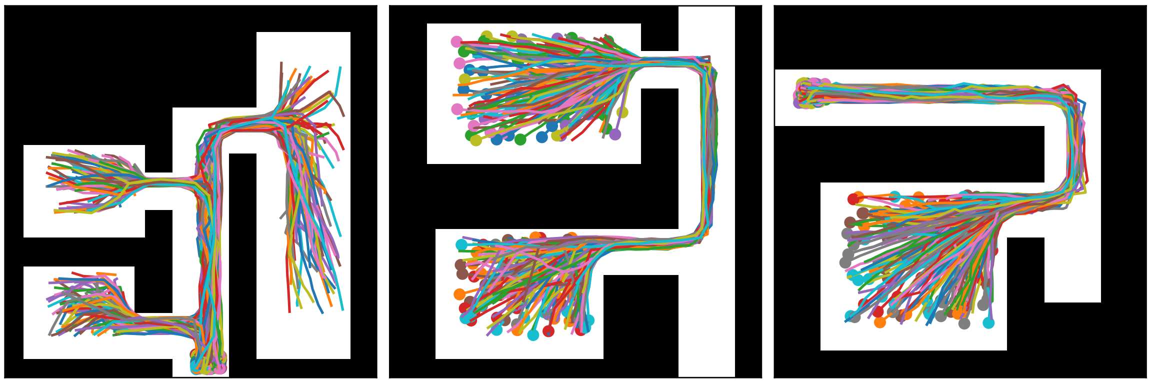


Figure 6.4: Occupancy maps in the Occ-Traj 120 dataset, along with associated trajectories. We aim to generalise trajectories to unseen maps.

We evaluate the generated trajectories against a test set with hidden ground truth trajectories. Continuous trajectories outputted are discretised for evaluation by querying at uniform intervals. Due to the probabilistic and multi-modal nature of our output, the metric used is *minimum trajectory distance* (MTD), and is defined by: $MTD = \min_{i=1,\ldots,P} \mathcal{D}(\xi_{gen}, \xi_i)$, where P denotes the number of trajectories observed in the environment, with i indexing each trajectory, and $\mathcal{D}(\xi_{gen}, \xi_i)$ is a distance measure of trajectory distance between the generated ξ_{gen} and a ground truth trajectory ξ_i. In our evaluations the Euclidean *Hausdoff distance* and *discrete Frechét distance* are considered. These trajectory distances are commonly used in distance-based trajectory clustering to quantify the dissimilarity between trajectories, and a review of these distances can be found in (Besse et al., 2016). We also wish to evaluate the quality of the uncertainty captured. To this end, we calculate the average negative log-likelihood (ANLL) between the predicted distribution at 100 uniform time-steps and each ground truth trajectory. When only samples of generated trajectories are available, we fit Gaussian distributions over world-space coordinates at uniform time-coordinates. ANLL can take into account of probabilistic multi-modal distributions, and a relative lower ANLL

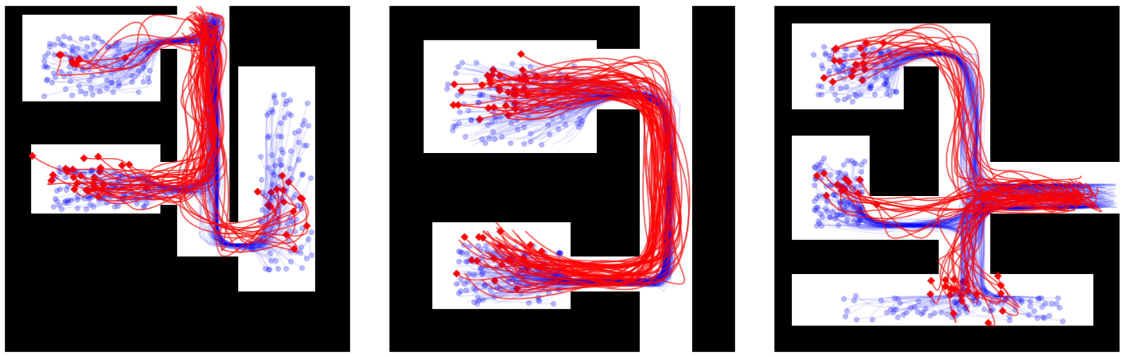


Figure 6.5: Generated likely motion trajectories (in red) in new environments. End points are indicated by scatter points. OTNet captures the probabilistic, multi-modal nature of motion trajectories. The ground truth trajectories (in blue) are hidden during training.

indicates better performance.

6.3.2 Experimental Setup

We compare our proposed method, OTNet, with neural-network based generative models, a learning-based motion reconstruction method, and a k-nearest neighbour (k-NN) based method. The details of these models are as follows:

OTNet

We train OTNet with the length-scale hyper-parameters $\ell_H = 50$ and $\ell_b = 5$ for 20 epochs. The number of bases are set to be $M = 15$, and the number of mixture components, $Q = 4$. These hyper-parameters are hand-picked and fairly not sensitive, with values in the same ball-park giving similar results. Cross-validation could be applied for further improvements. As the number of map examples in the training set is relatively small at 96, we select all our training maps to be in our database of maps, when encoding maps of interest as feature vectors. In each of our experiments, trajectories can be generated efficient during test time. On a standard desktop machine, predicting the distribution of trajectories can be done in under one second, negligible time is required to randomly sample the distribution to generate trajectories.

Generative network models

Our proposed method is generative, we evaluate two popular generative models: GANs (Goodfellow et al., 2014) CVAEs (Sohn et al., 2015), trained for 300 epochs to generate trajectory parameters $\mathbf{w}$. The discriminator uses convolutional layers, and the generator samples a 100-dimensional latent vector, concatenated with the map for conditioning. Five fully-connect layers are used to output $\mathbf{w}$. The hyperparameters of the CVAE model, are identical to the GAN model.

Learning-based motion reconstruction

We evaluate the performance of the learning-based MPNet (Qureshi et al., 2019), which conditions on the environment and imitates training trajectories. We pre-train on MPNet's public 2D dataset with further training with the *Occ-Traj120 dataset* for 300 epochs. MPNet requires trajectory start and end coordinates, during evaluation we provide ground truth mean start and end points of groups of trajectories. To guarantee valid trajectories, we use the hybrid MPNet-RRT variant suggested by the authors. We emphasise that MPNet can only generate a single prediction, and cannot generate distributions of trajectories.

Nearest-neighbour

We also consider a baseline k-NN based approach. Using the symmetric Hausdoff distance, we find the nearest map training map, and transfer the trajectories directly to the queried map. As we cannot average trajectories, we use $k = 1$. Note storage of not only all maps, but also trajectories are required.

6.3.3 Comparisons to Other Generative Models

The performance results of our experiments are tabulated in Table 6.1, we see that OTNet outperform the other generative models compared. The ANLL for CVAE and GANs suffer from a loss of precision due to an extremely low likelihood, resulting in taking log of a near 0 value, the ANLL will be much higher than OTNet. The ANLL for MPNet-RRT is also not applicable, as it generates single trajectories. The map representations are of too high dimensions to directly train neural networks with a limited dataset. In particular, the encoding of each occupancy map as a feature vector of similarities, $\boldsymbol{\phi}$, allows for flexible

	Hausdoff	Frechet	ANLL
OTNet	1.98	2.13	29.83
GANs	11.79	16.66	NA
CVAE	9.48	14.67	NA
MPNet-RRT	2.99	5.14	NA
k-NN	2.03	2.14	31.46

Table 6.1: Performance of OTNet and comparisons models. Lower values are better. OTNet outperforms the comparisons, although the performance of k-NNs are strong, k-NNs lack the ability to generate new trajectories that do not exist in the provided dataset.

representations even when the number of maps in the dataset is small. Comparatively, other neural network-based models which attempt to directly extract map features using convolutional neural networks are unable to successfully condition on the occupancy maps. The nearest neighbour model, which transfers the trajectories of the nearest map, performs surprising well, comparable to the OTNet on the simulated on the Haussdoff and Frechét measures. This is due to OTNet often containing several relatively similar-looking maps. However, OTNet is able to take into consideration several maps, and weigh each by their similarity with the map of interest, while the nearest neighbour only considers the closest map. We also note that the single nearest neighbour approach often results in relative over-confidence of biased predictions, hence the higher ANLL relative to OTNet. We note that we can directly obtain closed form equations of trajectory distributions from OTNet, allowing us to repeatedly generate new trajectories. Whereas, MPNet-RRT produces single predictions. Although using k-NN can give trajectories with competitive performance, it cannot generate new trajectories, whereas OTNet is capable of generating trajectories that have not been observed in the original dataset. Examples of trajectories generated by OTNet are shown in red in Figure 6.5, along with ground truth trajectories (blue).

6.3.4 Diverse Trajectories and Conditioning Trajectory Start Points

We also want to further investigate whether generated trajectories capture different groups of motion, and whether we can specify start-points for generated trajectories. Figure 6.7 shows generated trajectories in an unseen test environment, along with plots of the trajectory coordinates wrt the normalised time parameter, τ. The test environment is an indoor environment of a corridor with a connected room. Predicted trajectories are coloured in

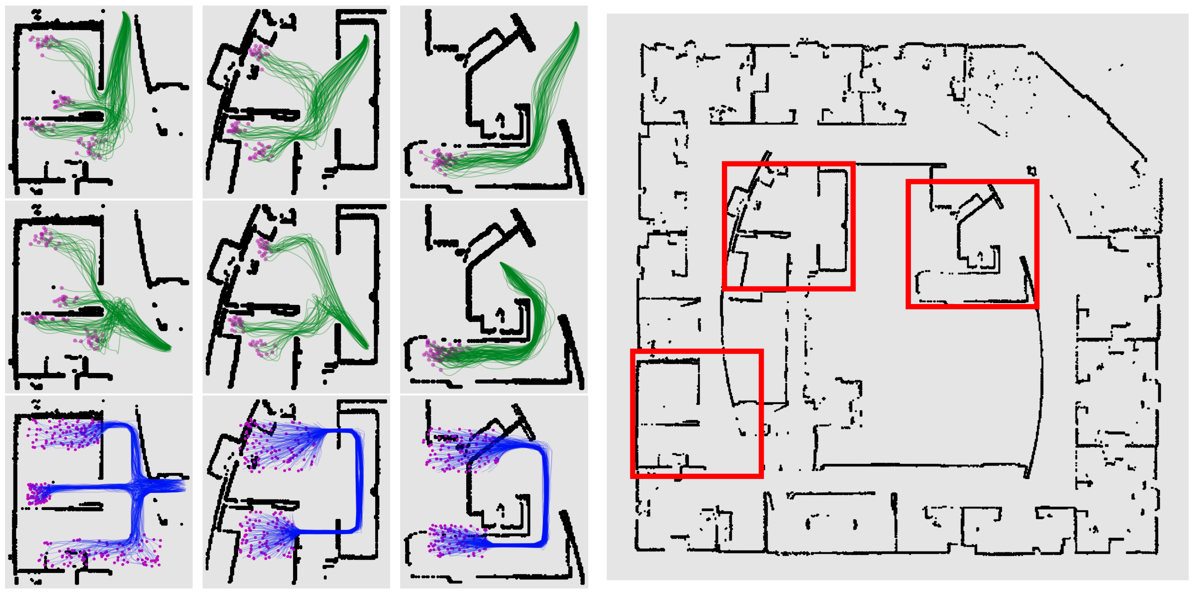


Figure 6.6: After training on a simulated dataset, we condition on sections of real world occupancy data, from the Intel-Labs dataset (Hahnel et al., 2003), to generate motion trajectories (top two rows). The start point of the trajectories are fixed, with the end-points illustrated with magenta markers. We can compare this to trajectories from a baseline nearest neighbour model (bottom row), which transfers trajectories from the most similar training map, and is unable to adapt to the new environments.

red, and hidden ground truth trajectories are under-laid in blue. The end coordinates of each trajectory have an attached marker. We clearly observe that the trajectories generated can belong to different groupings – one group starts from inside the room and exit into the corridor, while the other starts in the corridor and end in the room. The multi-modality of the distributions of trajectories learned is even more clear when observing the plots of coordinates against τ. If we observe the y-coordinates, $y(\tau)$, we clearly see two groupings of functions, one of which is at around $x(0) = 15$ and around $y(1) = 32$, the other grouping has values around $x(0) = 32$ and $y(1) = 15$.

In the same environment, we attempt to condition the generated trajectories to different start-points. The generated trajectories (top), as well as trajectory coordinates $x(\tau)$ (middle) and $y(\tau)$ (bottom) of the trajectories, are shown in Figure 6.8. The generated trajectories are in red, with red marker end-points, and black start-points. In the left and centre plots, we condition the trajectories' start point on two coordinates in the room, $(x(0) = 5, y(0) = 20)$ (left) and $(x(0) = 5, y(0) = 5)$ (centre), as well conditioning on a start-point in the corridor,

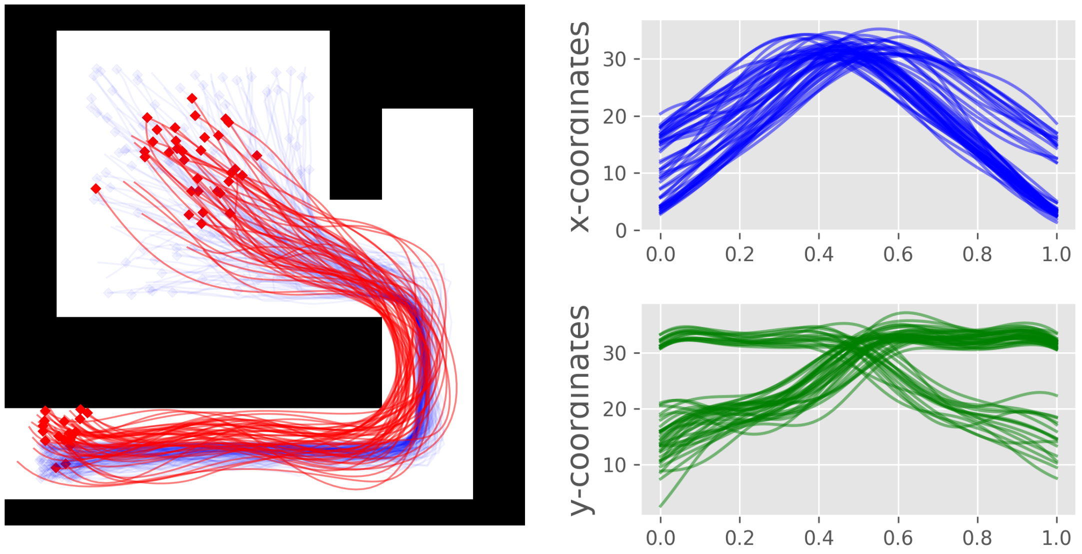


Figure 6.7: (Left) Generated trajectories (red) overlaid on the corridor and room test map, with markers indicating the endpoints. The hidden ground truth trajectories are under-laid in blue. We see two different groups of trajectories: one starts from inside the room and end in the corridor, the other moves in the opposite direction. (Right) Plots of coordinates wrt τ, $x(\tau)$ and $y(\tau)$ (top, bottom respectively), corresponding to the trajectories generated on the left plot. The ability to generate distinct groups of trajectories is clear.

$(x(0) = 2, y(0) = 35)$. We see that in all three cases, OTNet was able to generate reasonable trajectories, which follow the environment structure, with each starting at their designated start coordinates.

6.3.5 Transferring Trajectories in Simulated Environments to Real-world Data

We also investigate the transfer of trajectories from simulated maps to real-world environments. We select three different regions in the Intel-lab dataset (Hahnel et al., 2003), and transfer trajectory patterns trained on the simulated Occ-Traj dataset. Figure 6.6 shows trajectories (top two rows, green with magenta end-points) generated on three different sub-maps, and conditioned on two different starting locations. We see that OTNet is able to generate multi-modal distributions of trajectories, with distinct groupings, and largely conforms to the environmental geometry, even though the real-world occupancy differs from the

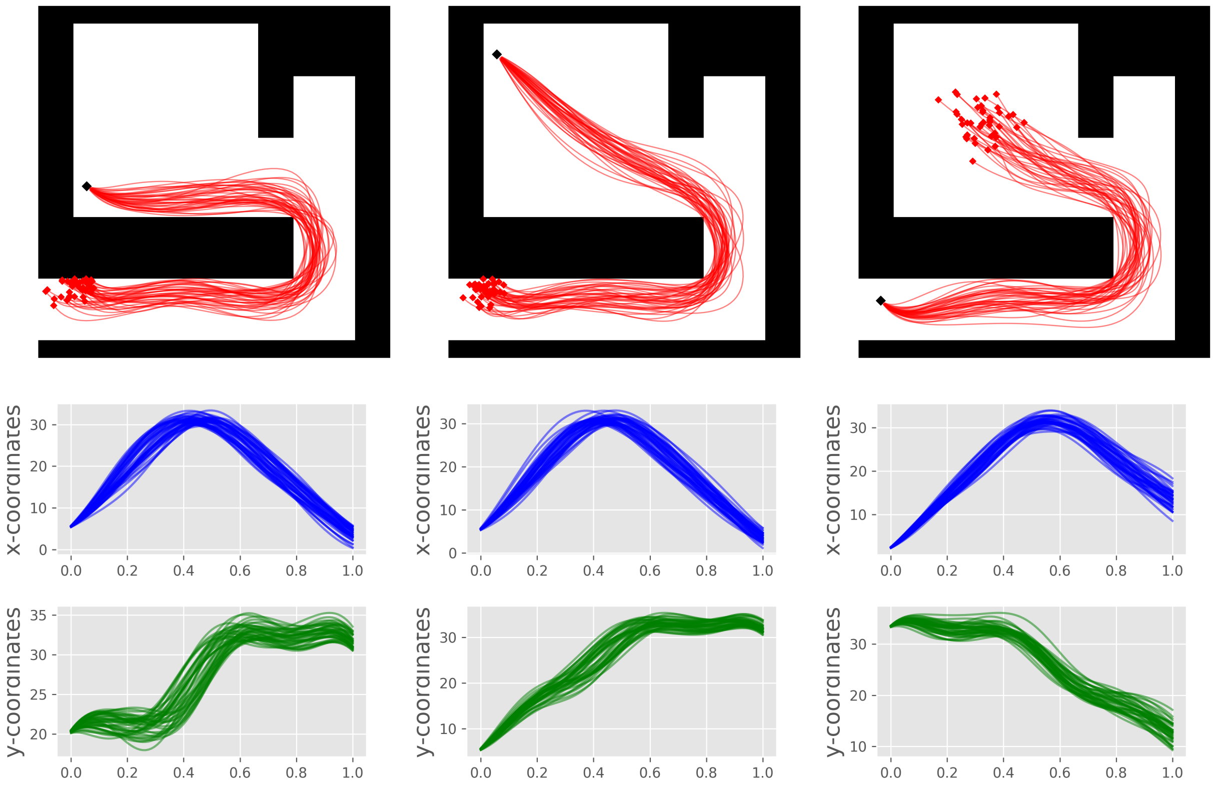


Figure 6.8: We generate trajectories (in red, with red markers as endpoints) conditional on the start points (black marker). We condition on two start points inside the room (left, centre), and one point in the corridor (right). The corresponding plots of x, y-coordinates wrt to τ, $x(\tau)$ and $y(\tau)$ (top, bottom respectively). We see that the trajectories can be conditioned on a variety of start points.

simulated training environments significantly. We visually compare trajectories generated by OTNet to those by the 1-nearest neighbour approach (bottom row, blue with magenta end-points), which transfers the trajectories from the most similar map to the queried map. Qualitatively, we evaluate the trajectories generated by OTNet to conform better to the real-world environment compared to the nearest-neighbour trajectories. The 1-nearest neighbour approach can only draw from the most similar map. We note that the 1-nearest neighbour approach performs well when transferring to the simulated dataset, as it could often find a relatively similar map that has been observed. Whereas, the real-world occupancy map is not very similar to any single map we have experienced, but OTNet can generalise trajectories from a combination of maps which resembles the test environment closer. Hence, OTNet is more capable of generalising to unseen environments. The nearest neighbour approach is also unable to immediately generate similar new trajectories beyond those included in the data, nor is it possible to condition predicted trajectories on a start-point. Both generating similar new trajectories, and conditioning on points can be achieved with OTNet.

6.4 Summary

We present a novel generative model, OTNet, capable of producing likely motion trajectories in new environments where no motion has been observed. We generalise observed motion trajectories in alternative training environments. The OTNet encodes maps as a feature vector of similarities, and embeds observed trajectories as function parameters. A neural network is used to learn conditional distributions over the parameters. Realisations of the vectors can then be sampled from the conditional distribution, and used to reconstruct generated trajectories. We empirically show the strong performance of OTNet against popular generative methods. Further improvements on OTNet could include incorporating temporal changes in trajectory patterns into the framework.

In the following chapter 7, we take an alternative combined learning and optimisation approach to accounting for environment structure when understanding motion patterns. This allows for hard-constraints to be applied on the distribution of trajectories predicted.

Chapter 7

Structurally Constrained Motion Prediction

7.1 Introduction

Autonomous robots, such as service robots and self-driving vehicles, are often required to coexist in environments with other dynamic objects. For robots to safely navigate and plan in an anticipatory manner in dynamic environments, accurate motion predictions for other nearby objects are needed. Many recent developments Alahi et al. (2016); Gupta et al. (2018); Zhi et al. (2019) in motion prediction have relied on learning based approaches, whereby future positions, or probabilities over future positions, are learned from motion trajectories of previously observed dynamic objects. Learning-based methods to motion prediction typically extract patterns from the training data. These patterns are then used to extrapolate future positions based on new observations without explicitly considering the feasibility of the prediction.

However, there are often rules for motion trajectories that are known *a priori*, but are not explicitly enforced during extrapolation by learning-based models. In particular, we address the problem of incorporating constraints that arise from the environment's structure, such as obstacles, into motion predictions. In most environments, motion trajectories are not highly

This chapter has been published in IROS as Zhi et al. (2021c).

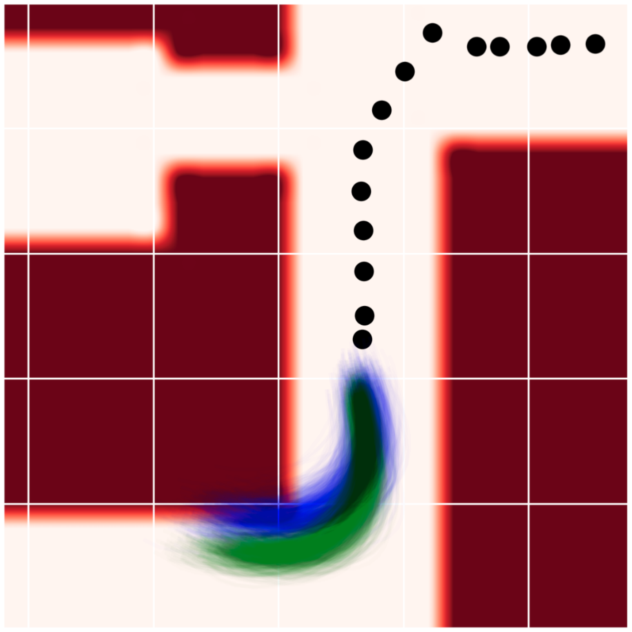


Figure 7.1: A simple example of enforcing predictions to be consistent with environment: Provided an observed trajectory segment (black), a purely learning method may predict collision-prone distributions of trajectories (blue). Our framework formulates an optimisation problem with chance constraints arising from environmental structure, as given by an occupancy map. We solve to obtain predictions (green) which are compliant with the environment, while remaining similar to the learned future.

unstructured, and depend closely on the environment occupancy structure.

To address the shortcomings of purely learning-based methods, we view motion prediction as a combined probabilistic trajectory learning and constrained trajectory optimisation problem. Constrained optimisation methods allow for constraints to be imposed directly onto the distribution of trajectories without incorporating the collected data. We propose a novel framework that incorporates structural constraints into probabilistic motion prediction problems. Our framework leverages the ability of learning based models to extract patterns from data, as well as the ability of constrained trajectory optimisation approaches to explicitly specify constraints, giving better generalisability and higher quality extrapolations. Specifically, our contributions consist of: (1) a method to learn a distribution over trajectories from observations that is amenable to being optimised; (2) an approach to enforce constraints on distributions of trajectories, particularly collision chance constraints when provided an occupancy map.

7.2 Related Work

Motion Trajectory Prediction

Motion prediction is a problem central to robot autonomy. The most simple methods of motion prediction are physics-based models, such as constant velocity/ acceleration models (Schubert et al., 2008). More complex physics-based models often fuse multiple dynamic models, or select a best fit model from a set (Kooij et al., 2018). Recent developments in machine learning have led to learning-based methods that learn motion patterns (Alahi et al., 2016; Zhi et al., 2019b) from a dataset of observed trajectories. In particular, long-short term memory (LSTM) and generative adversarial network (GAN)-based models (Alahi et al., 2016; Zhang et al., 2019; Amirian et al., 2019; Gupta et al., 2018) have gained popularity. Methods of learning entire trajectories, similar to our learning component, but with relaxed dependencies, is outlined in Zhi et al. (2019, 2021a). Further attempts to incorporate static obstacles in learning attempt to do so as a part of the learning pipeline (Sadeghian et al., 2019) and attempts to extrapolate behaviour. Approaches of this kind cannot guarantee that valid solutions satisfy specified chance constraints, and there have not been approaches that directly incorporate occupancy maps, which are typically built by robots from depth sensor data.

Trajectory Optimisation

Trajectory optimisation has been successfully applied in motion planning, by directly expressing obstacles as constraints to solve for collision-free paths between start and goal points. Popular methods include *Trajopt* (Schulman et al., 2013), *CHOMP* (Zucker et al., 2013), *STOMP* (Kalakrishnan et al., 2011), and *GPMP2* (Dong et al., 2016). Trajopt utilises sequential quadratic programming to find locally optimal trajectories compliant with constraints. CHOMP exploits obstacle gradients for efficient optimisation. Further work on gradient-based trajectory optimisation for planning can be found in Marinho et al. (2016); Francis et al. (2019). Although trajectory optimisation is a popular choice to obtain collision-free trajectories for motion-planning, there has yet to be extensive work in utilising trajectory optimisation for motion prediction. The optimisation required in this work is also different in that we optimise to obtain distributions of trajectories, rather than a single trajectory as required in motion planning.

Combined Learning and Optimisation

There has been previous explorations of combining learning and trajectory optimisation. Englert and Toussaint (2018) introduces a framework which combines a known analytical cost function with black box functions in reinforcement learning for manipulation. Rana et al. (2017) introduces CLAMP, a framework that combines learning from demonstration with optimisation based motion planning. Theoretical aspects of using learning in conjunction with optimisation, in a Bayesian framework known as posterior regularisation has also been investigated in Ganchev et al. (2010). All of these methods aim to add more structure to be encoded in learning problems. Similarly our proposed framework allows for encoding of structure, but specifically for the trajectory prediction problem, where we are required to do learning and optimisation on distributions of trajectories with respect to environment occupancy.

7.3 Continuous-time Motion Trajectories

Our framework operates over continuous motion trajectories, capable of being queried at arbitrary resolution, without forward simulating. Here, we briefly revisit the continuous-time motion trajectory representation used in this work, and introduce the relevant notation. We restrict ourselves to the 2-D case.

Similar to methods described in Francis et al. (2019); Marinho et al. (2016); Zhi et al. (2019), we represent motion trajectories as functions $\xi : [0, T] \to \mathbb{R}^2$. T represents the prediction time horizon. In this chapter, we limit our discussion to two-dimensional continuous trajectories. We construct our trajectories as dot products between weight vectors and *reproducing kernels*. We limit our discussion to squared exponential radial basis function (RBF) kernels, though methods in this study generalise to various kernels with minimal modifications. Using RBFs result in smooth trajectories, with good empirical convergence properties for functional trajectory optimisation (Marinho et al., 2016). A trajectory, $\xi(t)$, can be expressed as:

$$\xi(t) = \begin{bmatrix} \mathbf{w}^{\mathbf{x}\top}\boldsymbol{\phi}(t) \\ \mathbf{w}^{\mathbf{y}\top}\boldsymbol{\phi}(t) \end{bmatrix}, \qquad \boldsymbol{\phi}(t) = [\phi(t, t'_1), \ldots, \phi(t, t'_M)]^\top, \tag{7.1}$$

where the basis functions are given as

$$\boldsymbol{\phi}(t, t') = \exp(-\gamma||t - t'||_2^2). \tag{7.2}$$

The coordinates of a dynamic object at time, $t \in [0, T]$, is given as $x(t)$ and $y(t)$, $\mathbf{w}^x$, $\mathbf{w}^y \in \mathbb{R}^M$ are weight parameters, $\boldsymbol{\phi}(t) \in \mathbb{R}^M$, and $\boldsymbol{t'} = [t'_1, t'_2, \ldots, t'_M]^\top$ are M pre-define time points at which RBF kernels are centered. γ is a length-scale hyper-parameter which determines the impact a time point neighbouring times on the trajectory.

Motion data typically comes in the form of discrete sequences of timestamped coordinates $\{t_i, x_i, y_i\}, i = 0, \ldots, N$. To construct a best-fit continuous trajectory of discrete sequences from N timestamped coordinates, we solve the ridge regression problem:

$$\min_{\mathbf{w}^x, \mathbf{w}^y} \sum_{n=1}^{N} \left\{ (x_n - \mathbf{w}^{x^\top} \boldsymbol{\phi}(t_n))^2 + (y_n - \mathbf{w}^{y^\top} \boldsymbol{\phi}(t_n))^2 + \lambda(||\mathbf{w}^x||^2 + ||_2 \mathbf{w}^y||_2^2) \right\}, \tag{7.3}$$

where λ is a regularisation hyper-parameter, with the set of M uniform times $\boldsymbol{t'}$ selected *a priori*, as centers where the kernels are fixed. Evaluating the closed form ridge regression solution results in weight vectors $\mathbf{w}^x$ and $\mathbf{w}^x$ which forms the trajectory $\xi(t) = [\mathbf{w}^{x^\top} \boldsymbol{\phi}(t), \mathbf{w}^{y^\top} \boldsymbol{\phi}(t)]$. We denote the stacked vector of $\mathbf{w}^x$ and $\mathbf{w}^y$ as $\mathbf{w}$. As future motion trajectories are inherently uncertain, the motion prediction problem often requires predicting a distribution over future trajectories a dynamic object may follow. We denote distributions of trajectories as Ξ, with the vector of probabilistic weight parameters, $\boldsymbol{\omega}$, and the parameters of the distribution over $\boldsymbol{\omega}$ is denoted as Ψ. $\Xi(t)$ gives the distribution of coordinate position at time t. Every sample function of Ξ is a trajectory ξ, and samples of vector of random variables $\boldsymbol{\omega}$ give $\mathbf{w}$. Denoting the parameter distributions associated with x and y as $\boldsymbol{\omega}^x$ and $\boldsymbol{\omega}^y$, distributions of trajectories can be constructed as $\Xi = [\boldsymbol{\omega}^{x\top} \boldsymbol{\phi}(t), \boldsymbol{\omega}^{y\top} \boldsymbol{\phi}(t)]$. In practice, to obtain $\boldsymbol{\xi}$ we can also sample $\mathbf{w}$ from the distribution $\boldsymbol{\omega}$, and take dot products with $\boldsymbol{\phi}$.

7.4 Motion Prediction with Environmental Constraints

7.4.1 Problem Overview

Our goal is to reason about the possible future trajectories of dynamic objects, based on the immediate past trajectories of the objects, along with occupancy information from the

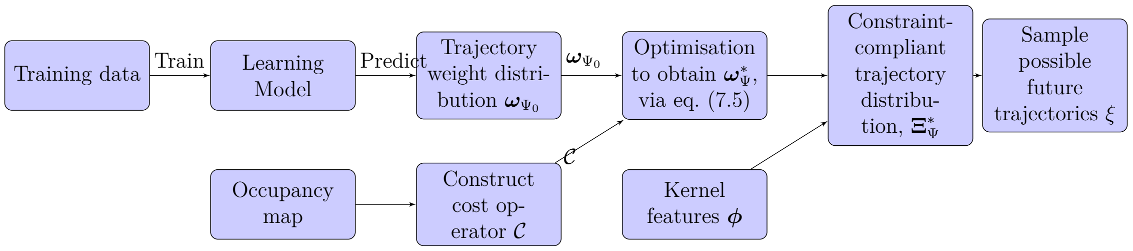

Training data
Train
Learning Model
Predict
Trajectory weight distribution $\boldsymbol{\omega}_{\Psi_0}$
$\boldsymbol{\omega}_{\Psi_0}$
Optimisation to obtain $\boldsymbol{\omega}^*_{\Psi}$, via eq. (7.5)
Constraint-compliant trajectory distribution, Ξ^*_{Ψ}
Sample possible future trajectories ξ
Occupancy map
Construct cost operator $\mathcal{C}$
$\mathcal{C}$
Kernel features $\boldsymbol{\phi}$

Figure 7.2: Overview of our proposed framework. We combine learning and optimisation to generalise data and enforce constraints.

environment. In this work, we limit our discussion to trajectories in 2-D, which usually arise from the tracking of humans and vehicles in video data. We assume to have a dataset containing coordinates of an object up to a time, along with its coordinates thereafter, and an occupancy map of the environment. Examples from a real-world dataset are shown in fig. 7.3, as well as a prediction given by a learning-based model. Given the observed historical coordinates, we aim to extrapolate the distribution of trajectories representing likely future motion thereafter. We require the distribution of trajectories compliant with the environment structure indicated in the occupancy map, such that the probability of hypothesising infeasible trajectories is constrained to a small value.

7.4.2 Framework Overview

An overview of our framework is shown in fig. 7.2. We learn from our training data to predict a vector of random variables of weights $\boldsymbol{\omega}_{\Psi_0}$ with distribution parameters Ψ_0, and produce a prior distribution of trajectories, Ξ_0, as described in eq. (7.1). Our aim is to find an alternative random vector $\boldsymbol{\omega}^*_{\Psi}$, defined by optimised parameters Ψ^*, which satisfies enforced constraints, while remaining close to the learned distribution $\boldsymbol{\omega}_{\Psi_0}$. The optimisation objective uses $\boldsymbol{\omega}_{\Psi_0}$, while constraints are provided through the *obstacle cost operator*, $\mathcal{C}$, which requires an occupancy map of the environment. The desired random vector $\boldsymbol{\omega}^*_{\Psi}$ defines a constraint-compliant distribution of trajectories, Ξ^*_{Ψ}, provided pre-defined RBF features $\boldsymbol{\phi}$, by eq. (7.1). Individual constraint-compliant future trajectories, ξ, can then be realised from Ξ^*_{Ψ}.

To find the constrain-compliant parameters Ψ^*, we formulate the optimisation problem in a similar manner to Bayesian posterior regularisation methods (Ganchev et al., 2010).

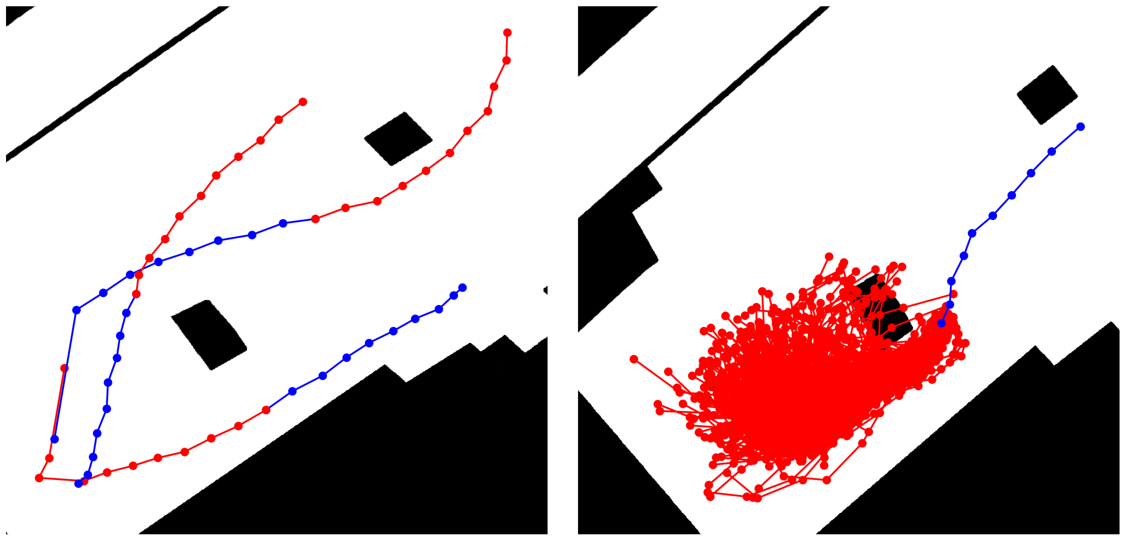


Figure 7.3: (Left) Examples from real-world dataset (Rudenko et al., 2020a). Historical trajectories in blue, and future in red. (Right) Provided a new unseen trajectory (blue), extrapolate futures (red). Learned future trajectories may collide with obstacles. Our framework tackles this by directly enforcing constraints on trajectory distributions, such that the trajectories are collision free.

The desired parameters Ψ^* is the solution to the constrained optimisation problem:

$$\min_{\Psi} \quad \mathrm{D}_{KL}(\boldsymbol{\omega}_{\Psi} \| \boldsymbol{\omega}_{\Psi_0}) \tag{7.4}$$

$$\text{s.t.} \quad \mathcal{C}(\Xi_{\Psi}) - \epsilon \leq 0 \tag{7.5}$$

where D_{KL} is the Kullback-Leibler (KL) divergence (Bishop, 2007), and $\mathrm{D}_{KL}(\boldsymbol{\omega}_{\Psi} \| \boldsymbol{\omega}_{\Psi_0})$ gives us a definition of "close-ness" between our predicted $\boldsymbol{\omega}_{\Psi_0}$ and optimised $\boldsymbol{\omega}_{\Psi}$ distributions. $\mathcal{C}(\Xi_{\Psi})$ is the obstacle cost operator, which maps distributions of trajectories Ξ_{Ψ} to a scalar cost value. The cost is interpreted as the time-averaged probability of generating a trajectory that collides with obstacles. We designate ϵ to be the allowed limit of the obstacle cost. Intuitively, our optimisation problem returns the closest distribution of trajectories, as defined by the KL divergence term, to that given by our predictive model, subject to the obstacle collision constraint. By separating the learning and optimisation, we disentangle the complexities of the learning model and the structural constraints. Our learned $\boldsymbol{\omega}_{\Psi_0}$ also provides us with a "warm-start" initial solution to our optimisation. In contrast to learning-based approaches, the constraints in our optimisation problem are enforced in valid solutions. The main challenges to solving this trajectory optimisation problem lie in how to obtain the vector of random variables $\boldsymbol{\omega}_{\Psi_0}$, which defines our prior trajectory distribution Ξ_0, as well as how to derive the obstacle cost operator $\mathcal{C}$ required for optimisation. These are addressed in the following subsections.

7.4.3 Learning Multi-modal Distributions of Trajectories

In this subsection, we outline a method of learning a mapping from observed historical trajectories to prior distributions of future trajectories, parameterised by $\boldsymbol{\omega}_{\Psi_0}$. The resulting prior distributions of trajectories are to be used directly in the optimisation problem eq. (7.5). We model the distribution of trajectories, Ξ, as mixtures of multi-output stochastic processes. To capture the correlation between x and y coordinates, the distribution over weight parameters are assumed to be mixtures of matrix normal distributions (Dawid, 1981). A matrix normal distribution is the generalisation of the normal distribution to matrix-valued random variables, where each sample from a matrix normal distribution gives a matrix. We work with the matrix of random variables defined by vertically stacking the parameters of coordinate dimensions, denoted as $\mathbf{W} = [\boldsymbol{\omega}^x, \boldsymbol{\omega}^y] \in \mathbb{R}^{M\times 2}$:

$$\mathbf{W} = \begin{bmatrix} w_1^x & w_1^y \\ \vdots & \vdots \\ w_M^x & w_M^y \end{bmatrix}, \qquad p(\mathbf{W}) = \sum_{r=1}^{R} \alpha_r \mathrm{MN}\Big(\mathbf{M}_r, \mathbf{U}_r, \mathbf{V}_r\Big). \tag{7.6}$$

We assume that there are R mixture components, similar to a Gaussian mixture model (Bishop, 2007), with each component being a matrix normal distribution, and associated weight components α_r. Matrix normal distributions have location matrices $\mathbf{M}_r = [\boldsymbol{\mu}_r^x, \boldsymbol{\mu}_r^y] \in \mathbb{R}^{M\times 2}$, and positive-definite scale matrices $\mathbf{U}_r \in \mathbb{R}^{M\times M}$, $\mathbf{V}_r \in \mathbb{R}^{2\times 2}$ as parameters (Dutilleul, 1999).

We utilise a neural network to learn a mapping from a fixed-length vectorised coordinate sequences, $\boldsymbol{\varphi}$, that encodes observed partial trajectories, to component weights $\boldsymbol{\alpha}$, along with location matrices $\mathbf{M}_r$ the lower triangular $\mathbf{V}_r^{\frac{1}{2}}$ and diagonal $\mathbf{U}_r^{\frac{1}{2}}$ matrices. $\mathbf{V}$ is positive-definite and will be constructed by taking the product of two lower triangular matrices. The dependence between different times on the trajectory are largely captured the RBF features in the time domain $\boldsymbol{\phi}(t)$, as described in section 7.3. Hence, it is sufficient for $\mathbf{U}$ to be a positive definite diagonal matrix. We obtain the input vector $\boldsymbol{\varphi}$ by taking a 10 time-step window of the latest observed coordinates, and vectorising to obtain a vector of 20 dimensions. The neural network used is relatively light-weight, consisting of 4 Relu-activated dense layers with number of neurons: $(15{\times}M{\times}R) \rightarrow (5{\times}M{\times}R) \rightarrow (5{\times}M{\times}R) \rightarrow (R{\times}[3M{+}3])$. In this work, our main goal is to outline a representation of future trajectories outputted by the network, such that the result can be directly optimised. Our method does not preclude alternative encoding schemes of conditioned history trajectories to be inputted into the network, such as features from LSTM networks (Hochreiter and Schmidhuber, 1997) or trajectory features outlined in (Zhi et al., 2019).

We are provided with a dataset of history and future trajectories, given by N pairs of timestamped coordinate sequences. After flattening the history trajectory sequence to obtain $\boldsymbol{\varphi}$, and applying eq. (7.3) on trajectory futures to obtain weight matrices, we have pairs denoted as $\{\boldsymbol{\varphi}_n, \mathbf{W}_n\}_{n=1}^{N}$. We learn a mapping from $\boldsymbol{\varphi}_n$ to parameters $\{\alpha_r, \mathbf{M}_r, \mathbf{V}_r, \mathbf{U}_r\}_{r=1}^{R}$ with the fully-connected neural network, with the negative log-likelihood of the weight matrices distributions as the loss function:

$$\mathcal{L} = -\sum_{n=1}^{N} \log \sum_{r=1}^{R} \alpha_r \frac{\exp\{-\frac{1}{2}\mathrm{tr}[\mathbf{V}_r{}^{-1}(\mathbf{W}_n - \mathbf{M}_r)^\top \mathbf{U}_r(\mathbf{W}_n - \mathbf{M}_r)]\}}{(2\pi)^M |\mathbf{V}_r||\mathbf{U}_r|^{\frac{M}{2}}}. \tag{7.7}$$

and by applying the activation functions:

$$\mathrm{diag}[(\mathbf{U}_r)^{\frac{1}{2}}] = \exp(z^{\mathrm{diag}(\mathbf{U}_r)}), \tag{7.8}$$

$$\mathrm{LowerTrig}[(\mathbf{V}_r)^{\frac{1}{2}}] = z^{\mathrm{LowerTrig}(\mathbf{V}_r)}, \tag{7.9}$$

$$\mathrm{diag}[(\mathbf{V}_r)^{\frac{1}{2}}] = \exp(z^{\mathrm{diag}(\mathbf{V}_r)}), \tag{7.10}$$

$$\alpha_r = \mathrm{softmax}(z^{\alpha_r}), \tag{7.11}$$

where $z^{\alpha_r}, z^{\mathrm{LowerTrig}(\mathbf{V}_r)}, z^{\mathrm{diag}(\mathbf{V}_r)}, z^{\mathrm{diag}(\mathbf{U}_r)}$ are neural network outputs, and $\mathrm{diag}(\cdot)$, $\mathrm{LowerTrig}(\cdot)$ indicate the matrix diagonal, and strictly lower triangular matrix elements respectively. The exp activation function guarantees the validity of the scale matrices, by enforcing $z^{\mathrm{diag}(\mathbf{V}_r)} > 0, z^{\mathrm{diag}(\mathbf{U}_r)} > 0$. We also enforce $\sum_{r=1}^{R} \alpha_r = 1$ using the softmax function. After we train the neural network, provided a vectorised historical trajectory, $\boldsymbol{\varphi}$, the neural network gives the corresponding random vector of weights of our prior trajectory distribution, $\boldsymbol{\omega}_{\Psi_0} = \mathrm{vec}(\mathbf{W})$, where $\mathrm{vec}(\cdot)$ is the vectorise operator. The prior distributions of trajectories is then the dot product of the trajectory parameters and the RBF features, $\boldsymbol{\phi}(t)$, as described in eq. (7.1).

7.4.4 Constraints on Distributions of Trajectories

After obtaining the weights, $\boldsymbol{\omega}_{\Psi_0}$, of a prior distribution of trajectories via learning, our main challenge is to design a cost operator, $\mathcal{C}(\Xi)$, to encode our desired constraints. Specifically we wish to design $\mathcal{C}(\Xi)$ to account for collisions of distributions of trajectories, provided an occupancy map. $\mathcal{C}(\Xi)$ maps a distribution of smooth trajectories, Ξ, to a real-valued scalar cost, which quantifies how collision-prone a given Ξ is. $\mathcal{C}$ differs from the cost operator in trajectory optimisation motion planning problems, such as those in Zucker et al. (2013); Marinho et al. (2016); Francis et al. (2019), in that the input is a distribution over trajectories rather than a single trajectory. Our optimisation can not only optimise over trajectory mean, but also over trajectory variances.

Like trajectory optimisation methods (Zucker et al., 2013), we assume the cost operator takes the form of average costs over prediction horizon T, where the cost at a given point in time, $\mathcal{C}(\Xi_\Psi(t))$, is the probability of a collision at a given time. Ξ_Ψ contains weight variables with a distribution parameterised by Ψ. We can then interpret $\mathcal{C}(\Xi_\Psi)$ as the average probability of collision over a time horizon, defined as:

$$\mathcal{C}(\Xi_\Psi) = \frac{1}{T}\int_0^T \mathcal{C}(\Xi_\Psi(t))\mathrm{d}t \tag{7.12}$$

$$= \frac{1}{T}\int_0^T \int_{\mathbb{R}^2} p(m=1|\mathbf{x})p(\mathbf{x}|\Psi,t)\mathrm{d}\mathbf{x}\mathrm{d}t. \tag{7.13}$$

The probability of collision at some time and location is computed by taking the product of the probability of the trajectory reaching the coordinate at the given time and the probability that the coordinate is occupied. We are provided with an occupancy map and the probability a coordinate of interest, $\mathbf{x} \in \mathbb{R}^2$, being occupied is denoted as $p(m = 1|\mathbf{x})$. To evaluate the cost, we could marginalise over the all trajectory parameters, but this may be difficult due to the high dimensionality of $\boldsymbol{\omega}$. We instead work in the 2D world space, where the environmental constraints are defined. As Ξ_Ψ is constructed by taking the dot product of weight vectors and feature vectors, trajectory coordinate distribution in world space at time

t is:

$$p(\Xi_\Psi(t)) = \sum_{r=1}^{R} \alpha_r \overbrace{\mathrm{MN}(\mathbf{M}_r^\top \boldsymbol{\phi}(t), \mathbf{V}_r, \boldsymbol{\phi}(t)^\top \mathbf{U}_r \boldsymbol{\phi}(t))}^{\text{matrix normal distribution}} \tag{7.14}$$

$$= \sum_{r=1}^{R} \alpha_r \overbrace{\mathcal{N}(\underbrace{\mathbf{M}_r^\top \boldsymbol{\phi}(t)}_{=:\boldsymbol{\mu}_r}, \underbrace{\boldsymbol{\phi}(t)^\top \mathbf{U}_r \boldsymbol{\phi}(t) \mathbf{V}_r}_{=:\Sigma_r})}^{\text{multivariate normal}}. \tag{7.15}$$

By the equivalences between matrix normal and multivariate normal distributions (Dawid, 1981), we have a mixture of two-dimensional multivariate normal distributions in world-space, $\Xi_\Psi(t) \sim \sum_{r=1}^{R} \alpha_r \mathcal{N}(\boldsymbol{\mu}_r, \Sigma_r)$. This is the probability distribution over trajectory co-ordinates at time t, $p(\mathbf{x}|\Psi, t)$. The mixture means and covariances of each component are defined as

$$\boldsymbol{\mu}_r := \mathbf{M}_r^\top \boldsymbol{\phi}(t), \qquad \Sigma_r := \boldsymbol{\phi}(t)^\top \mathbf{U}_r \boldsymbol{\phi}(t) \mathbf{V}_r. \tag{7.16}$$

Substituting eq. (7.15) into the inner integral of eq. (7.13), gives the cost operator equation at given time:

$$\mathcal{C}(\Xi_\Psi(t)) = \sum_{r=1}^{R} \alpha_r \int_{\mathbb{R}^2} \left\{ \frac{\exp[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_r)^\top \Sigma_r^{-1} (\mathbf{x} - \boldsymbol{\mu}_r)]}{(2\pi)|\Sigma_r|^{\frac{1}{2}}} \right\} \underbrace{p(m = 1|\mathbf{x})}_{\text{From map}} \, \mathrm{d}\mathbf{x} \tag{7.17}$$

Note that $\boldsymbol{\mu}_r$ and Σ_r are dependent on $\mathbf{M}_r, \mathbf{U}_r, \mathbf{V}_r$. During the optimisation, we assume that each mode can be optimised independently and that the importance of each mode, represented by α, is accurately captured by the learning model and does not change during trajectory optimisation. We then define the set of parameters to optimise as $\Psi = \{\mathbf{M}_r, \mathbf{U}_r, \mathbf{V}_r\}_{r=1}^{R}$. The integral defined in eq. (7.17) may be intractable to evaluate analytically, so we aim to find an approximation to it. The coordinate distribution at a given time in world space, $\Xi_\Psi(t)$, is a mixture of bi-variate Gaussian distributions. We use Gauss-Hermite quadrature, suitable for numerically integrating exponential-class functions, to approximate eq. (7.17). Quadrature methods approximate an integral by taking the weighted sum of integrand values sampled at computed points, called abscissae. We introduce the change of variables,

$$\boldsymbol{z}_r = \frac{1}{\sqrt{2}} L_r^{-1} (\mathbf{x} - \boldsymbol{\mu}_r), \tag{7.18}$$

where the Cholesky factorisation gives $\Sigma_r = L_r L_r^T$. Using Gauss-Hermite quadrature schemes, we have:

$$\mathcal{C}(\Xi_\Psi(t)) \approx \sum_{r=1}^{R} \alpha_r \pi^{-1} \sum_{i=1}^{I} \sum_{j=1}^{J} \beta^i \beta^j p(m=1|\sqrt{2}L\boldsymbol{z} + \boldsymbol{\mu}_r), \tag{7.19}$$

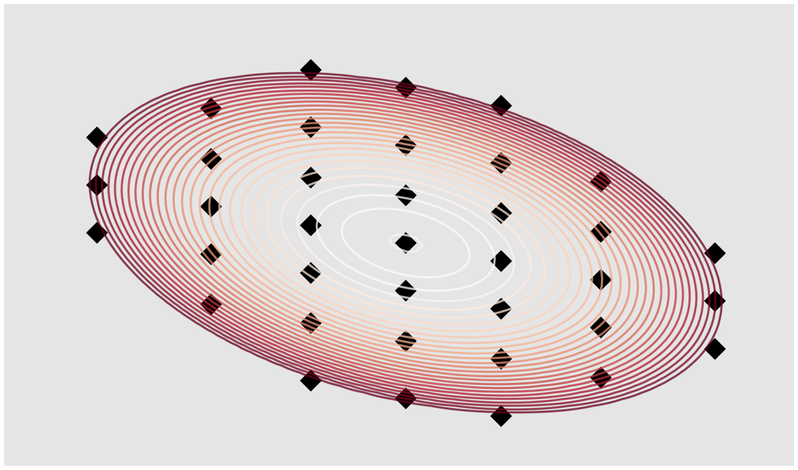
where we find the values of $\boldsymbol{z}^{i,j} = [z^i, z^j]^\top$ as abscissa obtained from roots of Hermite polynomials, and β^i, β^j are the associated quadrature weights. Details of abscissae and weight calculations, as well as a review on multi-dimensional Gauss-Hermite quadrature, can be found in (Liu and Pierce, 1994). An example of abscissae points when estimating a multi-variate Gaussian is shown in fig. 7.4. Next, we evaluate the outer integral in eq. (7.13) using rectangular numerical integration for

$$\mathcal{C}(\Xi_\Psi) = \int_0^T \mathcal{C}(\Xi_\Psi(t))\mathrm{d}t. \tag{7.20}$$

With $\mathcal{C}(\Xi_\Psi)$ defined, we are in a position to solve the combined learning and constrained optimisation problem, defined earlier in eq. (7.5), using a non-linear optimisation solver, such as the sequential least squares optimiser SLSQP (Kraft, 1988). Analytical gradients with respect to occupancy is provided when using a continuous differential occupancy map such as those introduced in Ramos and Ott (2016); Zhi et al. (2019a). By utilising the equivalence between matrix normal distributions and multivariate Gaussians (Dutilleul, 1999), the KL divergence term between $\boldsymbol{\omega}_\Psi$ and $\boldsymbol{\omega}_{\Psi_0}$ components can be computed (Bishop, 2007). After obtaining the solution of the optimisation we have the optimised distribution of $\boldsymbol{\omega}^*_\Psi$. By taking dot products between $\boldsymbol{\omega}^*_\Psi$ and $\boldsymbol{\phi}$, we can construct Ξ^*_Ψ the constraint-compliant trajectory distribution of futures. Individual realised trajectories ξ can then be generated from Ξ_Ψ.

Although we have focused on collision constraints with respect to environment structure, the optimisation allows for constraints on velocity or acceleration. The continuous nature of our trajectories allows time derivatives of trajectories to be obtained. Specifically, the *derivative reproducing property*, ensures the derivatives of smooth kernels also correspond to kernels (Zhou, 2008). We can reason about the n^{th}-order derivatives of displacement by replacing $\boldsymbol{\phi}(t)$ with $\frac{\partial^n \boldsymbol{\phi}(t)}{\partial t^n}$.

Figure 7.4: The abscissae (black) for estimating the distribution in world space with the Gauss-Hermite quadrature scheme



7.5 Experimental Evaluation

We empirically evaluate (1) the ability of our learning component to learn probabilistic representations of future trajectories to provide good prior trajectories distributions; (2) the ability of the proposed framework to enforce collision constraints, and its effect on trajectory quality.

7.5.1 Datasets and Metrics

We run our experiments on one simulated dataset, and two real-world datasets, including:

- Simulated dataset **(Simulated)**: This contains simulated trajectories on a floor-plan;
- THÖR dataset (Rudenko et al., 2020a) **(Indoor)**: This dataset contains real human trajectories walking in a room with three large obstacles in the center of the room.
- Lankershim dataset **(Traffic)**: This dataset contains real vehicle data along a long boulevard. We use a particular busy intersection between $x = [-80, 80]$ and $y = [250, 500]$.

Throughout each of our experiments, we ensure that the observed trajectory conditioned on contains 10 timesteps. The prediction horizons, T, for the simulated, indoor and traffic datasets are 15, 10, and 20 timesteps respectively. We split long trajectories, such that we have 200 pairs of partial trajectories in the simulated dataset, while the indoor dataset contained 871 pairs of partial trajectories, and the traffic dataset contained 6175 pairs. Trajectories that were shorter than the length of the prediction horizon were filtered out. A train-to-test ratio of 80:20 was used for each dataset. For our learning model, we set the number of mixture components $R = 2$, hyper-parameters $M = 8, 10, 11$, and $\gamma = 0.05, 0.1, 0.2$ for the simulated, indoors, and traffic datasets respectively. Maps for the simulated and indoor datasets are available, and we construct a map for the traffic dataset by considering the road positions. Our occupancy map is represented as a continuous Hilbert Map (Ramos and Ott, 2016), giving smooth occupancy probabilities over coordinates, and occupancy gradients.

Metrics used in the quantitative evaluation include:

- Average displacement error (ADE): The average euclidean distance error between the expectation of the nearest distribution component of trajectories with ground truth trajectory;

- Final displacement error (FDE): The euclidean distance error at the end of the time horizon between the expectation of the nearest distribution component of trajectories with ground truth trajectory.

- Average Likelihood (AL): To take into account the uncertain nature of motion prediction, for probabilistic models, we record the likelihood of drawing the trajectory averaged over the timesteps.

Note that likelihoods are difficult to interpret and not comparable over different datasets. Lower ADE and FDE, and higher AL indicate better performance. For the evaluation of collisions, we also record the percentage of constraint-violating trajectory distributions in the test set.

7.5.2 Learning Continuous-Time Trajectory Distributions as Priors

The trajectory distribution learning component of our framework provides a prior distribution for optimisation. We examine whether the learning component of our framework is able

		ADE	FDE	AL
Simulated	Ours	1.29	2.27	0.12
	KTM (Zhi et al., 2019)	1.44	2.41	0.11
	CV	4.27	11.66	-
	NN-naive	2.11	6.07	-
Indoors	Ours	0.94	1.32	3.32
	KTM	0.65	1.26	2.88
	CV	2.08	4.66	-
	NN-naive	1.84	3.35	-
Traffic	Ours	4.99	7.24	0.04
	KTM	4.98	7.03	0.043
	CV	5.14	10.38	-
	NN-naive	21.1	39.5	-

Table 7.1: We evaluate the quality of our learned trajectory prior with benchmark models. Lower ADE and FDE, and higher AL indicate better performance.

to learn distributions of future trajectories from data, comparing the performance with several benchmark models. The benchmark models are (1) Kernel trajectory maps (KTMs) (Zhi et al., 2019), (2) Constant velocity (CV) model, and (3) Naive neural-network (NN) model, where we learn a mapping between past trajectories and future trajectories by minimising the mean squared error loss. The neural network comprised of Relu-activated fully-connect layers with the number of neurons: $(560) \rightarrow (180) \rightarrow (180) \rightarrow (N_t)$, where N_t gives the trajectory time horizon.

We see that the learning component of our framework is able to learn high-quality distributions of trajectories. Figure 7.5 shows examples of extrapolated trajectory distributions in red, and the observed trajectory in blue. We see that the learned distribution is relatively close to the ground truth green trajectory, and is able to capture the uncertain nature of motion prediction. Although a non-negligible number of sampled trajectories collide with the environment. Table 7.1 summarises the performance results of our method against our benchmarks. We see that our learning model is comparable to the method proposed in Zhi et al. (2019), while outperforming a constant velocity and naive neural network model. While the KTM method performs slightly better on the traffic dataset, it is purely learning-based and lacks mechanisms to enforce constraints. We note that our method encodes the observed trajectory by simply flattening the input coordinates, as described in section 7.4.3.

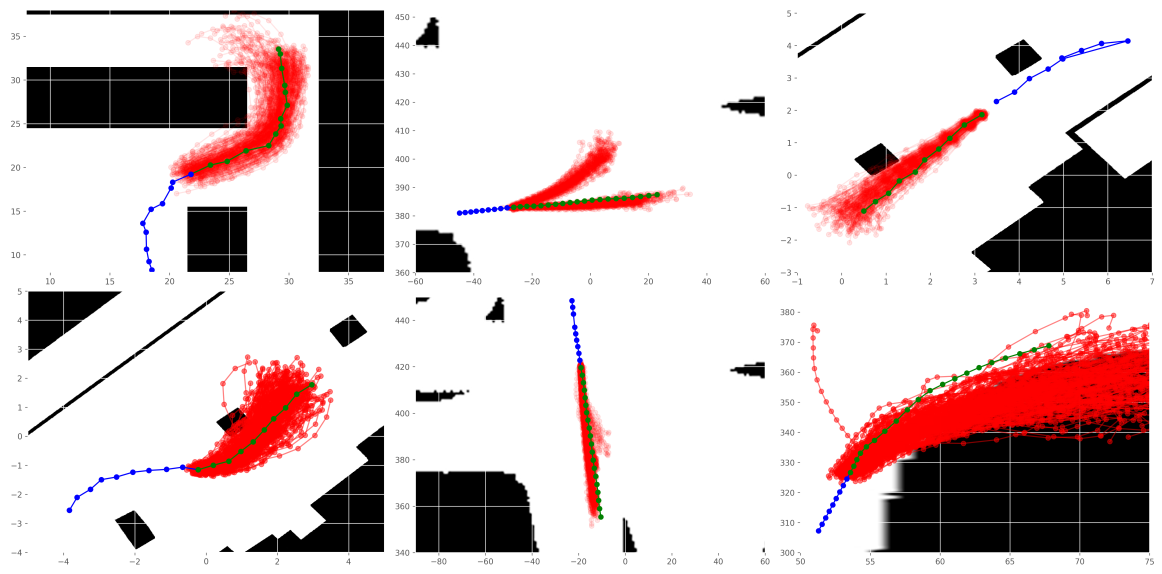


Figure 7.5: Quantitative evaluations of learned distributions of trajectories (red), Conditioned observations (blue), ground truth (green). We see that our learning model gives relatively good priors, but many sampled trajectories collide with obstacles. The prior distributions are further enforced to be collision-free (See fig. 7.7)

More sophisticated methods of encoding the observed trajectory (such as with LSTM networks (Hochreiter and Schmidhuber, 1997)), independent of our outlined output of trajectory distributions, could improve the performance of the learning model.

7.5.3 Enforcing Structure Compliance

After obtaining a prior from our learned model, we apply constraints on the prediction via trajectory optimisation. We will evaluate whether collision constraints can be enforced, and their effect on trajectory quality. We define the constraint such that the time-averaged probability of collision is less than or equal to 0.05, $\mathcal{C} - 0.05 \leq 0$. The SLSQP optimiser was able to solve the optimisation quickly, under half a second, with a python implementation. We obtain trajectory distribution priors from our learned model, and select all of the observations where the learned prior trajectory is constraint-violating. We evaluate the performance of the trajectory priors, and compare them with the trajectory distribution predictions after enforcing constraints. The quantitative results are listed in table 7.2. We see after solving with structural constraints, all predictions are feasible. Additionally, the trajectory distribution quality improves after enforcing constraints for all the datasets considered.

		ADE	FDE	AL	C.V.P
Simulated	Unoptimised	1.48	2.87	0.11	40%
	Optimised	1.28	2.33	0.17	0%
Indoors	Unoptimised	1.62	2.41	3.42	15.4%
	Optimised	1.44	2.38	3.3	0%
Traffic	Unoptimised	6.56	11.81	0.034	5.7%
	Optimised	5.14	8.95	0.043	0%

Table 7.2: We evaluate the quality of our optimised trajectory distribution with collision constraints, relative to the prior. We see that providing constraints improves prediction quality across all dataset. The percentage of trajectory distributions violating constraint, $\mathcal{C} - 0.05 \leq 0$, relative to the entire test set is also given, as Constraint Violation Percentage (C.V.P). After optimisation all trajectory distributions are constraint-compliant.

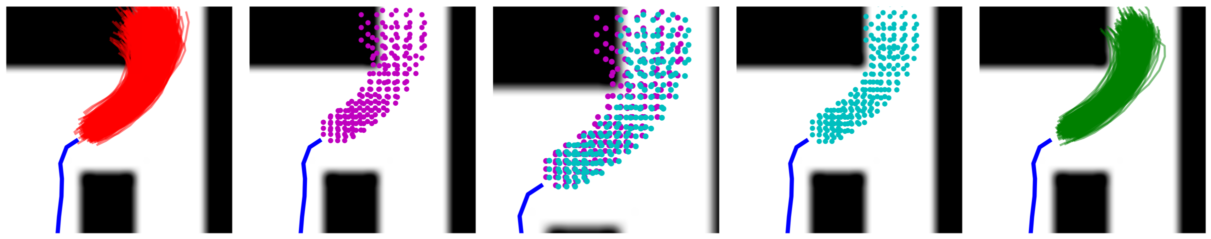


Figure 7.6: An example of an originally non-compliant prior distribution of trajectories becomes more constraint compliant after the constraint enforcement. Left to right: (1) Non-compliant distribution of trajectories; (2) Abscissae of constraint violating distribution; (3) Abscissae of before and after constraint enforcement (zoomed in); (4) Abscissae of constraint compliant distribution; (5) Compliant distribution of trajectories.

Qualitative results are shown in fig. 7.7 and fig. 7.6. We see the abscissae points of the largest trajectory distribution component before (top) and after (bottom) the trajectory optimisation of examples in each dataset. Our framework is able to ensure that the distributions comply with environmental structure. Notably, we see that the trajectory optimisation not only adjusts the mean of the trajectory distribution, but also optimises the variances, giving more robust trajectory distributions. This effect is most prominent in the left sub-figure, where the predicted distribution of trajectories in a simulated hallway recovers a much tighter variance which follows the environment structure. We also observe instances of collision-avoidance with objects in open space (demonstrated in the middle sub-figure), as well as increased adherence to road structure (right sub-figure).

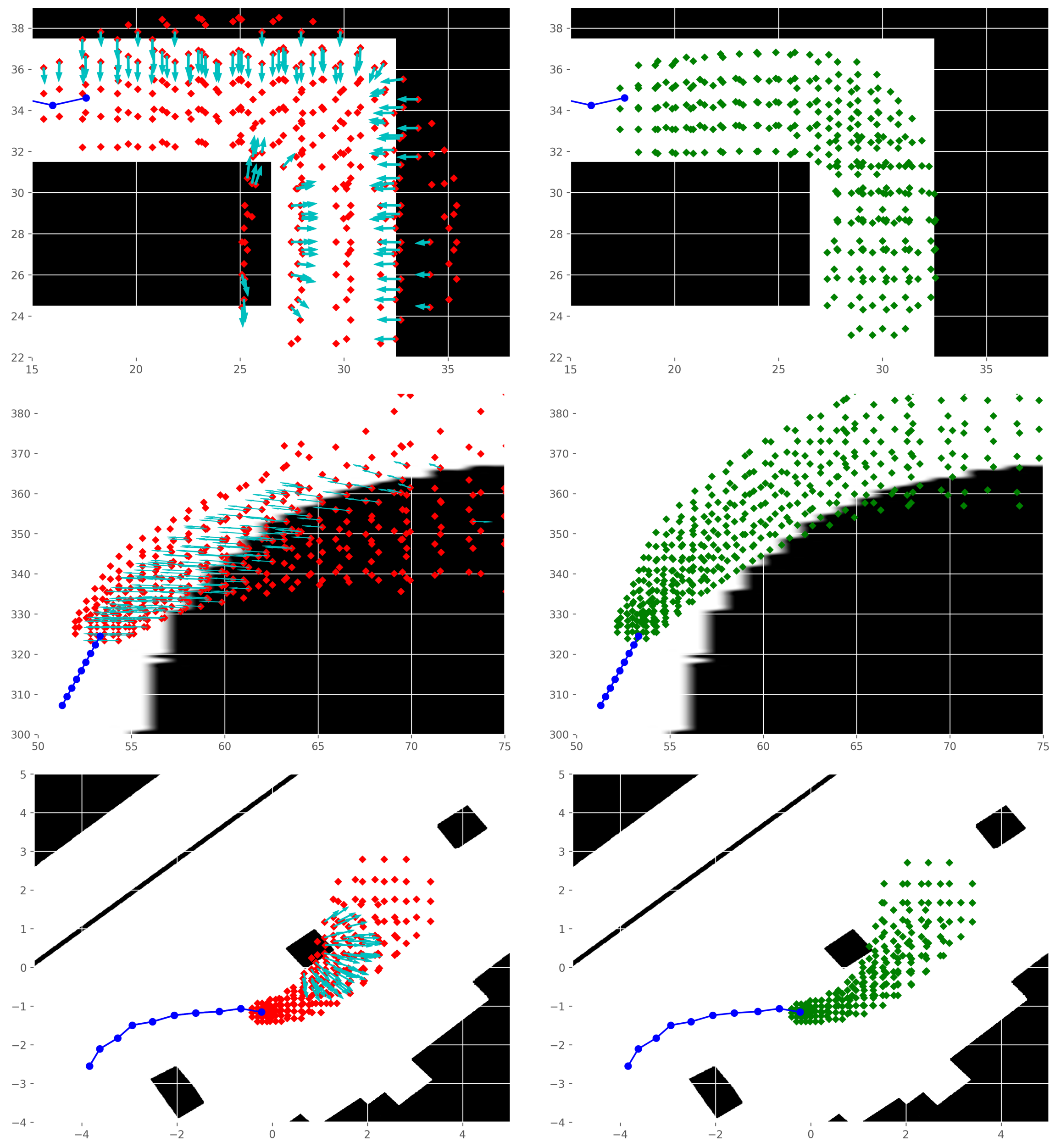


Figure 7.7: Examples from each dataset with the optimisation to apply constraints. The left column shows abscissae points of the main distribution component (component with the largest weight), with notable gradients with respect to occupancy shown as arrows (cyan) at the abscissae. The right column shows the same abscissae points after optimisation. We see that the constrained distribution subtly conforms better to structures in the environment, while still closely resembling the learned trajectory distribution. The left, middle and right sub-figures are examples from the simulated, indoors, and traffic datasets respectively.

7.6 Summary

We introduced a novel framework to learn motion predictions from observations while conforming to constraints, such as obstacles. Our framework consists of a learning component which learns a distribution over future trajectories. This distribution is used as a prior in a trajectory optimisation step which enforces chance constraints on the trajectory distribution. We empirically demonstrate that our framework can learn complex trajectory distributions and enforce compliance with environment structures via optimisation. This results in reduced variance and the avoidance of obstacles by the predicted trajectories, leading to improved prediction quality.

We shall now move to part III, and explore the usage of learning in the generation of motion trajectories for manipulator arm robots. Commercially-available robot manipulators have much higher degrees-of-freedom relative to wheeled ground robots and present a different set of challenges to researchers. In the upcoming chapter 8, we shall contribute a framework which allows robots to learn from expert demonstrations, and generalise the knowledge when the surroundings change.

Part III

Learning for Robot Manipulator Motion Generation

Chapter 8

Diffeomorphic Transforms for Generalised Imitation Learning

8.1 Introduction

Imitation learning, or learning by demonstration, is an approach where robots are taught to execute novel motions from a few human expert demonstrations. Such an approach is natural and intuitive for generating complex motions, as it circumvents hand coding movements or specifically designing control costs (Osa et al., 2018). Crucial to imitation learning is a generative model mimicking the provided demonstrations, as well as methods to reproduce motions in novel situations. Generative models for imitation learning can typically be described as having a time-dependent or state-dependent representation. In this chapter, we focus on the state-dependent category, where motions are generated by integrating a time-invariant non-linear dynamical system.

A major challenge of representing robot motion with a state-dependent dynamical system is the need to ensure the system's stability. Various methods (Khansari-Zadeh and Billard, 2011; Saveriano, 2020; Rana et al., 2020; Urain et al., 2020) have been developed to learn stable dynamics. In particular, recent methods such as Rana et al. (2020); Urain et al. (2020) attempt to construct inherently stable dynamical systems by learning a smooth bijective

This chapter has been published in L4DC as Zhi et al. (2022b).

mapping (i.e. a diffeomorphism) linking a known stable system and a desired target system, using invertible neural networks. Although these methods are capable of learning stable systems to accurately reproduce demonstrated examples, it is difficult to gauge how the neural network deforms the known stable system. The opaqueness of the network offers little insight into how to inject additional knowledge to alter the system upon encountering a change in the environment.

Furthermore, prior work often focuses solely on learning a stable representation to imitate demonstrations, and struggles to operate in different environments where no new demonstrations are provided. A specific example is described in fig. 8.1, where additional obstacles cause the previous demonstrations to be in collision. We contribute the diffeomorphic transforms framework for generalised imitation learning. We place emphasis on the ability of the system to generalise when surroundings change, by factoring in other sources of information, such as occupancy data or additional user specification. We represent each behaviour, such as reproducing demonstrations and obstacle avoidance, as individual diffeomorphic transforms. Each transform can be built in various ways, such as learning from demonstrated examples, learning from occupancy data, or be hand-crafted. By combining diffeomorphic transforms, generated motions can not only be learned from demonstrations, but also be adjusted based on changes in the environment, such as avoiding new obstacles or warping towards a designated point, all while conserving the stability of the resulting system.

8.2 Related Work

Imitation learning is an approach that replicates motion based on expert demonstrations, allowing human operators to transfer skills to robot systems. One class of methods in imitation learning represent robot motions as time-dependent functions, these include methods proposed in Paraschos et al. (2013); Huang et al. (2019); Kulak et al. (2020). Another class of methods represents robot motion by a stable state-dependent dynamical system. Such a representation is more robust to perturbations in the starting position, particular in regions with no data, as the system is guaranteed to stabilise at some point. A pioneer example of a stable state-based representation is SEDS (Khansari-Zadeh and Billard, 2011), which selects a candidate from a family of quadratic Lyapunov functions and is used to constrain the learned system via sequential quadratic programming. However, the expressiveness of SEDs is restricted by being limited to candidates from the quadratic Lyapunov functions

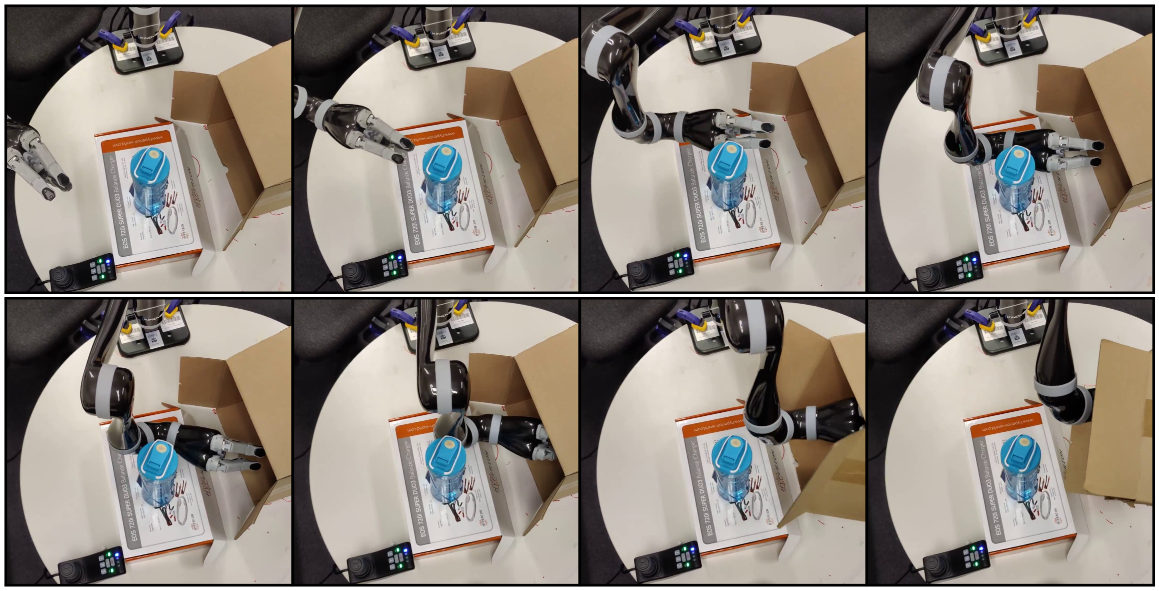


Figure 8.1: A motivating application for our work is generalising stable systems learned from expert demonstrations, when surroundings change. Here, a 6 degrees-of-freedom JACO arm is shown 8 recorded human demonstrations. Then, an obstacle bottle is placed right in the path of the recorded demonstrations. The JACO arm is able to warp smoothly around obstacle and successfully reach in and lift the box.

family. Further work in Khansari-Zadeh and Billard (2014) expands the family of Lyapunov functions for candidates.

Recent approaches on learning stable systems often take a different strategy. Instead of explicitly parameterising and learning the class of stable systems, these approaches learn smooth bijective mappings, diffeomorphisms, which map between a stable base system and some target. The resulting dynamical system can be thought of as a "morphing" of the base system, and preserves the stability properties of the original system. This approach could be thought of as analogous to normalising flows (Rezende and Mohamed, 2015) for density estimation, which utilises invertible mappings to map a known probability distribution to the target distribution. The methods outlined in Neumann and Steil (2015); Urain et al. (2020); Rana et al. (2020) take this approach. Particularly, Rana et al. (2020) interprets the target system as a stable natural gradient system (Amari, 1998) on a manifold with an associated metric provided by the diffeomorphism. Our work takes a similar approach, by using diffeomorphisms to transform a stable base system, we can ensure the resulting system is stable in a principled manner.

Despite the advancements made in imitation learning, there have been relatively few works addressing adapting imitations to new environments. Calinon (2016) propose to adjust parameters of Gaussian mixture models on based tasks, while the authors of Rana et al. (2017) make attempts to alter generated trajectories, by post-processing with motion planning. However, these method are restricted to time-dependent trajectories, rather than state-dependent systems. In particular, the method in Rana et al. (2017) is restricted to discrete-time trajectories. Our approach is able to learn stable dynamical systems, and generalise the learned stable systems based on environment occupancy and user specifications.

8.3 Preliminaries

8.3.1 Motion Representation

Robot motion is represented as a mapping between the robot state $\boldsymbol{x} \in \mathbb{R}^d$ and its velocity $\dot{\boldsymbol{x}} \in \mathbb{R}^d$. We model the motion as a time-invariant non-linear first-order dynamical system:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t)), \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{8.1}$$

where $f : \mathbb{R}^d \to \mathbb{R}^d$ is a non-linear mapping, and $\boldsymbol{x}_0$ is the initial state. A trajectory of the system is given by the solution of eq. (8.1), i.e. $\xi(t, \boldsymbol{x}_0) = \boldsymbol{x}_0 + \int_0^t \dot{\boldsymbol{x}}(s)\mathrm{d}s$. Additionally, like many other works in this area (Neumann and Steil, 2015; Urain et al., 2020; Rana et al., 2020), we require the motion to be *globally asymptotically stable*, as outlined in section 2.6.2. Stable systems behave in an expected manner during robot execution, and have vastly improved generalisation capabilities, as demonstrated in Sindhwani et al. (2018).

8.3.2 Imitating demonstrations

Provided a set of N demonstrations, Ξ, where the i^{th} demonstration is given by a sequence of states, of length l_i, i.e. $\Xi = \{\{\boldsymbol{x}_{i,1}, \boldsymbol{x}_{i,2}, \ldots, \boldsymbol{x}_{i,l_i}\}\}_{i=1}^{N}$. The velocities, $\dot{\boldsymbol{x}}_{i,j}$, are assumed to be available. To imitate the demonstrations, we aim to learn an $f(\boldsymbol{x})$ such that trajectories generated using eq. (8.1) match the empirical demonstrations Ξ. Assuming that $\boldsymbol{\theta}$

parameterises f, we can optimise $\boldsymbol{\theta}$ in a least squares manner:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \sum_{j=1}^{l_i} ||\dot{\boldsymbol{x}}_{i,j} - f_{\boldsymbol{\theta}}(\boldsymbol{x}_{i,j})||_2^2, \qquad \text{s.t. } \dot{\boldsymbol{x}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}) \text{ is stable.} \tag{8.2}$$

8.3.3 Generalising the Learned System

After optimising for parameters $\boldsymbol{\theta}$, we can generate trajectories from $f_{\boldsymbol{\theta}}$ which imitate the provided demonstrations. Beyond simply mimicking the provided demonstrations, we are also interested in modifying $f_{\boldsymbol{\theta}}$ to satisfy further requirements, particularly when the situation changes and no further demonstrations are provided. A motivating example is when the position of objects in the environment changes, resulting in the original demonstrations no longer being collision-free. With no access to new demonstrations but only the environment's occupancy, many open questions exist: How do we make use of occupancy data to update the system such that generated trajectories avoid collision and imitate demonstrations while enforcing the stability of the system? Furthermore, how should we modify the system such that generated trajectories incorporate more general additional user specifications, such as a bias towards a certain direction?

8.3.4 Preserving Stability with Diffeomorphisms

An elegant method, as presented in Rana et al. (2020), of ensuring the stability of $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$, is by viewing the target system as a *natural gradient system* defined on some d-dimensional manifold $\boldsymbol{M}$, where integral curves of the target system on $\boldsymbol{M}$ correspond to integral curves of another known simple system in Euclidean space. A bijective mapping is a *diffeomorphism* if ψ and ψ^{-1} are continuously differentiable. A diffeomorphism $\psi : \boldsymbol{M} \to \mathbb{R}^d$ maps points on the manifold to Euclidean space.

Given a differentiable potential function, $\Phi : \mathbb{R}^d \to \mathbb{R}$, and diffeomorphism ψ, and following Rana et al. (2020), we can define the *natural gradient descent* system (Amari, 1998) on $\boldsymbol{M}$, corresponding to the system $\dot{\boldsymbol{z}} = \nabla_{\boldsymbol{z}}(\boldsymbol{z})$ defined in Euclidean space as:

$$\dot{\boldsymbol{x}} := f(\boldsymbol{x}) = -\boldsymbol{G}(\boldsymbol{x})^{-1} \nabla_{\boldsymbol{x}} \Phi(\psi(\boldsymbol{x})), \tag{8.3}$$

where $\boldsymbol{G} : \boldsymbol{M} \to \mathbb{R}_{++}^{d \times d}$ is a real positive definite matrix *Riemannian metric* which varies smoothly on $\boldsymbol{M}$. We take the metric as

$$\mathbf{G}(\boldsymbol{x}) = \boldsymbol{J}_{\psi}(\boldsymbol{x})^{\top}\boldsymbol{J}_{\psi}(\boldsymbol{x}), \tag{8.4}$$

where $\boldsymbol{J}_{\psi}(\boldsymbol{x})$ is the Jacobian of ψ.

Theorem 1 ((Rana et al., 2020)) *Let $\Phi : \mathbb{R}^d \to \mathbb{R}$ be a potential function, and $\psi : \boldsymbol{M} \to \mathbb{R}^d$ be a diffeomorphism mapping between some manifold $\boldsymbol{M}$ to $\mathbb{R}^d$ Euclidean space. If the system defined by $\dot{\boldsymbol{z}} = -\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}) \in \mathbb{R}^d$ is globally asymptotically stable, then the system defined by the natural gradient descent in eq.* (8.3) *is globally asymptotically stable.*

Building inherently stable systems can be done in a principled fashion by specifying a stable potential function $\boldsymbol{\Phi}$ in Euclidean space, and constructing diffeomorphisms, ψ, such that the natural gradient system of $\boldsymbol{\Phi}$, as defined in eq. (8.3), matches a desired *target system*. The problem at hand becomes how to construct diffeomorphisms such that the natural gradient system of a known stable system exhibits the desired behaviours.

8.4 Diffeomorphic Transforms

In this section, we introduce diffeomorphic transforms (DTs) as a tool to generalise imitation learning. Diffeomorphic Transforms are diffeomorphisms which encode desired behaviour. By composing multiple DTs sequentially, we can build a stable system which is capable of incorporating various behaviours such as imitating demonstrations and generalising to novel environments. For example, by composing two DTs, one encoding the imitation of demonstrations and the other encoding obstacle avoidance, we can obtain a stable system that attempts to follow the given demonstrations while avoiding obstacles in the environment.

This section is structured as follows: We start by discussing how multiple DTs can sequentially be composed together (section 8.4.1). Next, we outline the general theory on which DTs are built and how to imbue DTs with desirable properties (section 8.4.2). Thereafter, we derive DTs with the following behaviours: (i) imitating demonstration data (section 8.4.3), (ii) avoiding collision (section 8.4.4), and (iii) biasing towards specified coordinates (section 8.4.6).

8.4.1 Composition of Diffeomorphic Transforms

Diffeomorphisms provide smooth bijective mappings between manifolds. If we have two manifolds which are linked by a diffeomorphism and a system on one manifold, we can find a natural gradient system on the other manifold via eq. (8.3). In this chapter, diffeomorphisms map between manifolds of the same dimension and can be thought of as a change of coordinate systems.

We denote diffeomorphisms by ψ and manifolds by $\boldsymbol{M}$ with superscripts for specific diffeomorphisms or manifolds. Previous work Rana et al. (2020); Ratliff (2013), define forward diffeomorphisms ψ as a mapping from some warped manifold to Euclidean space. We adopt this convention and define the diffeomorphisms ψ of a DT as a mapping from some manifold which encodes desirable behaviour to one that does not. Correspondingly, the inverse ψ^{-1} maps to a manifold with encoded behaviour. Compositions of DTs can be done in a sequential manner. Consider a simple system in Euclidean space linked by a DT to its natural gradient on some manifold, which in turn is linked to another natural gradient system on another manifold by another DT, and so forth. Each of the defined dynamical systems shares the same stability attributes as they are linked via a chain of diffeomorphic transforms.

A specific example of such a composition is given in Figure 8.2, where we obtain a system which imitates demonstrations while also avoiding collisions. We start with a stable attractor in Euclidean space, with potential $\Phi(\boldsymbol{z}) = ||\boldsymbol{z} - \boldsymbol{z}_e||_2$. The DT, ψ^{i}, is learned from demonstration data, and maps from the d-dimensional manifold, $\boldsymbol{M}^{\mathrm{i}}$ to Euclidean space. The natural gradient system

$$\dot{\boldsymbol{x}} = f_i(\boldsymbol{x}) = -\boldsymbol{G}^{\mathrm{i}}(\boldsymbol{x})^{-1}\nabla_{\boldsymbol{x}}\Phi(\psi^{\mathrm{i}}(\boldsymbol{x})), \qquad \boldsymbol{x} \in \boldsymbol{M}^{\mathrm{i}} \tag{8.5}$$

generates trajectories that mimic expert demonstrations. The DT ψ^{o} is constructed from occupancy data, and maps from another d-dimensional manifold $\boldsymbol{M}^{\mathrm{o}}$ to $\boldsymbol{M}^{\mathrm{i}}$. The resulting natural gradient system $\dot{\boldsymbol{y}} = f_{\mathrm{o}}(\boldsymbol{y}) = -\boldsymbol{G}^{\mathrm{o}}(\boldsymbol{y})^{-1}f_i(\psi^{\mathrm{o}}(\boldsymbol{y}))$ thus encodes both the imitation and collision-avoidance properties.

Throughout the chapter, we denote system states in Euclidean space by $\boldsymbol{z}$, those on a manifold encoding demonstrated behaviour by $\boldsymbol{x}$, and those on a manifold which encodes generalisation behaviour, such as collision avoidance, by $\boldsymbol{y}$. The system in Euclidean space is always given as an attractor with potential $\Phi(\boldsymbol{z}) = ||\boldsymbol{z} - \boldsymbol{z}_e||_2$ in Euclidean space, with

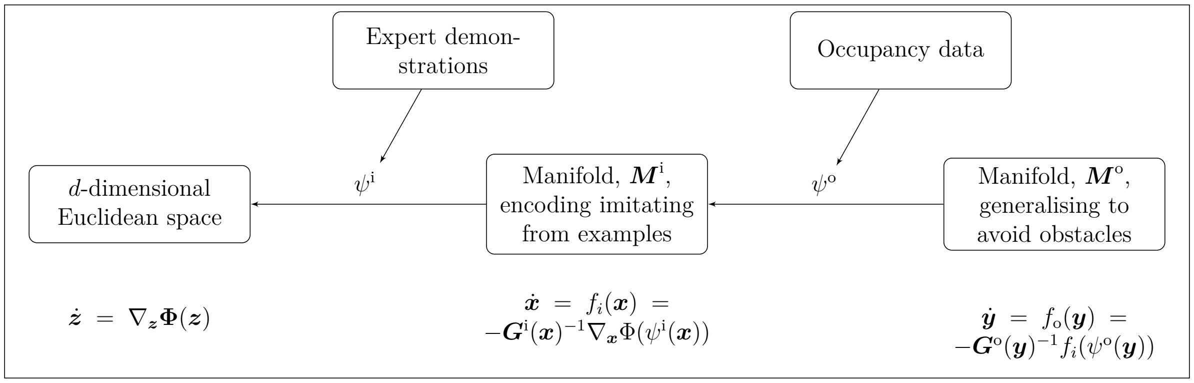

Expert demonstrations
Occupancy data
ψ^{i}
ψ^{o}
d-dimensional Euclidean space
Manifold, $\boldsymbol{M}^{\mathrm{i}}$, encoding imitating from examples
Manifold, $\boldsymbol{M}^{\mathrm{o}}$, generalising to avoid obstacles
$\dot{\boldsymbol{z}} = \nabla_{\boldsymbol{z}}\boldsymbol{\Phi}(\boldsymbol{z})$
$\dot{\boldsymbol{x}} = f_i(\boldsymbol{x}) = -\boldsymbol{G}^{\mathrm{i}}(\boldsymbol{x})^{-1}\nabla_{\boldsymbol{x}}\Phi(\psi^{\mathrm{i}}(\boldsymbol{x}))$
$\dot{\boldsymbol{y}} = f_{\mathrm{o}}(\boldsymbol{y}) = -\boldsymbol{G}^{\mathrm{o}}(\boldsymbol{y})^{-1}f_i(\psi^{\mathrm{o}}(\boldsymbol{y}))$

Figure 8.2: To obtain a stable system capable of both imitating demonstrations and avoiding collisions, we compose two DTs: ψ^{i} mapping from manifold $\boldsymbol{M}^{\mathrm{i}}$ to Euclidean space, and ψ^{o} from $\boldsymbol{M}^{\mathrm{o}}$ to manifold $\boldsymbol{M}^{\mathrm{i}}$. The natural gradient system of $\boldsymbol{\Phi}$ on $\boldsymbol{M}^{\mathrm{i}}$ reproduces the expert demonstrations, while that on $\boldsymbol{M}^{\mathrm{o}}$ avoids obstacles and imitates the demonstrations.

equilibrium point $\boldsymbol{z}_e \in \mathbb{R}^d$. Subscripts or superscripts of i, o, b, in our notation relate to the *imitation*, *obstacle-avoiding*, *bias-incorporation* capabilities we wish to endow DTs with.

8.4.2 Infinitesimal Generators of Flows

In this section, we introduce the building block of diffeomorphic transforms: Smooth flows (integral curves) generated by infinitesimal generators (smooth vector fields):

Definition 2 (Flows) *A (global) flow on $\mathbb{R}^d$ is a continuous mapping $\gamma : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$, such that $\forall \boldsymbol{s} \in \mathbb{R}^d$ and $\forall t_1, t_2 \in \mathbb{R}$, there is $\gamma(0, \boldsymbol{s}) = \boldsymbol{s}$ and $\gamma(t_2, \gamma(t_1, \boldsymbol{s})) = \gamma(t_1 + t_2, \boldsymbol{s})$.*

Definition 3 (Infinitesimal Generator) *The infinitesimal generator, $V : \mathbb{R}^d \to \mathbb{R}^d$, of a smooth flow γ is a smooth vector field mapping from each $\boldsymbol{s} \in \mathbb{R}^d$ to the tangent vector $\gamma'(0, \boldsymbol{s})$.*

We shall use the following property of flows to build DTs: For every smooth infinitesimal generator V where flows exist for all points in $\mathbb{R}^d$, the flow for time $t \in \mathbb{R}$, $\gamma(t, \cdot)$, is a diffeomorphism. Specifically, for any initial point $\boldsymbol{s}_0 \in \mathbb{R}^d$ and time t, if $\boldsymbol{s}_t = \gamma(t, \boldsymbol{s}_0) \in \mathbb{R}^d$, then $\boldsymbol{s}_0 = \gamma(-t, \boldsymbol{s}_t)$. That is, the inverse of $\gamma(t, \cdot)$ is unique and given by $\gamma(-t, \cdot)$. This is provided, with proof, in Theorem 9.12 of Lee (2012).

Intuitively, this means that following a smooth vector field V (infinitesimal generator) for time t from $\boldsymbol{s}_0$ leads to a new location $\boldsymbol{s}_t$. To return to $\boldsymbol{s}_0$ from $\boldsymbol{s}_t$ requires the reversed vector field $-V$, which for an infinitesimal generator is equivalent to following the vector field V for the negative amount of time, i.e. $-t$. This property provides us with a principled mechanism to construct a diffeomorphism: The smooth infinitesimal generator V and its associated flow γ form our diffeomorphism and its inverse, i.e.

$$\psi(\cdot) = \gamma(t, \cdot) \qquad\qquad \psi^{-1}(\cdot) = \gamma(-t, \cdot)\ . \tag{8.6}$$

Therefore the following sections describe how to construct infinitesimal generators with different properties. Once constructed these infinitesimal generators form a flow which we use as diffeomorphic transforms which we combine to form complex yet stable motion generators.

8.4.3 Learning Infinitesimal Generators from Demonstrations

We outline a method to learn smooth infinitesimal generators which encode demonstrated motions. We exploit the smoothness of Gaussian radial basis functions to parametrise the infinitesimal generator $V^{\mathrm{i}} : \mathbb{R}^d \to \mathbb{R}^d$ with a weighted combination of basis functions,

$$V^{\mathrm{i}}(\boldsymbol{s}) = \boldsymbol{W}^\top K(\boldsymbol{s}, \hat{\boldsymbol{s}}), \tag{8.7}$$

where $K(\boldsymbol{s}, \hat{\boldsymbol{s}}) = [k(\boldsymbol{s}, \hat{\boldsymbol{s}}_1), k(\boldsymbol{s}, \hat{\boldsymbol{s}}_2), \dots k(\boldsymbol{s}, \hat{\boldsymbol{s}}_m)]^\top \in \mathbb{R}^{d\times m}$ is the projection of point $\boldsymbol{s} \in \mathbb{R}^d$ to m pre-determined *inducing states*, $\hat{\boldsymbol{s}}_i \in \mathbb{R}^d$, for $i = 1, 2, \dots, m$. The radial basis functions $k(\boldsymbol{s}, \hat{\boldsymbol{s}}_i) = \exp(-\ell||\boldsymbol{s} - \hat{\boldsymbol{s}}_i||_2)$ are centered on each inducing state. The vector field V^{i} is parameterised by weight matrix $\boldsymbol{W} \in \mathbb{R}^{d\times m}$. Like Rana et al. (2020); Urain et al. (2020), the policies are defined for world space position, and are typically coordinates in $\mathbb{R}^3$. The hyper-parameter ℓ of the radial basis function controls its "width".

The flow on the infinitesimal generator is obtained by taking integral curves. As outlined in section 8.3.4, assume there exists a d-dimensional manifold $\boldsymbol{M}^{\mathrm{i}}$, where trajectories follow demonstrations. Points on $\boldsymbol{x} \in \boldsymbol{M}^{\mathrm{i}}$ are mapped to the corresponding point in Euclidean space, $\boldsymbol{z}$, by following the flow γ^{i} on V^{i} for time t. We define the diffeomorphism $\psi^{\mathrm{i}}_{\boldsymbol{W}} : \boldsymbol{M}^{\mathrm{i}} \to \mathbb{R}^d$ and its inverse as:

$$\psi^{\mathrm{i}}_{\boldsymbol{W}}(\boldsymbol{x}) := \gamma^{\mathrm{i}}(t, \boldsymbol{x}) = \boldsymbol{x} + \int_0^t V^{\mathrm{i}}(\gamma^{\mathrm{i}}(u, \boldsymbol{x}))\mathrm{d}u = \boldsymbol{z}, \tag{8.8}$$

$$\psi^{\mathrm{i}}_{\boldsymbol{W}}(\boldsymbol{z})^{-1} := \gamma^{\mathrm{i}}(-t, \boldsymbol{z}) = \boldsymbol{z} + \int_{-t}^0 V^{\mathrm{i}}(\gamma^{\mathrm{i}}(u, \boldsymbol{z}))\mathrm{d}u = \boldsymbol{x}.$$

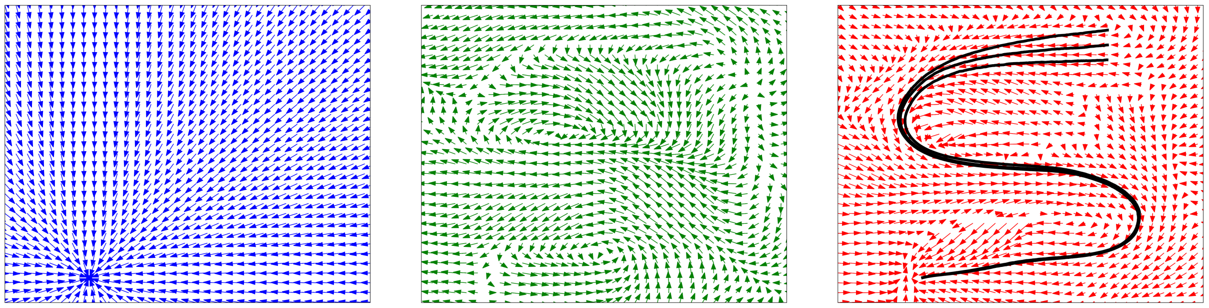


Figure 8.3: Visualisation of vector fields, learning to generate an "S" motion. Left: a stable base system, a simple attractor; Center: the infinitesimal generator $V_{\boldsymbol{W}}^{\mathrm{i}}$ learned on demonstrations of "S"; Right: the stable system $\dot{\boldsymbol{x}} = f_i(\boldsymbol{x})$ imitating demonstrations, with black trajectory rollouts.

As analytical integrals are often difficult to obtain we use numerical integration techniques, Euler's method in our case due to its simplicity.

In the context of the imitating demonstrations, as outlined in section 8.3.2, we are assumed to have N demonstrations, each a sequence of states. That is, $\hat{\xi}_i = \{\boldsymbol{x}_{i,1}, \boldsymbol{x}_{i,2}, \ldots, \boldsymbol{x}_{i,l_i}\}$, where l_i is the length of the i^{th} demonstration. The state derivatives are also assumed to be available or obtained via finite differences. By eqs. (8.2), (8.3) and (8.8), we can formulate the optimisation problem as:

$$\min_{\boldsymbol{W}} \Big\{ \sum_{i=1}^{N} \sum_{j=1}^{l_i} ||\dot{\boldsymbol{x}}_{i,j} + \boldsymbol{G}^{\mathrm{i}}(\boldsymbol{x}_{i,j})^{-1} \nabla_{\boldsymbol{x}_{i,j}} \Phi(\psi_{\boldsymbol{W}}(\boldsymbol{x}_{i,j}))||_2^2 \Big\} \tag{8.9}$$

$$\boldsymbol{G}^{\mathrm{i}}(\boldsymbol{x}_{i,j}) = \boldsymbol{J}_{\psi_{\boldsymbol{W}}}(\boldsymbol{x}_{i,j})^{\top} \boldsymbol{J}_{\psi_{\boldsymbol{W}}}(\boldsymbol{x}_{i,j}). \tag{8.10}$$

We can optimise eq. (8.9) to obtain $\boldsymbol{W}$. By eq. (8.3), the natural gradient system which generates imitated trajectories, with $\boldsymbol{x} \in \boldsymbol{M}^{\mathrm{i}}$, is then,

$$\dot{\boldsymbol{x}} = f_i(\boldsymbol{x}) = -\boldsymbol{G}^{\mathrm{i}}(\boldsymbol{x})^{-1} \nabla_{\boldsymbol{x}} \Phi(\psi_{\boldsymbol{W}}^{\mathrm{i}}(\boldsymbol{x})). \tag{8.11}$$

8.4.4 Building Infinitesimal Generators for Collision Avoidance

Beyond learning to reproduce motions from demonstration data, we wish to adapt the generated motions so that they remain collision-free when the occupied space in the environment changes. In this section, we introduce diffeomorphic transforms which encode collision-avoiding behaviours. Again, we aim to generate a diffeomorphism by constructing an appropriate smooth infinitesimal generator.

We use smoothly varying obstacle gradients to build the collision-avoiding diffeomorphic transforms. These are obtained via a continuous occupancy representation, such as the representation described in Ramos and Ott (2016). We exploit its continuous nature to find analytical gradients of the probability of being occupied. Using occupancy information about the new environment in the form of a training dataset of n pairs, $\{(\boldsymbol{s}_k, o_k)\}_{k=1}^{n}$, where each sample $\boldsymbol{s}_k \in \mathbb{R}^d$ is a point in the state-space, and $o_k \in \{0, 1\}$ is a binary variable indicating whether $\boldsymbol{s}_k$ is occupied or free. Ramos and Ott (2016) use a kernelised logistic regressor Bishop (2007) wrapped around Gaussian radial basis functions to learn a mapping between coordinates to the probability of them being occupied (occupancy). Likewise, we use the continuous occupancy representation introduced in Ramos and Ott (2016). The continuous representation is given as $f^{\text{map}}(\boldsymbol{s}) := p(o = 1|\boldsymbol{s})$, and is smooth as it consists of sigmoid and Gaussian functions. Unlike signed distance representations (Malladi et al., 1995), the produced gradients of occupancy, w.r.t. coordinates, are smooth over the entire domain. We exploit the smooth obstacle gradients to construct an infinitesimal generator:

$$V^o(\boldsymbol{s}) = \nabla_{\boldsymbol{s}} p(o = 1|\boldsymbol{s}) = \nabla_{\boldsymbol{s}} f^{\text{map}}. \tag{8.12}$$

A simple example of a continuous occupancy representation from occupancy data is given in fig. 8.4 (left and center). Gradient vectors with large negative values in $-V$ are shown in fig. 8.4 (right). Following the flow on V^o in reverse time reduces the observed occupancy, i.e. lower collision likelihood. This captures the goal of the inverse diffeomorphism ψ^{o-1}, which is designed to map to a manifold encoding collision-avoiding behaviour. Accordingly, we define a d-dimensional manifold $\boldsymbol{M}^{\text{o}}$ where trajectories exhibit obstacle avoidance behaviour.

Provided some manifold $\boldsymbol{M}$, the infinitesimal generator V^o generates flow γ^o. For some $t \in \mathbb{R}$, let $\boldsymbol{y} \in \boldsymbol{M}^{\text{o}}$ and $\boldsymbol{x} \in \boldsymbol{M}$, we can define the diffeomorphism $\psi^o : \boldsymbol{M}^{\text{o}} \to \boldsymbol{M}$ as:

$$\psi^o(\boldsymbol{y}) := \gamma^o(t, \boldsymbol{y}) = \boldsymbol{y} + \int_0^t V^o(\gamma^o(u, \boldsymbol{y}))\mathrm{d}u = \boldsymbol{x}, \tag{8.13}$$

and its inverse as

$$\psi^o(\boldsymbol{x})^{-1} := \gamma^o(-t, \boldsymbol{x}) = \boldsymbol{x} + \int_{-t}^0 V^o(\gamma^o(u, \boldsymbol{x}))\mathrm{d}u = \boldsymbol{y}, \tag{8.14}$$

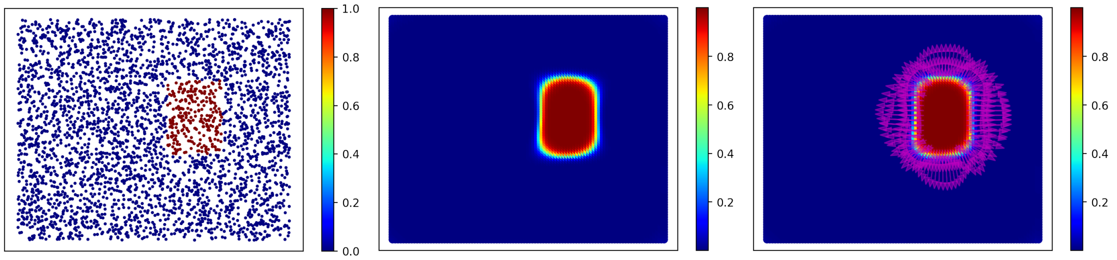


Figure 8.4: A 2d example of building a continuous occupancy representation. (Left) Occupancy data; (Centre) A continuous function mapping from coordinates to the probability of being occupied; (Right) Gradients of the probability of being occupied, wrt coordinates, are available and displayed in magenta.

where a numerical integrator is used to evaluate the integral. The natural gradient system on $\boldsymbol{M}^{\text{o}}$ can then be defined as:

$$\dot{\boldsymbol{y}} = -\boldsymbol{G}^{\text{o}}(\boldsymbol{y})^{-1} f(\psi^{o}(\boldsymbol{y})), \tag{8.15}$$

where f corresponds to a dynamical system on $\boldsymbol{M}$. This could be for example, the original simple attractor or f_{i} that encodes demonstrations.

When rolling out trajectories with the collision-avoiding system, the initial conditions given by some $\boldsymbol{y_0} \in \boldsymbol{M}^{\text{o}}$ are assumed to be collision-free. As $\boldsymbol{y}$ approaches the boundary of the obstacle, the natural gradient system smoothly warps around the obstacle. Figure 8.5 (right) gives a simple 3d example of trajectories of the natural gradient system on $\boldsymbol{M}^{\text{o}}$ which smoothly avoids the cube obstacle. The corresponding system on $\boldsymbol{M}$, illustrated in fig. 8.5 (left), is a simple attractor. Although the example given is a simple attractor, the system on $\boldsymbol{M}$ can encode imitation.

8.4.5 Building Infinitesimal Generators for Collision Avoidance in C-space

In imitation learning setups with a manipulator, it is often necessary to learn a stable dynamical system that governs the Cartesian *world-space* coordinates of the end-effector. At the same time, collision avoidance in such a scenario is typically not limited to the end-effector but required for the entire manipulator based on its *configuration-space* parametrisation. We can define diffeomorphic transforms which encode avoidance behaviours for multiple body

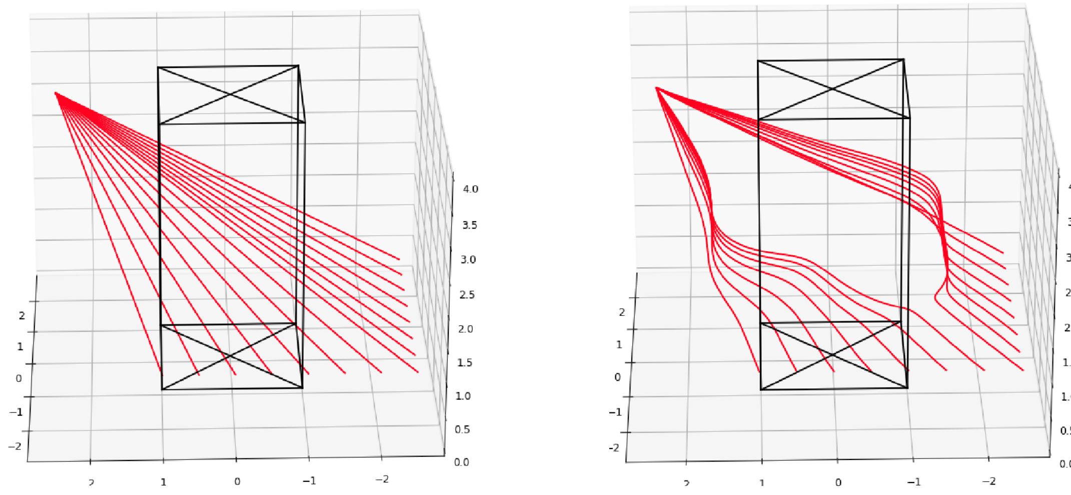


Figure 8.5: A 3d example of avoiding a cube obstacle (black). (Left) As a simple example, an attractor $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ is on $\boldsymbol{M}$; (Right) The corresponding natural gradient system on $\boldsymbol{M}^{\mathrm{o}}$, given by $\dot{\boldsymbol{y}} = f_{\mathrm{o}}(\boldsymbol{y})$ generates trajectories (red) which avoid the obstacle by smoothly warping around.

points on the manipulator operating in the configuration space (C-space) (Lavalle, 2006) of the robot. We focus our discussion on the common case where the manipulator configuration is given by joint angles.

A $d_{\boldsymbol{q}}$-dimensional configuration of the manipulator is defined as $\boldsymbol{q} \in \mathcal{Q} \subset \mathbb{R}^{d_{\boldsymbol{q}}}$, where the C-space is denoted as $\mathcal{Q}$. Robot configurations can be mapped into the world-space coordinates of c body points on the robot by a set of c kinematic functions, $\{\mathbf{x}_i^{\mathrm{f}}\}_{i=1}^{c}$. Each kinematic function, $\mathbf{x}_i^{\mathrm{f}} : \mathcal{Q} \to \mathbb{R}^d$, represents the Cartesian world space coordinates of the i^{th} body point from a given configuration. For multiple-link manipulators, these kinematic functions are often modelled by of simple operations on basic trigonometry functions (Denavit et al., 1955) and are smooth with respect to the configurations. We set the manipulator end-effector as the first body point, thus $\mathbf{x}_1^{\mathrm{f}}(\cdot)$ is the forward kinematics of the robot. Denoting the end-effector position as $\mathbf{x}^e$, the velocity of the end-effector and the configurations are described by:

$$\dot{\mathbf{x}^e} = \boldsymbol{J}_{\mathbf{x}_1^{\mathrm{f}}}(\boldsymbol{q})\dot{\boldsymbol{q}}, \qquad \dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}_{\mathbf{x}_1^{\mathrm{f}}}(\mathbf{x}^e)\dot{\mathbf{x}^e}, \qquad \boldsymbol{J}_{\mathbf{x}_1^{\mathrm{f}}}(\cdot) = \frac{\partial \mathbf{x}_1^{\mathrm{f}}}{\partial \boldsymbol{q}}(\cdot). \tag{8.16}$$

The dimensions of $\boldsymbol{q}$ are typically greater than that of the end-effector coordinates $\mathbf{x}^e$, and many configurations can map to the same end-effector coordinates. To obtain a unique $\dot{\boldsymbol{q}}$ for some $\dot{\boldsymbol{x}}$, we take $\boldsymbol{J}^{\dagger}_{\mathrm{x}^{\mathrm{f}}_1} = \boldsymbol{J}^{\top}_{\mathrm{x}^{\mathrm{f}}_1}(\boldsymbol{J}_{\mathrm{x}^{\mathrm{f}}_1}\boldsymbol{J}^{\top}_{\mathrm{x}^{\mathrm{f}}_1})^{-1}$ as the the Moore-Penrose pseudoinverse. For some $\dot{\boldsymbol{x}}$, its product with $\boldsymbol{J}^{\dagger}_{\mathrm{x}^{\mathrm{f}}_1}$ provides the unique $\dot{\boldsymbol{q}}$ where $||\dot{\boldsymbol{q}}||$ is also minimised. The Moore-Penrose pseudoinverse preserves stability of the resulting system (Wang et al., 2018). Provided a dynamical system in the world-space coordinates of the end-effector, we can use $\boldsymbol{J}^{\dagger}_{\mathrm{x}^{\mathrm{f}}_1}$ to *pull* the end-effector velocity to C-space and obtain $\dot{\boldsymbol{q}}$. If the end-effector follows a dynamical system given by $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$, we can define the system in C-space:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}_{\mathrm{x}^{\mathrm{f}}_1} f(\mathrm{x}^{\mathrm{f}}_1(\boldsymbol{q})). \tag{8.17}$$

Using inverse kinematics, we can find a starting configuration $\boldsymbol{q}_0$ which matches up to the initial world-space coordinate, i.e. $\mathrm{x}^{\mathrm{f}}_1(\boldsymbol{q}_0) = \boldsymbol{x}_0$, then we can numerically integrate to roll out a sequence of configurations.

In order for body points on the robot to be collision-free, we shall extend our approach in section 8.4.4 to construct an infinitesimal generator to encode collision-avoidance in C-space. Instead of using the derivative of occupancy with respect to world-space coordinates, we make use of the derivative of occupancy over all body points with respect to configurations. Provided a continuous occupancy representation, as given in eq. (8.12), the derivative of the combined occupancy of body points, wrt to configurations, gives smooth vector field V^o_q,

$$V^o_{\boldsymbol{q}}(\boldsymbol{q}) := \nabla_{\boldsymbol{q}} \sum_{j=1}^{c} f^{map}(\mathrm{x}^{\mathrm{f}}_j(\boldsymbol{q})), \tag{8.18}$$

where f^{map} is the continuous occupancy representation as outlined in section 8.4.4. We can then use $V^o_{\boldsymbol{q}}$ as the infinitesimal generator of a diffeomorphism in a manner similar to eq. (8.8). This gives us a $d_{\boldsymbol{q}}$-dimensional manifold, which we shall denote as $\mathcal{Q}^o$, where natural gradient systems corresponding to systems in $\mathcal{Q}$ are collision-free. The diffeomorphism, $\phi^o_{\boldsymbol{q}}$, mapping from $\mathcal{Q}^o$ to $\mathcal{Q}$ is given by following the flow, $\gamma^o_{\boldsymbol{q}}$, on $V^o_{\boldsymbol{q}}$. For a configuration $\boldsymbol{q} \in \mathcal{Q}$, and its corresponding collision-avoiding configuration $\boldsymbol{p} \in \mathcal{Q}^o$, we define the diffeomorphism:

$$\psi^o_{\boldsymbol{q}}(\boldsymbol{p}) := \gamma^o(t, \boldsymbol{y}) = \boldsymbol{p} + \int_0^t V^o_{\boldsymbol{p}}(\gamma^o(u, \boldsymbol{p}))\mathrm{d}u = \boldsymbol{q} \tag{8.19}$$

and its inverse:

$$\psi^o_{\boldsymbol{q}}(\boldsymbol{q})^{-1} := \gamma^o(-t, \boldsymbol{q}) = \boldsymbol{q} + \int_{-t}^0 V^o_{\boldsymbol{q}}(\gamma^o(u, \boldsymbol{q}))\mathrm{d}u = \boldsymbol{p}. \tag{8.20}$$

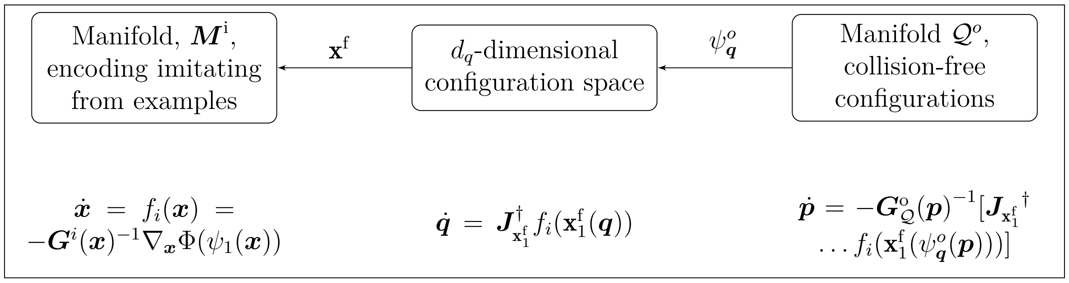

Manifold, $\boldsymbol{M}^{\mathrm{i}}$, encoding imitating from examples
$\mathbf{x}^{\mathrm{f}}$
d_q-dimensional configuration space
$\psi^o_{\boldsymbol{q}}$
Manifold $\mathcal{Q}^o$, collision-free configurations
$\dot{\boldsymbol{x}} = f_i(\boldsymbol{x}) = -\boldsymbol{G}^i(\boldsymbol{x})^{-1}\nabla_{\boldsymbol{x}}\Phi(\psi_1(\boldsymbol{x}))$
$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}_{\mathbf{x}^{\mathrm{f}}_1} f_i(\mathbf{x}^{\mathrm{f}}_1(\boldsymbol{q}))$
$\dot{\boldsymbol{p}} = -\boldsymbol{G}^{\mathrm{o}}_{\mathcal{Q}}(\boldsymbol{p})^{-1}[\boldsymbol{J}_{\mathbf{x}^{\mathrm{f}}_1}{}^{\dagger} \dots f_i(\mathbf{x}^{\mathrm{f}}_1(\psi^o_{\boldsymbol{q}}(\boldsymbol{p})))]$

Figure 8.6: An example of a system on $\mathcal{Q}^{\mathrm{o}}$, for imitating demonstrations and avoiding collision along body points. The mappings between manifolds in world-space and C-space, along with the system constructed on each manifold is shown.

Following eq. (8.3), we can use the diffeomorphism to define our system on $\mathcal{Q}^o$ as:

$$\dot{\boldsymbol{p}} = -\boldsymbol{G}^{\mathrm{o}}_{\mathcal{Q}}(\boldsymbol{p})^{-1}[\boldsymbol{J}_{\mathbf{x}^{\mathrm{f}}_1}{}^{\dagger} f_i(\mathbf{x}^{\mathrm{f}}_1(\psi^o_{\boldsymbol{q}}(\boldsymbol{p})))], \tag{8.21}$$

$$\boldsymbol{G}^{\mathrm{o}}_{\mathcal{Q}}(\boldsymbol{p}) = \boldsymbol{J}_{\psi^o_{\boldsymbol{q}}}(\boldsymbol{p})^{\top}\boldsymbol{J}_{\psi^o_{\boldsymbol{q}}}(\boldsymbol{p}). \tag{8.22}$$

Figure 8.6 shows a specific example of using diffeomorphic transforms to construct a system that avoids collision along chosen body points and also imitates demonstrations. Provided a stable dynamical system on $\mathbf{M}^i$, $\dot{\boldsymbol{x}} = f_i(\boldsymbol{x})$, learned from demonstration data in world-space (as outlined in section 8.4.3), we can pull back the system to C-space $\mathcal{Q}$ via eq. (8.17). The natural gradient system on the collision-avoiding manifold in C-space is then given in eq. (8.21), we can roll out $\boldsymbol{p}$ to obtain collision-free configurations.

8.4.6 Infinitesimal Generators for User-Specified Bias

Sometimes it is desirable to deform learned motions based on user input without recording new expert demonstrations. In this section, we outline a method to construct a diffeomorphism which can apply deformation biases to specific coordinates along a trajectory.

Similar to previous sections, we focus on building diffeomorphic transforms by directly crafting their infinitesimal generator. We design the manifold $\boldsymbol{M}^{\mathrm{b}}$ by constructing the diffeomorphism ψ^b such that the inverse $\psi^{b^{-1}}$ maps straight trajectories in a specific region to trajectories with the specified bias. As we define $\psi^{b^{-1}}$ from reverse-time integral curves of the generator, we need to build the generator V^{b} such that the biased direction is given by $-V^{\mathrm{b}}$. This is achieved by regressing specified vectors onto the infinitesimal generator representation introduced in eq. (8.7). We collect sample points within a fixed region in the

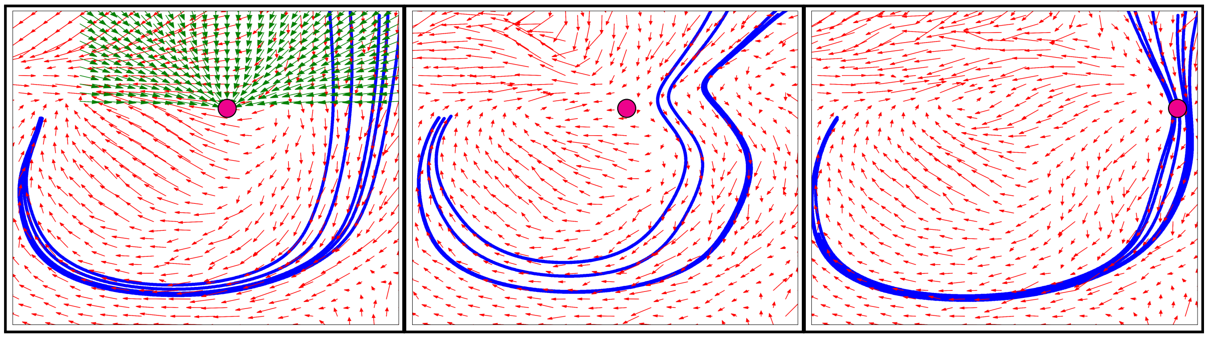


Figure 8.7: (Left) a stable dynamical system (visualised as red vector field) on $\boldsymbol{M}^{\mathrm{i}}$ learned to imitate drawing a 'J' (example trajectories in blue). We wish to bias the system to the magenta point, the green vector field illustrates, $-V^{\mathrm{b}}(\boldsymbol{s})$; (Center) the resulting system by biasing towards the magenta point is shown; (Right) biasing the stable system towards another coordinate (magenta point).

state-space and compute a unit vector pointing away from the point of interest, giving us a dataset of $n_{\boldsymbol{s}}$ pairs $\mathcal{D} = \{(\boldsymbol{s}_i^{\mathrm{a}}, \nabla_{\boldsymbol{s}_i^{\mathrm{a}}}||\boldsymbol{s}_i^{\mathrm{a}} - \boldsymbol{s}^{tar}||_2)\}_{i=1}^{n_{\boldsymbol{s}}}$, where $\boldsymbol{s}^{\mathrm{a}} \in \mathcal{S}$ are sample points in some region $\mathcal{S}$ where the bias takes effect, and $\boldsymbol{s}^{tar}$ is the coordinate of interest. To produce a valid diffeomorphism, we require an infinitesimal generator to be smooth. This is achieved by fitting a smooth function approximator introduced in eq. (8.7) on $\mathcal{D}$, by minimising:

$$\boldsymbol{W}^* := \arg\min_{\boldsymbol{W}} \sum_{i=1}^{n_{\boldsymbol{s}}} ||\nabla_{\boldsymbol{s}_i^{\mathrm{a}}}||\boldsymbol{s}_i^{\mathrm{a}} - \boldsymbol{s}^{tar}||_2) - \boldsymbol{W}^\top K(\boldsymbol{s}_i, \hat{\boldsymbol{s}})||_2^2, \tag{8.23}$$

where we now have a smooth $V^{\mathrm{b}} := \boldsymbol{W}^{*\top} K(\boldsymbol{s}, \hat{\boldsymbol{s}})$. Like eq. (8.8), we can integrate V^{b} to obtain diffeomorphism ψ^b. With eq. (8.3) we can take natural gradients on the resulting manifold $\boldsymbol{M}^{\mathrm{b}}$.

An example of biasing a learned stable system towards two different coordinates of interest is illustrated in fig. 8.7. Constructing a DT for complicated morphing on the original system may be difficult, as it is challenging to intuit what the infinitesimal generator needs to look like. Furthermore, unexpected behaviour may arise in complex setups, as stretching in one space contracts another (Ratliff, 2013). However, for relatively simple cases such as bias towards a point or direction, a simple generator such as V^b, which points away from the point of interest, results in the desired behaviour.

8.5 Experimental Evaluation

We empirically evaluate the performance of diffeomorphic transforms to imitate and generalise robot motions, both in simulation and on a real-world 6-DOF JACO manipulator. In section 8.5.1, we evaluate the ability of our method to accurately reproduce human demonstrations. In section 8.5.2, we evaluate the generalisation performance of our approach when occupancy in the environment changes and compare our results against multiple baselines. Additionally, in section 8.5.3, we demonstrate the ability of diffeomorphic transforms to generalise based on user-specified requirements.

8.5.1 Imitating Expert Demonstrations

The first analysis we carry out is on the ability of diffeomorphic transforms to imitate provided demonstrations. We train diffeomorphic transforms from 8 alphabet characters from the LASA handwriting dataset (Khansari-Zadeh and Billard, 2011). There are 7 demonstrations for each character. We use $m = 720$ inducing states for the Gaussian basis function, each with hyperparameter $\ell = 0.005$, arranged in an equally-spaced grid covering the range of the demonstrations. To build the diffeomorphism. Qualitative results are presented in fig. 8.8, where generated trajectories are given in red with demonstrations in green. We observe that the generated trajectories are smooth and consistent with the demonstrated motions. We note that first-order dynamical systems cannot model intersections which exist in the data (Dupont et al., 2019), and explains some of the differences between generated and ground truth motion. These results demonstrate that our method is sufficiently flexible to learn diffeomorphic transforms to morph the simple base attractor from complex demonstrations.

8.5.2 Generalised Collision-Avoiding Motion Reproduction

In this section, we evaluate the efficacy of diffeomorphic transforms to adapt the motion according to the occupancy of the surroundings and avoid collisions. We work with demonstrations from the following tasks: (i) pushing a box (7 demonstrations collected with JACO arm); (ii) lifting a box, by reaching inside and abruptly moving upwards (8 demonstrations collected with JACO arm); (iii) drawing an “S” character (7 demonstrations from LASA dataset). We train diffeomorphic transforms to imitate expert demonstrations to complete

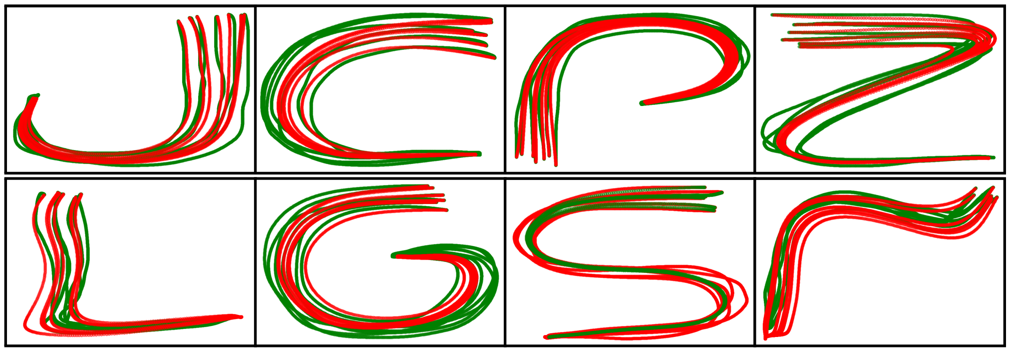


Figure 8.8: Imitating expert examples from the LASA dataset (Khansari-Zadeh and Billard, 2011). Reproduced motion in red and demonstrations in green.

these tasks. Then, we place an obstacle of diameter 15cm such that the original trajectories are in collision. We then provide occupancy data of the new obstacle, and construct diffeomorphic transforms in C-space to provide collision avoidance for the entire manipulator.

Baseline Methods

We evaluate the performance of the following methods and variants:

1. Our method with DTs to imitate demonstrations and avoid collisions.
2. Our method with only imitation transformation.
3. Euclideanizing Flows (Eflow) (Rana et al., 2020).
4. Eflow with an additional repulsor overlaid for collision avoidance.
5. Motion-planning networks (MPNet) (Qureshi et al., 2021): Although MPNet is typically used for planning, at its core, it uses a neural network to imitate demonstrations from a motion planner.
6. MPNet with RRT replanning (MPNet-RRT) (Qureshi et al., 2021): a variant where results are touched up by a RRT planner (Lavalle, 1998).
7. Linear motion in the C-space provided start and end configurations.

It is tempting to generalise imitation learning by simply learning on more demonstrations gathered in more diverse scenarios. Hence, MPNet provides a learning-based comparison, as

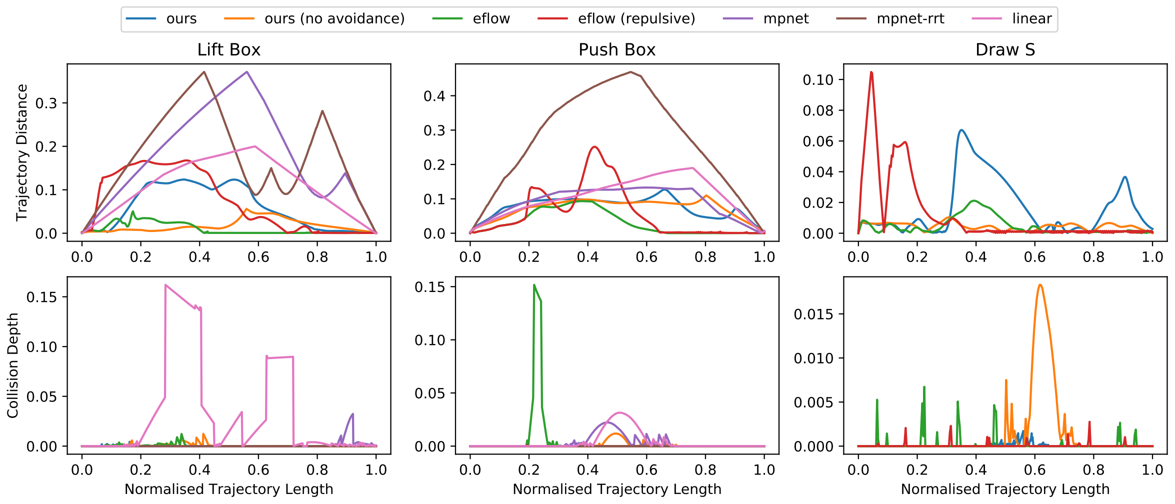


Figure 8.9: Qualitative results of diffeomorphic transforms along with comparisons. We normalise the generated trajectories by arc-length, and report the trajectory distance and M.C.D. at fixed intervals. For the draw "S" task, results for MPNet, MPNet-RRT, and linear are discarded. These methods were unable to imitate, and departed greatly from the ground truth.

it attempts to learn avoidance directly from examples of a planner. Training data of non-colliding trajectories in a variety of environment occupancy scenarios are required to train MPNet and MPNet-RRT. For each task, we simulate 200 environments each with a simulated collision-free demonstration obtained with planners along with demonstration waypoints.

Evaluation Results

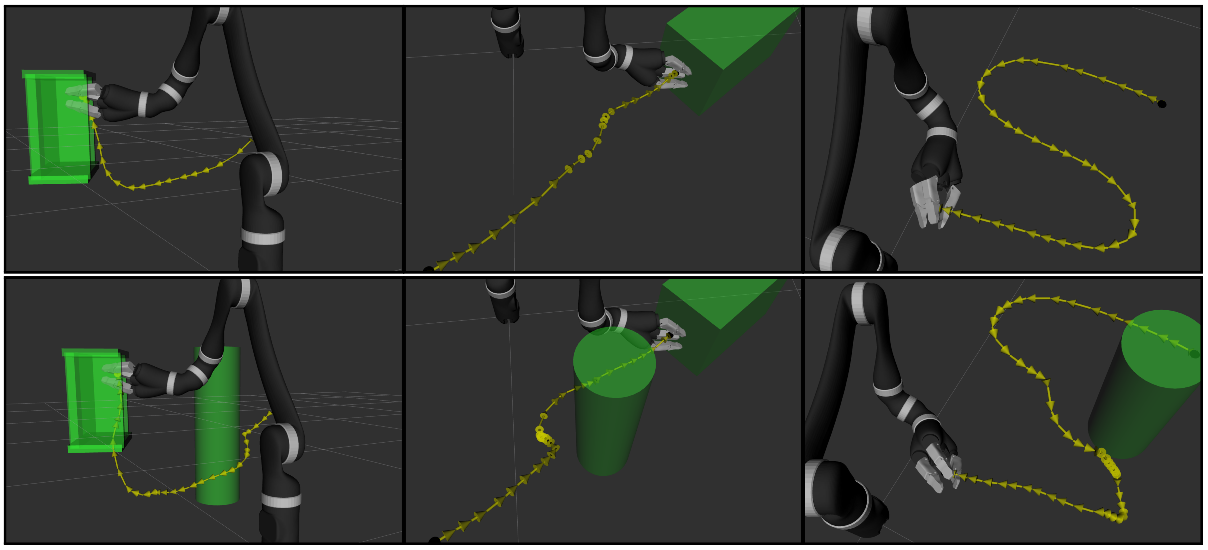
We quantitatively evaluate the quality of the generalised imitations produced under environment occupancy changes. Results are shown in fig. 8.9, and tabulated in table 8.1. In fig. 8.9, the metrics used are: (i) trajectory distance (metres), the distance between a point on a generated trajectory and the nearest coordinate of a demonstration; (ii) Maximum Collision Distance (M.C.D.), the depth of the manipulator is in collision with the obstacle. We evaluate both metrics at fixed intervals along arc-length normalised generated trajectories. In table 8.1, we report: (i) the Fréchet distance (Alt and Godau, 1995) between the generated and ground truth trajectories, which gives a single value over the entire trajectories; (ii) M.C.D..

We observe from tabulated figures, for pure imitation, our method generates trajectories

Table 8.1: Performance, as measured by Frechet Distance (F.D.) and Maximum Collision Distance (M.C.D.) (3 d.p.)

	Distance	Push Box	Lift Box	Draw S
Ours	F.D	0.124	0.127	0.067
	M.C.D	0	0	0.002
Ours (no avoidance)	F.D	0.070	0.112	0.011
	M.C.D	0.012	0.012	0.018
EFlow	F.D	0.068	0.093	0.021
	M.C.D	0.012	0.152	0.007
EFlow (repulsive)	F.D	0.168	0.252	0.195
	M.C.D	0	0.001	0.003
MPNet	F.D	0.371	0.133	N/A
	M.C.D	0.038	0.022	N/A
MPNet-RRT	F.D	0.371	0.478	N/A
	M.C.D	0	0	N/A
Linear	F.D	0.242	0.205	N/A
	M.C.D	0.147	0.031	N/A

with a relatively low Frechet Distance to the ground truth, comparable to Eflow. After factoring in the obstacle for both our method and Eflow, our method is able to remain relatively close to the original demonstration, without large changes. In contrast, Eflow (repulsive) typically has a higher distance, particularly when an obstacle is reached, the repulsion results in sharper and larger changes relative to the original demonstration, while our method manages to diverge from the original demonstration in a smoother and more gradual manner. This is highlighted in the sharper peaks of trajectory distance in fig. 8.9 by Eflow (repulsive). The M.C.D. results in table 8.1 show that both our method and Eflow (repulsive) are able to significantly reduce collision, with ours performing slightly better. In addition, unlike our approach, Eflow with added repulsor does not have theoretical guarantees on stability. Our method produces non-colliding trajectories for the first two tasks, and is in contact with the obstacle briefly in the “S”-drawing task, with a M.C.D. of 2mm. MPNet and MPNet-RRT produced trajectories that are less similar to given examples. Notably, the collision-avoidance capability of MPNet (without RRT-replanning) is significantly less than our approach, which indicates the difficulty to learn directly from collision-avoiding motions. This is particularly the case in an imitation learning setup, when the number of demonstrations is few.



(a) Lift Box (b) Push Box (c) Draw S

Figure 8.10: The simulated environments under the MoveIt framework (Coleman et al., 2014), where the top row illustrates the trajectories integrated from the infinitesimal generator and the bottom row contains the corresponding trajectories after applying diffeomorphic transformation for obstacle avoidance.

In fig. 8.10, we illustrate generated motion in both the absence and in the presence of the added obstacles. We see in all cases, with the addition of the obstacle, the manipulator warps smoothly around the cylinder obstacle, while maintaining a similar shape required the original task upon passing the obstacles. This is particularly pronounced in both the lift-box and draw "S" tasks, where the upwards movement to lift box and the general shape of the "S" character is largely intact.

Evaluation on Real Robot

To evaluate the robustness of our method, we repeat the three tasks on a real-world JACO manipulator, placing obstacles in the path of demonstrations (fig. 8.11 and fig. 8.1). For all three tasks, the JACO arm was able to warp around the obstacle with no visible collision, and successfully complete the task. A video of the real-world experiments can be found at `https://youtu.be/LoLQ0bzfw9I`.

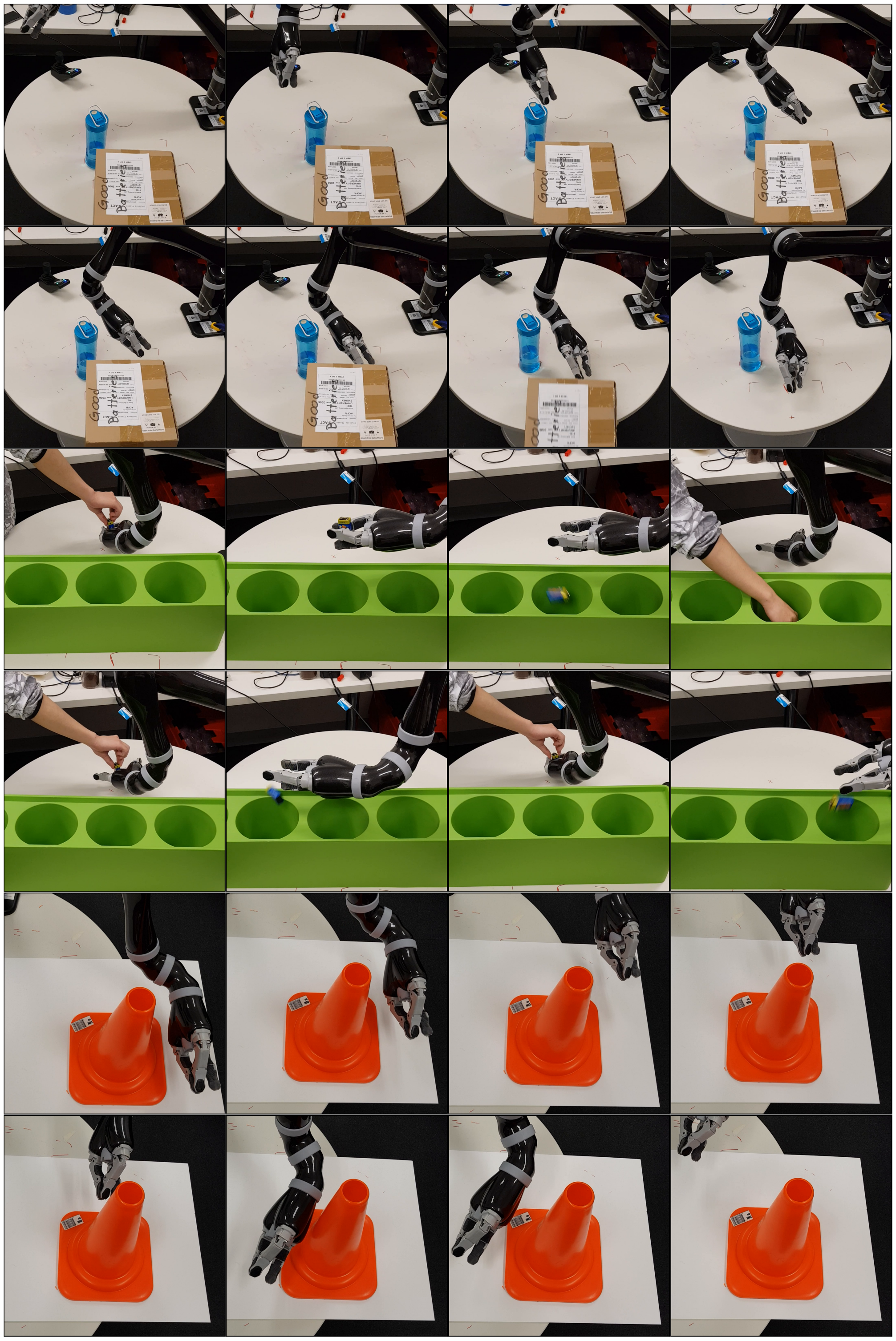
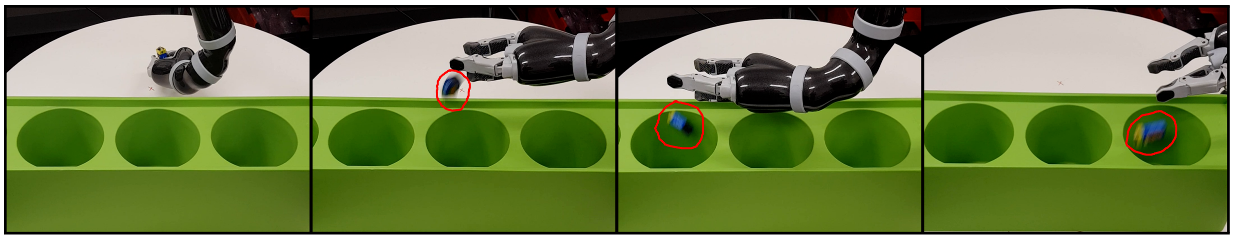


Figure 8.11: The *Push Box*, *Drop to Pot*, *Draw S* experiments on the JACO robot.



(a) Start position (b) center (c) Left pot (d) Right pot

Figure 8.12: We provide 8 demonstrations of dropping moving over the center pot to drop an object, we can create user-specific diffeomorphisms to warp the reproduced movement to drop the object in the left or right pots. Left-most image shows starting position. The dropped objects are circled in red.

8.5.3 Generalised User-Influenced Motion Reproduction

We also demonstrate the ability diffeomorphic transforms possess to be built based on user specifications. A simulated case studied of warping a dynamic system trained on imitations of drawing "J" characters, based user specified points, is illustrated in fig. 8.6. We observe by crafting diffeomorphic, the original system is morphed to bias the designated positions.

We extend our experiments to real-world robots and evaluate the ability of the dynamical system to be biased in user-specified positions with the JACO: We provide demonstrations of moving on top of a pot to drop an object. Then, we craft diffeomorphic transforms to morph the motion to either side such that the reproduced motion drops the object in neighbouring pots. This is performed by following the outline in section 8.4.6 and setting the position above the neighbouring pots as biased coordinates. Qualitative evaluations are provided in fig. 8.12. We see that without additional demonstrations, by applying diffeomorphic transforms, we can shift the robot motion to neighbouring pots. The objects were then successfully dropped in the corresponding pots.

8.6 Summary

Diffeomorphisms provide a principled method to transform dynamical systems such that stability properties are not altered. We utilise this property and propose a framework which generalises imitation learning in scenarios where changes in the environment renders collected demonstrations infeasible due to. We modularise robot motion using diffeomorphic transforms, where each diffeomorphic transform encodes specific desirable behaviours, including imitating demonstrations, avoiding obstacles, and incorporating simple specified biases. When these individual transforms are sequentially applied to a known stable system the resulting system inherits the stability properties of the known system. We demonstrate the capabilities of this novel approach on a range of imitation learning tasks in both simulation and on a real robot.

In the next chapter 9, we shall contribute a method for generating reactive and also global motion in a motion planning and control problem setup. We additionally develop a self-supervised learning framework to warm-start and speed-up the motion generation process.

Chapter 9

Motion Generation with Geometric Fabric Command Sequences

9.1 Introduction

Motion generation is a central problem in robotics. It is concerned with finding collision-free and executable trajectories from start to goal. Generating robot motions in dynamic environments is particularly challenging: the motion needs to be reactive to changes, and also be intuitive to any spectating humans. Recent reactive approaches to motion generation, such as Riemannian Motion Policies (Ratliff et al., 2018) and Geometric Fabrics (Van Wyk et al., 2022), produce local policies to efficiently generate smooth and natural motions. These approaches produce a policy that find the instantaneous optimal controls, and allow for perturbations and deviations in the environment. However, these locally optimal solutions are often globally infeasible, resulting in robots getting stuck in local minima. Sampling-based planners, such as RRTs (LaValle and James J. Kuffner, 2001), can find global solutions, but often produce non-intuitive paths, and typically require re-planning to handle deviations and local changes in the environment.

In this work, we introduce *Geometric Fabric Command Sequences* (GFCS), a new approach to motion generation that produces globally feasible solutions while retaining local

This chapter has been accepted and will appear in ICRA as Zhi et al. (2023).

reactivity. In broad strokes, our method solves the global motion generating problem by optimising over a sequence of attractor states, which we call *commands*, for a *Geometric Fabric* policy (Van Wyk et al., 2022). Geometric Fabrics produce local motion by combining decoupled dynamical systems, which each represent an individual behaviour, such as goal reaching, obstacle avoidance, and joint limit handling. We propose to directly apply global optimisation over the commands. Furthermore, we hypothesise the solutions of the optimisation are transferable over different problem setups. We take a *self-supervised* approach and learn, over a dataset of motion generation problems and solutions, to warm-start the optimisation procedure. Specifically, we use an implicit generative model to progressively learn to recommend candidate solutions, as more and more problems are solved, with solutions of past optimisation runs used as training labels. This allows for faster and higher quality solutions as more problems are solved. As time progresses, the implicit model itself can accurately generate optimal solutions, without further optimisation.

Concretely, we contribute: (1) *Geometric Fabric Command Sequences* (GFCS), a formulation of global and reactive motion generation as optimisation over sequentially connected commands, inputted to *Geometric Fabrics* (Van Wyk et al., 2022); (2) *Self-supervised GFCS*, a learning approach to warm-start the optimisation, using self-labelled data. Thus, improving the speed and quality of motion generation.

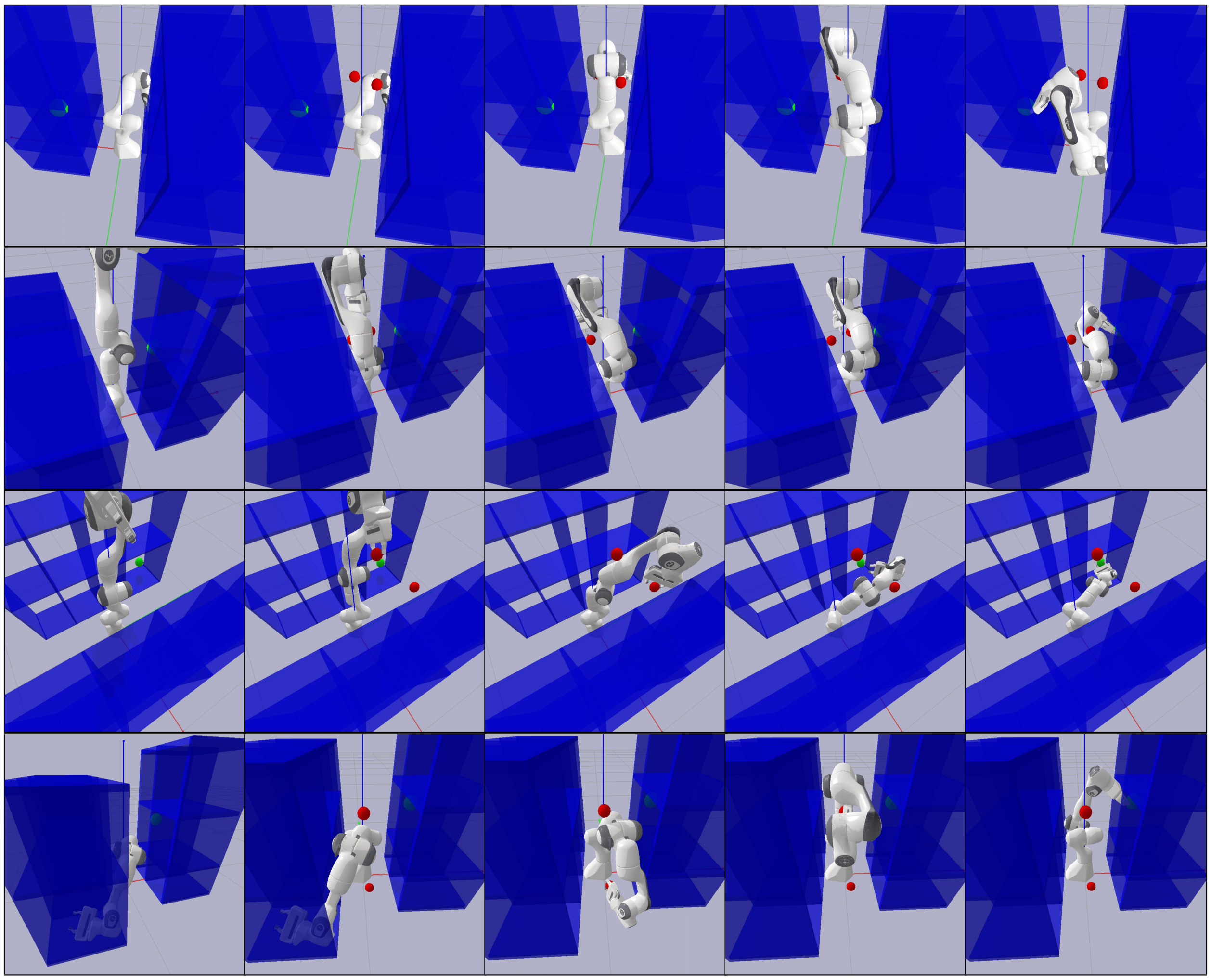


Figure 9.1: A manipulator reaching from a cupboard to the goal (green sphere) in another. The environment is tightly-packed, as the cupboards sandwich the manipulator. Previous local reactive methods fall into local minima, while our approach finds a sequence of *commands* to produce reactive motion to the goal. The end-effector coordinates of the command states are shown in red spheres.

9.2 Related Work

Reactive Motion Generation

Our method is built atop reactive methods for motion generation. Reactive methods take into account the local environment and the current robot state to compute the optimal immediate next control. Methods in this category include Khatib (1985); Ijspeert et al. (2013); Ratliff et al. (2018); Van Wyk et al. (2022). These approaches are able to react to changes in the environment, perturbations, and sensor uncertainty, as the instantaneous best control is found at every timestep. However, these methods can often get stuck in local minima when the workspace geometry is non-convex.

Probabilistic Complete Motion Planning:

Probabilistic Complete Motion Planning lies on the other end of the spectrum, where planners aim to find a path (represented as waypoints) from start to goal configuration, by iteratively sampling and maintaining a graph structure. As the number of sample points approaches infinity, the probability of finding a path, if it exists, tends to one. Representative approaches in this category include PRMs (Kavraki et al., 1996), RRTs (LaValle and James J. Kuffner, 2001), as well as subsequent variants such as RRT* (Karaman and Frazzoli, 2011), Bi-RRT (Jordan and Perez, 2013). Like our proposed method, these motion planners aim to find a globally optimal solution rather than immediate optimal controls. However, methods in this category cannot reactively adapt to changes, and often produce non-smooth, non-intuitive motion with drastic swings.

Learning to Plan

Learning approaches to aid planning have recently gained popularity. Machine learning models can be used to memorise solutions of planning, by training over a large dataset containing problem setups, and with corresponding labelled solutions. During inference, the learning model can speed up or improve planning by suggesting promising candidates to use in the downstream planning algorithm. Motion planning networks (Qureshi et al., 2021) condition on the problem to iteratively predict deterministic waypoints. Further work in trajectory or sample generation take a probabilistic approach and draw samples from mixture (Zhi et al., 2021a), normalising flow (Lai et al., 2021), or variational autoencoder models (Ichter et al., 2018). Similarly, our method also applies learning to aid motion generation. However, we tackle this in a self-supervised learning manner, where learning and motion generation is combined into the same loop. An implicit generative model capable of flexibly modelling unnormalised multi-modal distributions is applied.

9.3 Preliminaries: Geometric Fabrics

Geometric Fabrics (GFs) are the central building block of our method. Our work does not alter the machinery of GFs, but uses GFs as building blocks. For completeness, here we give an intuitive introduction to GFs, with further details and theoretical analysis available in Van Wyk et al. (2022).

Fabric terms

Geometric fabrics (GFs) modularise robot motion by representing each individual local behaviour as position and velocity-dependent second order dynamical systems, referred to as *fabric terms*. Examples of these local behaviour include: accelerating towards a specified goal; reactively avoiding an obstacle by getting "repelled" away; avoiding the robot joint limits. Each of these behaviours are defined independently, and may be defined in different task spaces. For example, obstacle avoidance is defined with respect to body points on the robot, while joint limits are defined in the configuration space. These fabric terms can then be fused into a single system. We denote the position in some task space as $\mathbf{x}$, its velocity and acceleration as $\dot{\mathbf{x}}$, $\ddot{\mathbf{x}}$. We denote a configuration belonging to a d-dimensional C-space $\mathcal{Q} \subseteq \mathbb{R}^d$ as $\mathbf{q} \in \mathcal{Q}$, its time derivatives as $\dot{\mathbf{q}} \in \mathbb{R}^d$, $\ddot{\mathbf{q}} \in \mathbb{R}^d$. Formally, each fabric term is defined by a pair $(\mathcal{L}_e, \pi)$, where $\mathcal{L}_e(\mathbf{x}, \dot{\mathbf{x}}) \rightarrow \mathbb{R}$ is a *Finsler energy* (Ratliff et al., 2021), and $\ddot{\mathbf{x}} = \pi(\mathbf{x}, \dot{\mathbf{x}})$ is a policy. $\mathcal{L}_e$ satisfies: (1) $\mathcal{L}_e(\mathbf{x}, \dot{\mathbf{x}}) \geq 0$ with equality only when $\dot{\mathbf{x}} = 0$; (2) $\mathcal{L}_e(\mathbf{x}, \lambda\dot{\mathbf{x}}) = \lambda^2 \mathcal{L}_e(\mathbf{x}, \dot{\mathbf{x}})$, for $\lambda \geq 0$; (3) $\mathbf{M} = \partial^2_{\dot{\mathbf{x}}\dot{\mathbf{x}}}$ is positive definite, and π satisfies $\pi(\mathbf{x}, \lambda\dot{\mathbf{x}}) = \lambda^2 \pi(\mathbf{x}, \dot{\mathbf{x}})$ for $\lambda \geq 0$. Here, the policy π defines time-independent paths, an operation $\texttt{Energize}_{\mathcal{L}_e}(\pi)$ is then used to select an execution velocity, and $\mathbf{M}$ is a priority metric.

Fusion

Fabric terms defined in different task spaces can be fused, by first finding the configuration space coordinates of each term, in a *transform tree* (Cheng et al., 2018) manner. Suppose, from each fabric term, we have a set of N_{ft} triples, $\{(\phi_i, \mathbf{M}_i, \ddot{\mathbf{x}}_i)\}_{i=1}^{N_{ft}}$, where ϕ_i maps a $\mathbf{q}$ to task space coordinates $\mathbf{x}_i$. The combined acceleration, $\ddot{\mathbf{q}}$, is given by:

$$\ddot{\mathbf{q}} = \Big(\sum_{i=1}^{N_{ft}} \mathbf{J}_i^\top \mathbf{M}_i \mathbf{J}_i\Big)^{-1} \Big(\sum_{i=1}^{N_{ft}} \mathbf{J}_i^\top \mathbf{M}_i (\ddot{\mathbf{x}}_i - \dot{\mathbf{J}}_i \dot{\mathbf{q}})\Big), \tag{9.1}$$

where $\mathbf{J}_i$ is the Jacobian of ϕ_i, $\dot{\mathbf{J}}_i$ its time derivative.

9.4 Self-supervised Learning of Geometric Fabric Command Sequences

We shall first formulate motion generation as a global optimisation over a sequence of *commands* (section 9.4.1). Then, we develop a generative model to *learn* to optimise (section 9.4.2), and a self-supervised learning framework that encompasses both optimisation and learning (section 9.4.3).

9.4.1 Geometric Fabrics Command Optimisation

Let us consider how Geometric Fabrics can be used to generate locally optimal motion trajectories. A combined Geometric Fabric, $\ddot{\mathbf{q}} = f(\mathbf{q}, \dot{\mathbf{q}}, |\mathbf{q}_0, \dot{\mathbf{q}}_0, \mathbf{q}_g, E, C)$, can be viewed as a policy that maps from $\mathbf{q}$ and $\dot{\mathbf{q}}$ to $\ddot{\mathbf{q}}$. It is also dependent on problem-specific parameters: the start configuration $\mathbf{q}_0 \in \mathcal{Q}$ and velocity $\dot{\mathbf{q}}_0 \in \mathbb{R}^d$, goal $\mathbf{q}_g \in \mathcal{Q}$, environment E, and a set of robot-specific configuration parameters C. Trajectories can be obtained by a double integrator:

$$\begin{bmatrix}\dot{\mathbf{q}}_t \\ \mathbf{q}_t\end{bmatrix} = \int_0^t \begin{bmatrix} f(\mathbf{q}_u, \dot{\mathbf{q}}_u, |\mathbf{q}_0, \dot{\mathbf{q}}_0, \mathbf{q}_g, E, C) \\ \dot{\mathbf{q}}_u \end{bmatrix} \mathrm{d}u + \begin{bmatrix}\dot{\mathbf{q}}_0 \\ \mathbf{q}_0\end{bmatrix}. \tag{9.2}$$

We define $\texttt{rollout}_\texttt{t}$ as an operation that finds the time steps used, $t^* \in \mathbb{R}^+$, to roll-out near the goal,

$$t^* = \texttt{rollout}_\texttt{t}(\mathbf{q}_0, \dot{\mathbf{q}}_0, \mathbf{q}_g, E, C) := \min\{\min(t), t_{max}\}, \qquad \text{s.t. } ||\mathbf{q}_t - \mathbf{q}_g||_2 < \epsilon, \tag{9.3}$$

where ϵ denotes some tolerance. In practice $\mathbf{q}_t$ is obtained via numerical integration, such as Euler's method, of eq. (9.2), with a time step budget of $t_{max} \in \mathbb{R}^+$. Instead of defining a single goal, we inject global behaviour into the policy by stringing together a sequence of attraction states, or commands $\mathbf{q}_g^i$, while sharing local collision-avoidance and joint-limit handling fabric terms. We define the Geometric Fabrics Command Optimisation Problem (GFCOP), where we have a sequence of $n+1$ commands with different goal configurations, and optimise over the goals of the first n commands. The final command is specified as the final global goal. Specifically, we define:

$$[\text{GFCOP}] : \min_{\mathbf{q}_g^1 \dots \mathbf{q}_g^n} \sum_{i=1}^{n+1} t_i^* + M\mathbb{1} \tag{9.4}$$

$$\text{s.t. } t_i^* = \texttt{rollout}_{\texttt{t}}(\mathbf{q}_0^i, \dot{\mathbf{q}}_0^i, \mathbf{q}_g^i, E, C), \text{ for } i = 1, \dots, n+1 \tag{9.5}$$

$$\mathbf{q}_0^1 = \mathbf{q}_0,\ \dot{\mathbf{q}}_0^1 = \dot{\mathbf{q}}_0,\ \mathbf{q}_g^n = \mathbf{q}_g, \qquad \text{(Boundary conditions)} \tag{9.6}$$

$$\mathbf{q}_0^i = \mathbf{q}_{t_{i-1}^*}^{i-1},\ \dot{\mathbf{q}}_0^i = \dot{\mathbf{q}}_{t_{i-1}^*}^{i-1}, \quad \text{for } i = 2, \dots, n+1, \qquad \text{(Intermediate continuity)} \tag{9.7}$$

$$\mathbb{1} = \{0, \text{ if } t_{n+1}^* < t_{max}; 1, \text{ otherwise.}\} \qquad \text{(Penalty Indicator)} \tag{9.8}$$

where the cost (eq. (9.4)) is defined as the sum of the integration times exhausted until we reach our goal. Note that superscripts indicate the command index, while the subscript specifies the integration steps, i.e. $\mathbf{q}_0^i$ indicates the initial configuration at the i^{th} command. If the roll-out towards the final command in the sequence cannot reach our goal, then a large penalty of $M \in \mathbb{R}$ is applied. This is controlled by the indicator $\mathbb{1} \in \{0, 1\}$, as defined in eq. (9.8). Equation (9.6) designates that the start configuration and its velocity, match given initial conditions $\mathbf{q}_0$ and $\dot{\mathbf{q}}_0$, and the final command is also the specified global goal, $\mathbf{q}_g$. Equation (9.7) enforces continuity, specifying that the initial conditions when integrating towards a command, shall be the terminating configuration and velocity of that of the previous. Constraints eqs. (9.5) to (9.7) are satisfied by running the roll-outs in a sequential manner, using the terminal conditions of the former integration as the initial conditions of the next.

The underlying local Geometric Fabric, given by f in eq. (9.2), performs local obstacle avoidance, self-collision avoidance, joint coordinate and velocity limit handling. This allows the GFCOP to be concisely defined. Additionally, as local avoidance fabric terms are repulsion-based, motion slows down as the robot approaches obstacles, resulting in larger $\texttt{rollout}_{\texttt{t}}$ values. Therefore, solutions of GFCOP tend to smoothly move around obstacles, and not scrape past them.

The defined GFCOP is discontinuous and non-convex. We can apply the black-box optimiser *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) (Hansen, 2016), which iteratively updates a multivariate Gaussian (MVG) sampling distribution over our command decision variables, $\mathbf{q}_g^1 \ldots \mathbf{q}_g^n$, after ranking the computed cost of previously evaluated candidates. The probability of CMA-ES drawing samples over the entire domain is non-zero, and has been shown to efficiently find globally optimal solutions (Hansen and Kern, 2004; Hansen et al., 2010). We denote each solution, $\mathbf{y}_i = [{\mathbf{q}_g^1}^\top, \ldots, {\mathbf{q}_g^n}^\top]_i^\top$, as a sequentially concatenated vector of the command goals. In each iteration, we sample a batch of candidates from the MVG, compute the cost of the candidates in parallel, and then update the sampling distribution. Details of CMA-ES can be found in the tutorial Hansen (2016). Although CMA-ES finds globally optimal solutions, under a fixed time budget, the quality and speed of the optimisation is impacted by the initial batch of samples. Vanilla CMA-ES simply initialises a MVG based on random guesses of the mean and covariance, or simply sample uniformly. This motivates us to imbue prior knowledge into the optimisation and warm-start (Nomura et al., 2021) the process by learning to sample.

Algorithm 4: `SolveGFCOP` with warm-started CMA-ES

```
input   : $\mathbf{q}_0$,$\dot{\mathbf{q}}_0$, $\mathbf{q}_g$, $E$, $f_{sampler}$, $C$, number of commands in sequence $n+1$,
          iterations $N_{iters}$, samples batch-size $N_{samp}$, $M$ large penalty for failure.
$\mathbf{y}_1, \ldots, \mathbf{y}_{N_{samp}} \sim f_{sampler}$ ;   // Draw candidate solutions from $f_{sampler}$ to warm-start
$AllCandidatesWithCosts \leftarrow \{\}$ ;   // Empty set for all solutions with costs
for ( $j = 1; j <= N_{iters}; j = j + 1$ ) {
    $BatchCandidatesWithCosts \leftarrow \{\}$ ;   // Empty set for solutions with costs
    ▷ Compute costs for each candidate in parallel;
    Do in Parallel for each ( $\mathbf{y}$ in $[\mathbf{y}_1, \ldots, \mathbf{y}_{N_{samp}}]$ ) {
        $\mathbf{q}_g^1, \ldots, \dot{\mathbf{q}}_g^n \leftarrow \mathbf{y}$ ;   // Unpack y to get candidate solution goals
        $\mathbf{q}_g^{n+1} \leftarrow \mathbf{q}_g$ ;   // Specify the command goal as the global goal
        $\mathbf{q}_0^1 \leftarrow \mathbf{q}_0$, $\dot{\mathbf{q}}_0^1 \leftarrow \dot{\mathbf{q}}_0$ ;   // Set Initial configuration and velocity
        $Cost \leftarrow 0$;   // Initialise the cost of a candidate solution
        for ( $i = 1; i <= n + 1; i = i + 1$ ) {
            $t_i^* \leftarrow \mathtt{rollout_t}(\mathbf{q}_0^i, \dot{\mathbf{q}}_0^i, \mathbf{q}_g^i, E, C)$ ;   // Roll-out the $i^{th}$ command in sequence
            $Cost \leftarrow Cost + t_i^*$
        }
        if $t^* < t_{max}$ then $\mathbb{1} \leftarrow 0$ else $\mathbb{1} \leftarrow 1$;
        $Cost \leftarrow Cost + M\mathbb{1}$ ;   // If we don't reach before time-out, apply penalty
        $BatchCandidatesWithCosts$.insert({$\mathbf{y}$, Cost});
        $AllCandidatesWithCosts$.insert({$\mathbf{y}$, Cost});
    }
    ▷ Update CMA-ES and sample the next batch of candidates;
    CMA-ES.update($BatchCandidatesWithCosts$) ;   // Update CME-ES with new costs
    $\mathbf{y}_1, \ldots, \mathbf{y}_{N_{samp}} \sim$ CMA-ES.sampler();   // Sample next batch with CMA-ES sampler
}
$top\text{-}m \leftarrow$ GetTopMByCost($AllCandidatesWithCosts$) ;   // Sort by cost and get lowest $m$ candidates
output  : $top\text{-}m$
```

9.4.2 Learning to Sample for Optimisation

We postulate that the optimal motions across problem setups with similar start, $\mathbf{q}_0$ $\dot{\mathbf{q}}_0$, goal, $\mathbf{q}_g$, and environments, E, are similar, and that we can *learn* to generate promising candidates from the optimisation solutions in alternative problem setups. We can use the generated candidates to warm-start CMA-ES, or directly use the candidate solution with the lowest cost as an approximate solution. Next, we define operations needed for the training, and outline the training, sampling and encoding used to construct the generative model.

Solutions of GFCOP

Let us define the operation $\texttt{SolveGFCOP}(\mathbf{q}_0, \dot{\mathbf{q}}_0, \mathbf{q}_g, E|f_{sampler})$ to solve a GFCOP problem for a specific initial configuration, velocity, goal, environment, and return the top-m candidate solutions. A detailed algorithm is shown in algorithm 4. A sampler $f_{sampler}$ is passed on to provide the initial samples, and warm-start CMA-ES. We keep track of candidate solutions and their cost. For each call of `SolveGFCOP`, we obtain a set of $\{\mathbf{y}_i\}_{i=1}^{m}$ of m solutions, where $\mathbf{y}_i = [\mathbf{q}_g^{1\top}, \ldots, \mathbf{q}_g^{n\top}]_i^\top$, is the sequentially concatenated vector of the command goals. Note that the GFCOP is also dependent on other inputs, such as the length of the sequence of commands, $n+1$, as well as the robot-specific parameters, C. However, we shall drop the explicit dependence on these, as they are held constant during learning, i.e. learning is conducted over command sequences of the same length and on the same robot.

Training the Generative Model

Implicit energy-based models (EBMs) (Florence et al., 2021) excel at learning complex *unnormalised* distributions, where only sampling and not density estimation is performed using the model. Here, we take an EBM approach and learn an energy function $E_\theta(\mathbf{x}^c, \mathbf{y}) \to \mathbb{R}$, mapping from a (context, target) pair to an energy value, by contrasting positive and negative samples, where positive samples have high energy and negative samples have low energy values. In our setup, the context is an encoding of the problem, while the target is the optimised command sequence. Suppose we have a dataset of positive examples $D^{pos} = \{\mathbf{x}_j^c, \mathbf{y}_j^*\}_{j=1}^{N_{pos}}$. Each $\mathbf{x}_j^c$ encodes the inputs to `SolveGFCOP`, and each $\mathbf{y}_j^*$ is outputted from `SolveGFCOP`. For each positive example $\mathbf{y}_j^*$, we actively generate N_{neg} negative examples $\{\widehat{\mathbf{y}}_j^i\}_{i=1}^{N_{neg}}$. Then, we can construct a InfoNCE-like loss (Florence et al., 2021; van den Oord et al., 2018):

$$\mathcal{L}_{\text{InfoNCE}} = \sum_{j=1}^{N_{pos}} -\log\Big\{\big(e^{-E_\theta(\mathbf{x}_j^c,\mathbf{y}_j^*)}\big)\big(e^{-E_\theta(\mathbf{x}_j^c,\mathbf{y}_j^*)} + \sum_{k=1}^{N_{neg}} e^{-E_\theta(\mathbf{x}_j^c,\hat{\mathbf{y}}_j^k)}\big)^{-1}\Big\}. \tag{9.9}$$

The energy function E_θ is modelled by a neural network with parameters θ, which can be trained via stochastic gradient descent and its variants.

Generating Samples with Stochastic Gradient Langevin Dynamics

Stochastic gradient Langevin dynamics (SGLD) (Welling and Teh, 2011) is a technique to obtain a full posterior distribution from some energy function E_θ, rather than a single maximum *a posteriori* mode. This allows us to draw samples from the energy function to: (1) provide challenging negative examples to contrast against (Du and Mordatch, 2019); (2) warm-start CMA-ES. Given a trained energy function E_θ, and an context encoding $\mathbf{x}^c$, we randomly initialise n_p particles $\mathbf{y}_1^0, \ldots, \mathbf{y}_{n_p}^0 \sim \mathcal{U}(\mathbf{q}^{upper}, \mathbf{q}^{lower})$ from a uniform distribution, where $\mathbf{q}^{upper}, \mathbf{q}^{lower}$ are the upper and lower joint limits. The particles are updated according to the rule:

$$\Delta\mathbf{y}_i^{k+1} = \mathbf{y}_i^k + \frac{\sigma_k^2}{2}(\nabla_{\mathbf{y}_i} E_\theta(\mathbf{x}^c, \mathbf{y}_i^k)) + \eta^k, \quad \text{where } \eta \sim \mathcal{N}(0, \sigma_k^2), \quad \text{for } i = 0, \ldots, n_p. \tag{9.10}$$

Additionally, we decrease σ^2 according to the scheduler $\sigma_k^2 = (1+k)^{-0.5}$. After a maximum number of gradient ascent iterations has been exhausted, we take the updated particles and either select them as negative examples during training, or use them as solutions during inference.

Encoding a GFCOP

The EBM requires a fixed length context vector, $\mathbf{x}^c$, to condition on. The initial configuration, its velocity, and the goal are all fixed in size. We aim to find a fixed-sized encoding for the environment E. Here, we assume that each environment is represented by a point set $\mathcal{S} = \{\mathbf{s}_i\}_{i=1}^{N_E}$ of size N_E, where $\mathbf{s}_i \in \mathbb{R}^3$ are coordinate points in Euclidean space. This represents surfaces of objects, and can be obtained from an RGBD camera. The size of the point set N_E varies with each environment. We take a reference point-set approach (Yang et al., 2019) and compute the minimum Euclidean distance between the point set, and a set of fixed N_R reference coordinates $\widehat{\mathcal{S}} = \{\widehat{\mathbf{s}}_i\}_{i=1}^{N_R}$

$$\mathbf{d}_E(\mathcal{S}, \widehat{\mathcal{S}}) := \big[\min_{\mathbf{s}\in\mathcal{S}} ||\mathbf{s} - \widehat{\mathbf{s}}_1||_2,\ \min_{\mathbf{s}\in\mathcal{S}} ||\mathbf{s} - \widehat{\mathbf{s}}_2||_2,\ \ldots,\ \min_{\mathbf{s}\in\mathcal{S}} ||\mathbf{s} - \widehat{\mathbf{s}}_{N_R}||_2\big]. \tag{9.11}$$

This provides us a fixed length (of length N_R) vector representation for environments with point sets of different sizes. The reference points are laid out in a lattice, with equal distance. If the size of the fixed length vector is excessively large, its dimensions are further reduced using an autoencoder, to obtain encoding $\widehat{\mathbf{d}}_E$. We concatenate $\mathbf{q}_0$, $\dot{\mathbf{q}}_0$, $\mathbf{q}_g$, and $\widehat{\mathbf{d}}_E$ to produce the input context $\mathbf{x}^c$.

9.4.3 Self-supervision: Optimisation and Learning in the Loop

A key insight is that the optimisation outlined in section 9.4.1 and the learning outlined in section 9.4.2 are complimentary. Specifically, as more GFCOPs are solved, the more self-labelled training data is available for our implicit EBM. The EBM is trained by eq. (9.9), with negative samples generated from SGLD. On the other hand, as our EBM is progressively more well-trained, the quality of its generated candidate solutions also improve. This section brings both the optimisation and learning into a unified loop, and outlines a self-supervised learning framework. Following the definition in Nava et al. (2019), in this context, self-supervision refers to the absence of external supervision signal (i.e. no human labelling) (Nava et al., 2019). Algorithm 5 outlines the self-supervised GFCS learning framework. We are assumed to have a set of N_{prob} motion generation problems, $\{\mathbf{q}_0^n, \dot{\mathbf{q}}_0^n, \mathbf{q}_g^n, E^n\}_{n=1}^{N_{prob}}$. To self-generate data, we iteratively add solutions of `SolveGFCOP` to a buffer D^{buffer} of some maximum length; we train our EBM, E_θ, on the data in the buffer, after every batch, of size n_{train}, of problems have been solved; E_θ is then used as the sampler $f_{sampler}$ for future calls of `SolveGFCOP`.

Algorithm 5: Self-supervised Geometric Fabric Command Sequence Learning Framework

```
input  : A set of $N_{prob}$ motion generation problems $\{\mathbf{q}_0^n, \dot{\mathbf{q}}_0^n, \mathbf{q}_g^n, E^n\}_{n=1}^{N_{prob}}$; an
         untrained implicit model $E_\theta(\mathbf{x}, \mathbf{y}) \rightarrow \mathbb{R}$, $N_{neg}$, $\mathbf{q}^{upper}$, $\mathbf{q}^{lower}$, $\mathcal{D}^{buffer}$.
$f_{sampler} \leftarrow \mathcal{U}(\mathbf{q}^{upper}, \mathbf{q}^{lower})$ ;            // Initialise the sampler to be uniform
for ( $n = 1$; $n <= N_{prob}$; $n = n + 1$ ) {
    ▷ Self-generate positive examples via optimisation
    $\mathcal{D}^{pos} \leftarrow$ SolveGFCOP($\mathbf{q}_0^n, \dot{\mathbf{q}}_0^n, \mathbf{q}_g^n, E^n | f_{sampler}$) ;  // Solve for top-$m$ solutions
    $\mathcal{D}^{buffer}$.insert($\mathcal{D}^{pos}$) ;                       // Add positive data into buffer
    $\mathcal{D}^{buffer} \leftarrow$ MaintainBuffer($\mathcal{D}^{buffer}$) ;    // Maintain the length of buffer
    $\mathcal{D}^{neg} \leftarrow \{\}$ ;                          // Initialise empty set for negatives
    for ( $j = 1$; $j <=$ length($\mathcal{D}^{buffer}$); $j = j + 1$ ) {
        $\mathbf{y}_1^0, \ldots, \mathbf{y}_{N_{neg}}^0 \sim \mathcal{U}(\mathbf{q}^{upper}, \mathbf{q}^{lower})$;
        $\{\widehat{\mathbf{y}}_j^i\}_{i=1}^{N_{neg}} \leftarrow$ SGLD($\mathbf{y}_1^0, \ldots, \mathbf{y}_{N_{neg}}^0 | E_\theta$) ; // Run SGLD using eq. (9.10) on $E_\theta$
        $\mathcal{D}^{neg}$.insert($\{\widehat{\mathbf{y}}_j^i\}_{i=1}^{N_{neg}}$) ;              // Insert into set of negatives
    }
    $\mathbf{x} \leftarrow$ encode($\mathbf{q}_0^n, \dot{\mathbf{q}}_0^n, \mathbf{q}_g^n, E^n$) ;                        // Encode problem.
    ▷ Train on self-generated data, and update the optimisation sampler
    if $n \% n_{train} == 0$ then
        $E_\theta$.train($\mathcal{D}^{buffer}, \mathcal{D}^{neg}$) ;            // train $E_\theta$ with eq. (9.9), after
         $n_{train} solves$
    end
    $f_{sampler} \leftarrow$ SGLD($\mathbf{y}_1^0, \mathbf{y}_2^0 \ldots | E_\theta$), where $\mathbf{y}_1^0, \mathbf{y}_2^0 \ldots \sim \mathcal{U}(\mathbf{q}^{upper}, \mathbf{q}^{lower})$ ;  // Update
     the sampler as SGLD to the trained $E_\theta$
}
output: Trained $E_\theta$
```

9.5 Experimental Evaluation

We empirically investigate the performance of self-supervised GFCS to reliably generate global and reactive motion in various complex environments, both on a simulation FRANKA EMIKA Panda and on a real-world JACO manipulator. We evaluate 6 different classes of benchmark problems from Chamzas et al. (2022), each class of problems contains 100 environments with different start/goals. Example problems are illustrated in fig. 9.2. For each problem class, we use 50 problems for self-supervised training, and report our evaluation on the remaining 50 test problems. We report on the following metrics:

- *% Success*: the percentage of problems a solution is found; *Times*: the time taken to generate the trajectories;
- *EE length*: the length of the end-effector trajectory; *C-space length*: the length of the C-space trajectory;
- *EE-LDJ*: The log-dimensionless jerk (LDJ) (Hogan and Sternad, 2009) of the end-effector trajectory;
- *C-space LDJ*: The LDJ (Hogan and Sternad, 2009) of the C-space trajectory.

EE length reflects trajectory legibility, with a low length indicating that little superfluous motion is present. LDJ reflects the trajectory smoothness, where closer to zero indicates a smoother trajectory.

9.5.1 Experimental Setup Details

In our experimental evaluations of trajectory quality, we use we evaluate 6 problem classes of benchmark problems. For each class, 50 problems are used within the self-supervised training loop, and we evaluate and report our results on the remaining 50. In total, 300 problems are used for self-supervised learning, and 300 used in evaluated results.

When rolling out trajectories for Geometric Fabric Command Sequences, we set the time budget per command, t_{max}, to 7s, and integrate with step-size 0.01. The Geometric Fabric terms for attractors, joint limit avoidance, and collision-avoidance follow the original implementation from the authors of Geometric Fabrics (Van Wyk et al., 2022). Over the 6 benchmark problems, we use two commands for every problem. During self supervised training, for each problem, we sample and optimise for 30 batches with 30 candidate samples per batch, while during test time, we draw 20 candidate samples per batch. We iterate through all the training problems twice, and add the top-4 solutions into the buffer for training, and the buffer is maintained to have 200 solutions. Training on solutions occurs after every 25 problems have been solved, that is $n_{train} = 25$.

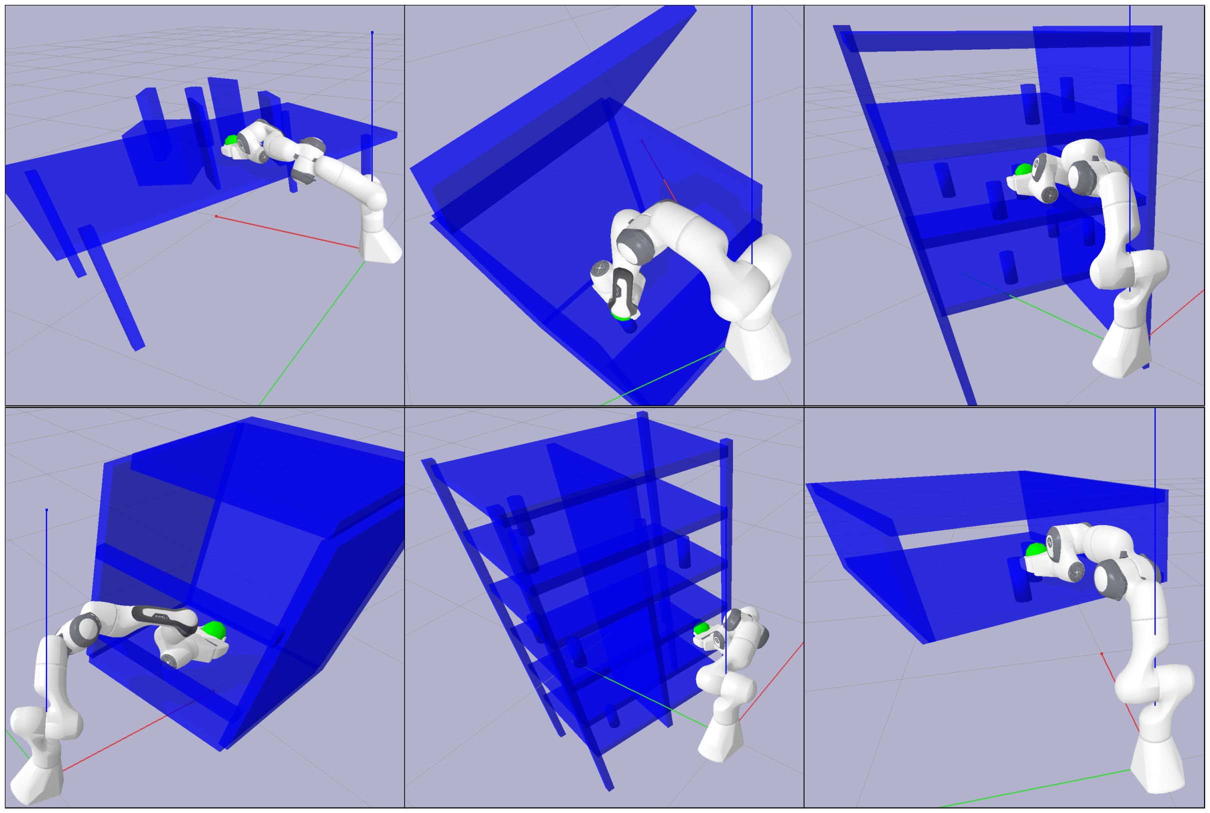


Figure 9.2: Examples from the different problem classes (Table under, Box, Tall shelf, Cage, Thin shelf, Single shelf) from Chamzas et al. (2022), goals shown in green.

During training, to encode the environment, we lay out fixed reference point in intervals of 0.15. After obtaining an encoding by projecting the obstacle points to the reference points, we further lower the dimensions of the encoding to size 300, using an auto-encoder. We then concatenate the environment encoding with the start and goal configurations along with the input commands to produce an input vector $\mathbf{x}^c$. This is passed into a neural network of with layers:
$Dense(input\_size, 800) \rightarrow ReLU() \rightarrow Dense(800, 800) \rightarrow ReLU() \rightarrow Dense(800, 800) \rightarrow ReLU() \rightarrow Dense(800, 1)$.

We generate negative samples by running SLGD on 200 particles, for 50 iterations with $\epsilon^2 = 0.2$. For each time we train the generative model, training is run for 800 iterations with an ADAM optimiser with a step-size of 0.0001. When we generate new obstacles to evaluate the reactive-ness of our method, obstacles positions are sampled randomly from the solved trajectory coordinates which we solve without the additional obstacles. Additionally, we require that the addition of the obstacles do not cause the initial robot configuration to be in collision, and is not in the trajectory portion corresponding to the last second of the original solution.

9.5.2 Global Solutions and Motion Trajectory Quality

We hypothesis that our method, like local motion generation approaches, is able to generate legible motion trajectories, without superfluous end-effector motion, while capable of handling non-convexities in the task space to generate globally feasible motion. We observe that a GFCS that is very short is sufficient to generate complex non-local behaviour. We compare GFCS, of length two, against:

1. *Geometric Fabrics (GFs)*, a reactive method capable of producing high quality motion trajectories;
2. Bi-directional Expansive Space Trees (Bi-EST) (Hsu et al., 1997), a bi-direction sampling-based motion planner;
3. Batch Informed Trees (BBT*) (Gammell et al., 2020), an asymptotically optimal sampling-based planner.

We use the Bi-EST and BIT* implementations in the Open Motion Planning Library (OMPL) (Şucan et al., 2012). As our method can continuously optimise and refine the solution, we evaluate our method after the first and fifth batches of samples have been evaluated. Likewise, BIT* is asymptotically optimal and will continuously refine the solution found. We evaluate BIT* with a budget of 1.2 seconds and 6 seconds, which is slightly longer than the times our method takes to evaluate the first and fifth batch of samples.

We tabulate the results over the six classes of benchmark environments in table 9.1. We observe that the local geometric fabrics approach fails to solve many of the problems, particularly in the "table under" class of problems, where the manipulator is tasked with starting from under a table and reaching a pose on a table. Completing such a task requires the manipulator to exhibit global behaviour and escape from a local minimum. On the other hand, GFCS is able to generate motions which escape from local minima. Bi-EST can typically find feasible solutions quickly, but produce undesirable trajectories, as measured by our metrics of trajectory quality. We also observe that the "cage" problems require the manipulator to traverse a narrow passage, through the bars of a cage, to reach inside.

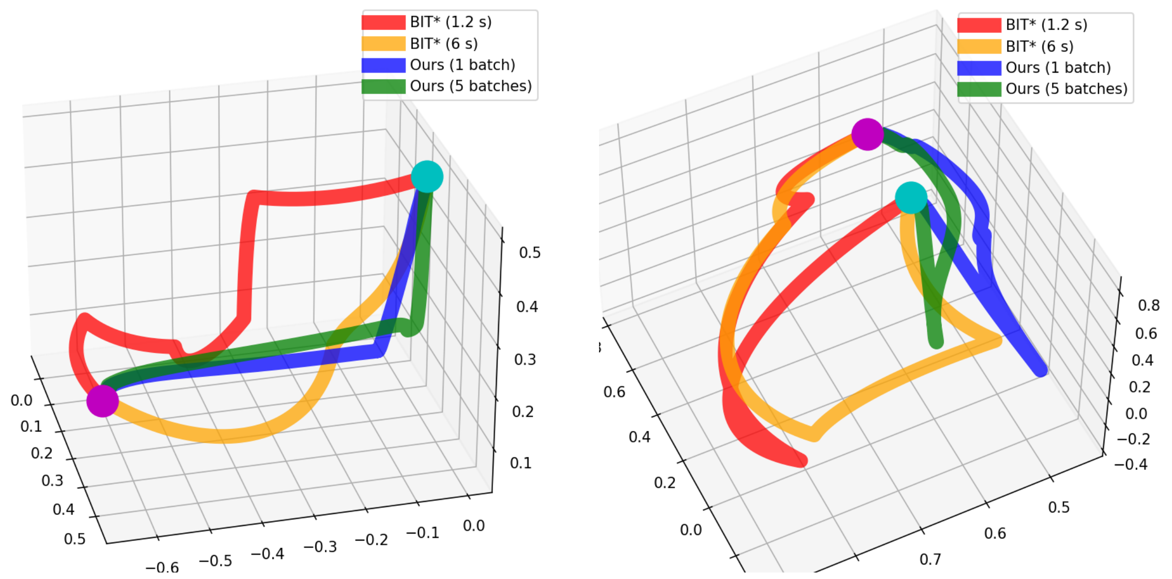


Figure 9.3: Examples of end-effector trajectories, from start (cyan) to goal (magenta), in "Tall shelf" (Left) and "Table under" (Right). GFCS consistently produces shorter and smoother motions. BIT*(1.2s) given in red, BIT*(6s) in yellow, ours (1 batch) in blue, and ours (5 batch) in green.

Sampling-based motion planning methods struggle at this: Bi-ESTs become inefficient and are slower than local approaches to find a feasible trajectory, while BIT* simply cannot find a feasible solution for most of the "cage" problems. Our method is able to generate high-quality trajectories. GFs and GFCS consistently produce trajectories with shorter and smoother end-effector motions than sampling-based planners. End-effector trajectories in two example problems are shown in fig. 9.3.

9.5.3 The Effect of Learning

Our implicit generative model learns to produce good samples for GFCS to optimise over. Here, we investigate the performance improvements provided by our learnt model. We observe that the learned generative model is able to suggest high quality candidate to the global optimiser. We investigate the performance of our learnt generative model compared against drawing from a uniform distribution, on "table under" problems, which particularly requires global solutions. The results in Table 9.2 show that using the learned sampler gives rises to more successful solutions, and higher quality motion trajectories.

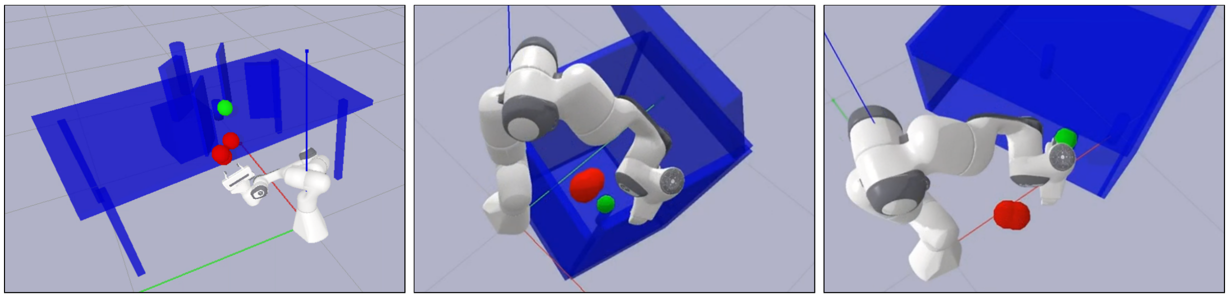


Figure 9.4: After obtaining a solution to the goal (in green), we add new obstacles (in red), GFCS reactively avoid the new obstacles without re-optimising.

9.5.4 Global and Reactive Motion Generation

Geometric Fabric Sequences inherit reactive behaviour from Geometric Fabrics while finding global solutions. Specifically, after commands in the GFCS have been optimised, upon encountering previously unseen obstacles, local collision avoidance fabric terms will execute local avoidance. We randomly generate new obstacles, blocking the previously solved motion trajectory, and check whether the motion can instantaneously adapt to the new obstacles. Examples are illustrated in fig. 9.4, where the red obstacles, after optimisation, are added in the way of the manipulator. We observe that GFCS can consistently generate collision-free trajectories which reactively avoid new obstacles, while escaping local minima.

Table 9.1: Evaluated metrics (mean ± std) for motion generated in multiple benchmark environments

(a) Evaluation in the "Table Under" problems

	% Success	EE length	C-space length	Times (s)	EE-LDJ	C-space-LDJ
GF	0	NA	NA	NA	NA	NA
Bi-EST	100	8.24 ± 3.2	23.69 ± 9.2	0.028 ± 0.04	-11.09 ± 1.7	-10.58 ± 1.3
BIT* (1.2s)	100	3.93 ± 0.9	7.71 ± 2.8	NA	-7.86 ± 1.3	-7.34 ± 0.9
BIT* (6s)	100	3.71 ± 1.0	7.12 ± 2.8	NA	-7.95 ± 1.0	-7.13 ± 1.1
Ours (1 batch)	96	1.43 ± 0.5	9.35 ± 2.4	0.96 ± 0.05	-7.45 ± 1.8	-7.51 ± 2.0
Ours (5 batches)	98	1.32 ± 0.3	8.65 ± 2.0	5.29 ± 0.4	-7.05 ± 1.6	-7.09 ± 1.8

(b) Evaluation in the "Box" problems

	% Success	EE length	C-space length	Times (s)	EE-LDJ	C-space-LDJ
GF	100	0.92 ± 0.07	5.25 ± 0.6	0.066 ± 0.01	-5.32 ± 0.9	-5.57 ± 1.5
Bi-EST	100	5.52 ± 3.3	19.68 ± 9.3	0.030 ± 0.03	-10.55 ± 1.8	-10.36 ± 1.3
BIT* (1.2s)	92	1.97 ± 0.7	8.21 ± 2.2	NA	-8.99 ± 1.1	-8.30 ± 0.8
BIT* (6s)	100	1.84 ± 0.6	7.39 ± 2.3	NA	-8.69 ± 1.0	-7.99 ± 1.0
Ours (1 batch)	100	1.00 ± 0.1	5.43 ± 0.8	0.74 ± 0.03	-5.84 ± 1.1	-5.55 ± 1.5
Ours (5 batches)	100	0.97 ± 0.1	5.23 ± 0.6	4.01 ± 0.2	-5.89 ± 1.6	-5.40 ± 1.6

(c) Evaluation in the "Tall shelf" problems

	% Success	EE length	C-space length	Times (s)	EE-LDJ	C-space-LDJ
GF	88	0.700 ± 0.2	6.51 ± 1.1	0.1 ± 0.03	-4.97 ± 1.3	-6.03 ± 1.9
Bi-EST	100	4.57 ± 2.6	14.01 ± 5.8	0.036 ± 0.08	-9.42 ± 2.2	-9.37 ± 1.5
BIT* (1.2s)	90	1.73 ± 0.7	5.78 ± 1.2	NA	-6.80 ± 2.4	-5.63 ± 2.4
BIT* (6s)	94	1.60 ± 0.6	5.72 ± 1.3	NA	-6.94 ± 2.4	-5.75 ± 2.5
Ours (1 batch)	100	1.02 ± 0.3	5.89 ± 1.0	0.75 ± 0.04	-6.39 ± 1.3	-6.02 ± 1.2
Ours (5 batches)	100	1.00 ± 0.2	5.74 ± 0.9	3.69 ± 0.25	-6.19 ± 1.4	-5.76 ± 1.3

(d) Evaluation in the "Cage" problems

	% Success	EE length	C-space length	Times (s)	EE-LDJ	C-space-LDJ
GF	88	0.66 ± 0.1	7.87 ± 0.8	0.16 ± 0.02	-5.3 ± 1.4	-7.02 ± 0.8
Bi-EST	100	10.67 ± 5.2	37.3 ± 15.4	1.99 ± 2.1	-12.57 ± 1.4	-12.02 ± 1.4
BIT* (1.2s)	2	NA	NA	NA	NA	NA
BIT* (6s)	2	NA	NA	NA	NA	NA
Ours (1 batch)	98	0.75 ± 0.1	6.91 ± 0.9	0.82 ± 0.04	-7.27 ± 2.2	-7.97 ± 1.7
Ours (5 batches)	100	0.73 ± 0.1	6.31 ± 0.9	4.06 ± 0.2	-7.46 ± 1.7	-7.94 ± 1.4

(e) Evaluation in the "Thin shelf" problems

	% Success	EE length	C-space length	Times (s)	EE-LDJ	C-space-LDJ
GF	80	0.68 ± 0.2	6.49 ± 1.4	0.11 ± 0.02	-6.06 ± 2.3	-6.56 ± 1.2
Bi-EST	100	5.11 ± 2.4	16.50 ± 7.1	0.033 ± 0.04	-9.89 ± 1.7	-9.86 ± 1.5
BIT* (1.2s)	86	1.89 ± 0.7	6.16 ± 1.2	NA	-8.04 ± 1.4	-6.79 ± 1.3
BIT* (6s)	92	1.66 ± 0.6	6.01 ± 1.1	NA	-8.51 ± 1.2	-6.70 ± 1.1
Ours (1 batch)	100	0.93 ± 0.2	5.64 ± 0.8	0.82 ± 0.04	-7.32 ± 1.7	-7.05 ± 1.4
Ours (5 batches)	100	0.92 ± 0.2	5.38 ± 0.8	4.33 ± 0.4	-7.38 ± 1.4	-6.68 ± 1.4

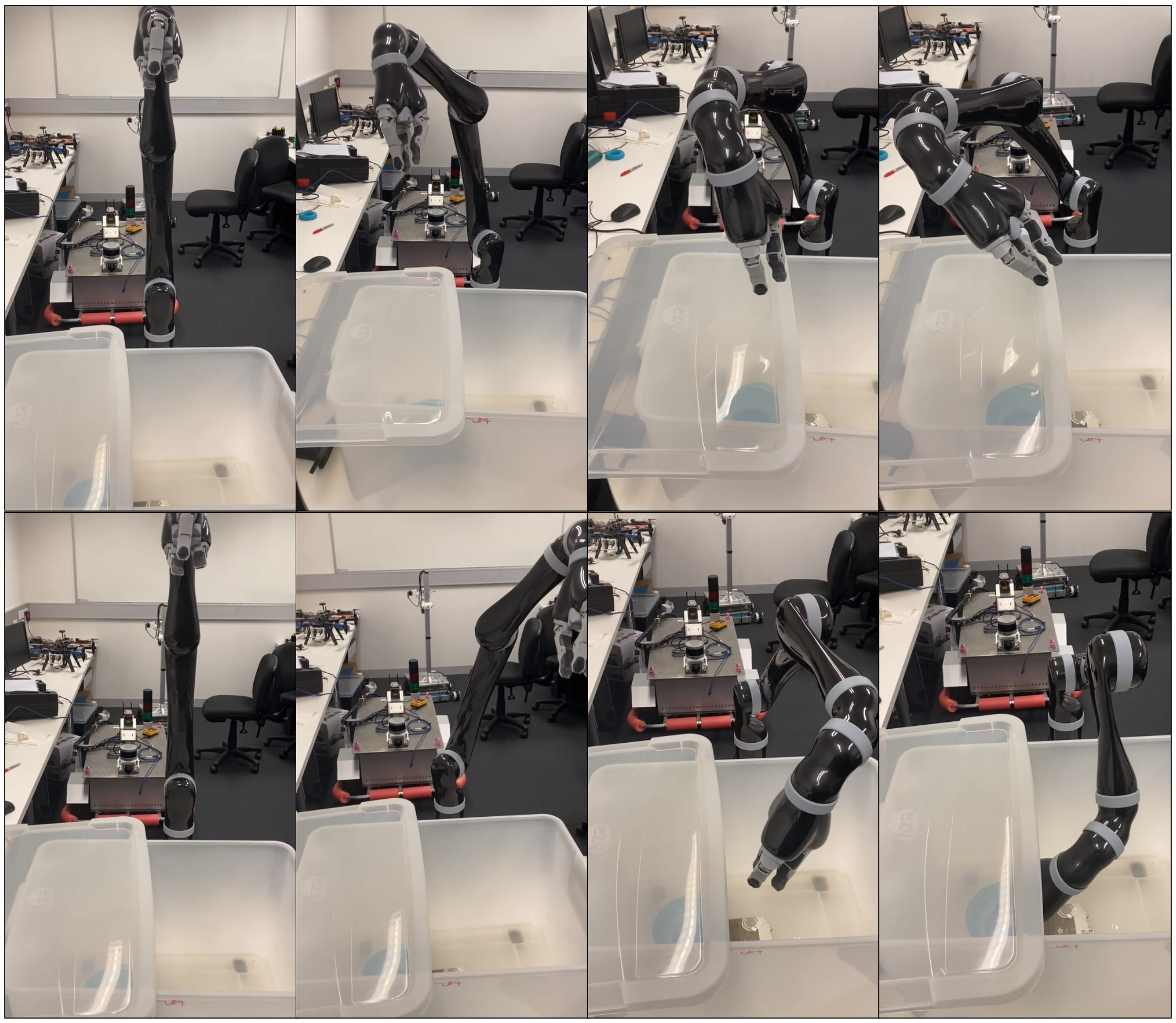
(f) Evaluation in the "Single shelf" problems

	% Success	EE length	C-space length	Times (s)	EE-LDJ	C-space-LDJ
GF	94	0.69 ± 0.2	6.51 ± 1.46	0.084 ± 0.02	-5.11 ± 1.8	-5.78 ± 1.4
Bi-EST	100	5.64 ± 2.8	16.26 ± 7.2	0.066 ± 0.13	-10.04 ± 1.8	-9.89 ± 1.6
BIT* (1.2s)	86	1.51 ± 0.6	5.79 ± 1.7	NA	-6.75 ± 2.8	-5.38 ± 2.7
BIT* (6s)	94	1.61 ± 0.8	5.48 ± 1.5	NA	-6.88 ± 2.9	-5.26 ± 2.7
Ours (1 batch)	100	0.90 ± 0.2	5.79 ± 1.2	0.73 ± 0.03	-6.97 ± 1.4	-6.65 ± 1.3
Ours (5 batches)	100	0.91 ± 0.2	5.68 ± 1.1	3.62 ± 0.3	-7.03 ± 1.5	-6.64 ± 1.2

Table 9.2: First batch motion by uniformly sampling, and by a learned model for "Table under".

	% Success	EE length	C-space length	EE-LDJ	C-space-LDJ
No learning	90	1.85 ± 0.5	13.53 ± 2.9	-8.61 ± 1.6	-8.76 ± 1.4
Learning	**96**	$\mathbf{1.43 \pm 0.5}$	$\mathbf{9.35 \pm 2.4}$	$\mathbf{-7.45 \pm 1.8}$	$\mathbf{-7.51 \pm 2.0}$

Figure 9.5: Geometric Fabrics (4 figs on the top) perform local avoidance and fail to reach the blue goal. GFCS (4 figs on the bottom) performs global optimisation and finds a non-local solution.



9.5.5 Execution on a Real Robot

To evaluate the robustness of global motion found from GFCS, we generate motion on a real-world JACO manipulator to reach a goal in a half-covered box. Qualitative results from Geometric Fabrics and GFCS are shown in fig. 9.5, where local avoidance by Geometric Fabrics is insufficient to reach the goal, and the manipulator stabilises on the cover. GFCS avoids the local minimum and reaches into the box towards the goal. A video illustrating GFCS controlling a manipulator to reactively avoid new obstacles in simulation, and execution on the real-world robot is available at `https://youtu.be/8LFFhriblLs`.

9.6 Summary

We present Geometric Fabric Command Sequences, a novel approach to generate global and reactive motion. This is achieved by running global optimisation over selected parameters of several sequentially joined command parameters for *Geometric Fabrics*. The speed and performance of the optimisation can then be improved by transferring the knowledge of solving similar problems. We introduce a self-supervised framework, where an implicit generative model iterative learns to provide informed candidate solutions to the global optimisation procedure. We validate our approach on a range of problem classes, both in simulation and on a real-world JACO manipulator.

In the following chapter 10, we shall summarise the contributions of this thesis, explore their connecting themes, and discuss potential future directions.

Chapter 10

Conclusions and Future Work

10.1 Summary of Contributions

This thesis presents methods for robots to learn from experience to understand, extrapolate and make decisions in unstructured environments. Contemporary robots often struggle to resiliently handle the endless possible permutations that can arise in the real-world, as doing so would require a prohibitively large set of hand-crafted rules to be specified. Learning approaches enable robots to move away from hand-designed rules and to generalise from experience. Learning can be flexibly integrated to a variety different components of the robotics autonomy stack. Under the overarching motivation of enabling greater autonomy in unstructured and dynamic environments with robot learning, we divide the thesis into three main themes:

- Learning for environment representation;
- Learning for anticipatory navigation;
- Learning for robot manipulator motion generation.

Within each of these themes, we contribute novel robot learning methods and evaluate them thoroughly, both in simulation and using world-real data or on real robots. While each of these themes deals with a different problem setting, there are underlying commonalities behind the techniques proposed, and each connects with discussions that arise from the previous. In this section, we shall summarise these contributions and discuss possible avenues for future research.

10.1.1 Learning Representations of the Environment

In part I, we focus our efforts on developing methods that learn continuous representations of the environment. Traditionally, grid-based methods have been used to represent the environment – these approaches divide the domain into fixed-resolution cells and compute attributes for each cell independently. Learning-based approaches instead learn a continuous mapping from the domain to the attributes of interest, and the representation is abstracted away as a concise set of parameters. These representations are a more natural way to represent the continuous real-world and provide benefits such as greater memory-efficiency and improved performance when data is sparse. Here, we describe the main contributions of each chapter under this theme.

Fusion of Continuous Occupancy Maps

In chapter 3, we study the problem of continuously representing the occupancy, i.e. the probability of a location being occupied. Specifically, we explore the application of continuous representation in multi-agent problem setups.

We introduce Fast Bayesian Hilbert Maps (Fast-BHM), a reformulation of Bayesian Hilbert Maps (BHM) (Senanayake and Ramos, 2017). We observe that BHM are prohibitively slow to build. To speed-up map building, we make the mean-field assumption when estimating the covariance matrix and re-derive equations for updating the map model. We demonstrate that imposition of the mean-field covariance assumption provides significant efficiency gains, with little degradation of performance.

More importantly, we contribute a method to iteratively combine multiple Fast-BHM models. We demonstrate that Fast-BHMs are more memory-efficient than grid-based approaches, owing to the number of parameters of Fast-BHMs being much smaller than number of grid-cells required to sufficiently represent the environment. This property makes Fast-BHMs particularly attractive for multi-agent systems where the bandwidth for map transmission is restricted.

Continuous Spatiotemporal Maps of Motion Directions

In chapter 4, we study the representation of continuous environments beyond the static setting, into the dynamic setting. In particular, we aim to build models capable of capturing the distribution of movement directions in the environment over time. These models endow robots with the ability to reason about how dynamic objects in the environment generally move, and may inform downstream planning tasks.

We contribute continuous spatiotemporal maps of motion directions, a model that learns a mapping from the space-time domain to a distribution of motion directions. Our model is capable of (1) handling the directional distributions which are multi-model, and (2) enforcing the support of the distribution to be between $[-\pi, \pi]$. We empirically demonstrate that the continuous nature of our model delivers performance benefits over discretised representations, particularly when the data is sparse. This is due to discretised representations assuming that each cell is independent from one another, and completely disregard information from neighbouring cells.

10.1.2 Anticipatory Navigation with Learning

Following the discussions presented in chapter 4, where we explored continuous representations for dynamic environments, in part II, we now look at imbuing a ground robot with the ability to navigate smoothly through dynamic environments. This is a setting where we require the robot to learn to continuously predict the dynamics of moving agents in the surroundings. To smoothly negotiate a path through the crowd, we develop methods to probabilistically *anticipate* the movement of others in the vicinity, and then incorporate the predictions of motions into the decision-making process.

Stochastic Process Anticipatory Navigation

In chapter 5, we contribute the Stochastic Process Anticipatory Navigation (SPAN) framework. SPAN learns the motion patterns of other moving agents in the surroundings, and models the agents' movements as stochastic processes – distributions over continuous functions mapping time from position. The outputs of the probabilistic predictive models are then integrated into the chance-constraints of a time-to-collision (TTC) control problem, which is solved in a receding horizon manner.

We empirically evaluate the performance of SPAN to navigate through simulated crowds and real-world pedestrian data. We demonstrate that for our problem setup of navigating through a dynamic crowd with a non-holonomic ground robot, reactive methods would often lead the robot into unrecoverable locations in the crowd and become stuck. On the other hand, SPAN is able to smoothly maneuver through the crowd. Additionally, we observe the emergent behaviour of the robot "hitch-hiking" behind pedestrians moving in the same direction.

Trajectory Generation in New Environments from Past Experiences

In chapter 6, we propose a framework, called Occupancy-Conditional Trajectory Network (OTNet), to condition on the structure of an environment to generate motion trajectories which are likely to appear in the environment. Humans have a great intuition to how agents in an environment would move, by observing a floor plan of the environment. We seek to empower robots to do the same, and have the ability to deduce motion patterns from the environment structure. OTNet encodes maps of environments by considering their Hausdorff distances from a set of representative maps to obtain fixed length encoding vectors. These are then inputted into a neural network to predict parameters of mixtures of trajectory distributions, which represent the motion trajectories we are expected to observe in the environment. We demonstrate that we can additionally enforce the start position of the distribution of trajectories. We evaluate on a simulated dataset containing environments and trajectories, and additionally qualitatively evaluate the quality of generating trajectories on a real world environment dataset, by generalising experience on the simulated dataset.

Structurally Constrained Motion Prediction

In chapter 7, we improve the quality of motion prediction and develop methods capable of modelling multi-modal distributions of trajectories, and can incorporate prior knowledge of the structure of the surrounding environment as constraints. We know *a priori* that agents in the environment are unlikely to move into occupied regions. Therefore, provided an occupancy representation of surroundings, our method allows us to impose chance constraints such that the probability of a moving agent entering occupied space is bounded. We leverage the availability of gradients from continuous representations of the environments, similar to the models discussed in part I, and apply gradient-based constrained optimisers. We demonstrate, on both simulated and real-world data, that our method encourages probabilistic predictions to be more compliant with the environment structure.

10.1.3 Learning to Generate Robot Manipulator Motion

In part III, we study how to generate desired motion trajectories for manipulators in unstructured environments. As discussed in part II, anticipation are particularly helpful in enabling ground robots to avoid "freezing" in a crowd, while in robot manipulation, we rarely encounter a "crowd" of dynamic objects in the manipulator's vicinity. Additionally, manipulators typically possess higher degree-of-freedoms, and are not subject to non-holonomic constraints. We instead opt for handling dynamic obstacles in the environment with reactive approaches which compute instantaneous control solutions by considering the current states only. Here, we use learning to enable robot manipulators to imitate and generalise human demonstrations (chapter 8), and to find more globally optimal solutions (chapter 9).

Diffeomorphic Templates for Generalised Imitation Learning

In chapter 8, we tackle the *generalised imitation learning* problem. It is often challenging to accurately design or specify the robot's behaviour when faced with new tasks. Instead, imitation learning, also known as learning from demonstration, can be applied for the robot to learn the behaviour from a fairly small set of human expert demonstrations. Beyond simply reproducing the demonstrated motions, we additionally wish to enable robots to generalise the learned skills, under changes in the problem setup or environment. These changes can include additional obstacles in the environment, or newly specified biases provided by the user.

We introduce the *Diffeomorphic Templates* (DT) framework for generalised imitation learning. Under this framework, robot motions are represented as integrals of asymptotically stable dynamical systems. DTs are defined as differentiable and invertible functions (diffeomorphisms) which encode a specific behaviour. We can learn DTs to imitate demonstrations, or craft DTs to produce generalisation behaviour. These DTs can encode various behaviours, such as avoiding new obstacle, or to accounting for additional instructions. We devise an approach to compose multiple DTs in the configuration space of the robot, producing a combined system while remaining asymptotically stable. We demonstrate the ability of the Diffeomorphic Templates framework to learn and generalise novel skills in both simulation and on a real-world manipulator.

Global and Reactive Motion Generation with Geometric Fabric Command Sequences

In chapter 9, we introduce *Geometric Fabric Command Sequences* (GFCS) – a global and reactive method to produce safe motion between start and goal configurations. GFCS builds upon Geometric Fabrics (Van Wyk et al., 2022), a reactive motion generation approach capable of producing smooth and legible trajectories. Purely reactive approaches, such as Geometric Fabrics, produce actions which only take into account the immediate state, often leading to undesirable solutions that are infeasible and trapped in a local minimum. To combat this drawback, GFCS casts motion generation as a global optimisation problem over a concise set of parameters of a sequence of Geometric Fabrics, which we call commands. We then solve the optimisation problem via a black-box optimiser.

Additionally, we contribute a self-supervised learning framework to learn to warm-start the optimiser. Global optimisers can be greatly sped up by providing informative initial batches of solutions. We hypothesise that the experience of finding solutions in one environment may be transferable to alternative environments. To this end, we train an implicit generative model, conditional on the environment and problem setup, to generate informative initial solutions. A key insight is that the training of the generative model and the solving of the optimisation problem are complementary. That is, as we continuously solve to obtain more solution, we have more training data to refine the generative model, which in turn boost the efficiency and quality of the optimisation. Therefore, we incorporate both the learning and optimisation into a single loop, and incrementally solve-and-train in a self-supervised manner.

10.2 Future Directions

In this section, we shall outline specific future lines of enquiry based on the methods and results presented in this thesis.

10.2.1 Time-to-Collision for Collaborative Manipulation

In chapter 5, we have demonstrated that Time-to-Collision (TTC) is effective as a component of a control problem where the robot is closely interacting with moving obstacles in the environment. We posit that TTC can also be effective to generate interactive behaviour for robot manipulators. Methods for approximating TTC can be computed very efficiently, for example in Davis et al. (2020), by assuming robot controls are held and agents move at constant velocities. This potentially allows for the approximate TTC to be included in settings where the manipulator needs to remain fast and reactive, such as those in chapter 9. Additionally, instead of simply trying to maximise the expected TTC to reduce the risk of being in-collision, we envision TTC to be constrained within a range to facilitate collaborative behaviour. For example, to enable the robot to stay close to humans in the environment, while also ensuring that collisions do not occur.

10.2.2 Differentiable SPAN for Learning to Navigate

The focus of part II has been to develop probabilistic predictive models to model the motion of other agents in the environment and to use these predictions in a downstream control problem. Here observe that how the robot behaves, conditioned on the predictions, is specified via a cost and not learned. However, it is improbable that a constructed control cost is able to capture all the nuances of navigating through crowds. A future line of work can focus on injecting learning directly into the decision-making process, while still making use of the strong inductive biases provided by the SPAN model. This can potentially be achieved by developing a differentiable Stochastic Process Anticipatory Navigation (SPAN) model, allowing the control problem to be differentiated end-to-end. This shall allow imitation learning or reinforcement learning methods to be applied to SPAN.

10.2.3 Environment Generation for Active Self-supervised Learning

The self-supervised learning framework to learn to warm-start optimisation, presented in chapter 9, brings optimisation for motion generation and learning a generative into a unified loop. A great opportunity for future work is to also incorporate an environment generator into the loop. The environment generator will then actively generate environments such that they provide the greatest improvement to the training of the generative model, while being feasible to solve. The environment generator may take the form of a generative model over continuous occupancy maps models, such as Fast-BHMs (chapter 3). Continuous occupancy representation is concise, requiring a relatively small number of parameters, and thus reduces the dimension of the data distribution for the generative model.

Bibliography

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. OSDI'16, 2016.

A. Agarwal and B. Triggs. Tracking articulated motion using a mixture of autoregressive models. In *ECCV*, 2004.

A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese. Social lstm: Human trajectory prediction in crowded spaces. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–971, 2016.

H. Alt and M. Godau. Computing the fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.*, 1995.

S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 1998.

R. Ambrus, P. Jensfelt, and J. Folkesson. Modeling motion patterns of dynamic objects by iohmm. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

O. Amidi and C. E. Thorpe. Integrated mobile robot control. In *Mobile Robots V*, volume 1388, pages 504–523. International Society for Optics and Photonics, 1991.

J. Amirian, J.-B. Hayet, and J. Pettré. Social ways: Learning multi-modal distributions of pedestrian trajectories with gans. In *CVPR Workshops*, 2019.

D. Arbuckle, A. Howard, and M. Mataric. Temporal occupancy grids: a method for classifying the spatio-temporal properties of the environment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

D. V. Arnold and N. Hansen. A (1+1)-cma-es for constrained optimisation. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, 2012.

A. Auger and N. Hansen. Tutorial cma-es: evolution strategies and covariance matrix adaptation. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, 2012.

P. J. Baddoo, B. Herrmann, B. J. McKeon, and S. L. Brunton. Kernel learning for robust dynamic mode decomposition: linear and nonlinear disambiguation optimization. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2022.

A. Bajcsy, S. Bansal, E. Ratner, C. J. Tomlin, and A. D. Dragan. A robust control framework for human motion prediction. *IEEE Robotics and Automation Letters*, 2021.

D. Bareiss and J. van den Berg. Generalized reciprocal collision avoidance. *The International Journal of Robotics Research*, 2015.

J. V. D. Berg, S. J. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In *ISRR*, 2009.

P. C. Besse, B. Guillouet, J. Loubes, and F. Royer. Review and perspective for distance-based clustering of vehicle trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 2016.

S. Bhattacharyya, A. Konar, and D. Tibarewala. Motor imagery and error related potential induced position control of a robotic arm. *IEEE/CAA Journal of Automatica Sinica*, 2017.

C. M. Bishop. Mixture density networks. Technical report, Aston University, 1994.

C. M. Bishop. *Pattern recognition and machine learning, 5th Edition.* Information science and statistics. Springer, 2007.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recogn.*, 30(7):1145–1159, July 1997.

L. F. Bruno Brito, Boaz Floor and J. Alonso-Mora. Model predictive contouring control for collision avoidance in unstructured dynamic environments. 2019.

N. C. Mitsou and C. Tzafestas. Temporal occupancy grid for mobile robot dynamic environment mapping. 2007.

P. Cai, Y. Luo, D. Hsu, and W. S. Lee. Hyp-despot: A hybrid parallel algorithm for online planning under uncertainty. *International Journal of Robotics Research*, 2020.

S. Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 2016.

C. Chamzas, C. Quintero-Peña, Z. K. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki. Motionbenchmaker: A tool to generate and benchmark motion planning datasets. *IEEE Robotics and Automation Letters*, 2022.

C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff. Rmpflow: A computational graph for automatic motion policy generation. In *International Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2018.

H.-T. Chiang, B. Chaudhuri, A. P. Vinod, M. Oishi, and L. Tapia. Dynamic risk tolerance: Motion planning by balancing short-term and long-term stochastic dynamic predictions. In *IEEE International Conference on Robotics and Automation*, 2017.

D. Coleman, I. A. Sucan, S. Chitta, and N. Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *Journal of Software Engineering for Robotics*, 2014.

R. N. Darmanin and M. K. Bugeja. A review on multi-robot systems categorised by application domain. In *Control and Automation (MED), 2017 25th Mediterranean Conference on*, pages 701–706. IEEE, 2017.

B. Davis, I. Karamouzas, and S. J. Guy. Nh-ttc: A gradient-based framework for generalized anticipatory collision avoidance. *Robotics: Science and Systems*, 2020.

A. P. Dawid. Some matrix-variate distribution theory: Notational considerations and a bayesian application. *Biometrika*, 1981.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 1977.

J. Denavit, R. Hartenberg, B. Mooring, G. Tang, D. Whitney, and C. Lozinski. 54 kinematic parameter. *Journal of applied mechanics*, 1955.

K. Doherty, J. Wang, and B. Englot. Probabilistic map fusion for fast, incremental occupancy mapping with 3d hilbert maps. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

J. Dong, M. Mukadam, F. Dellaert, and B. Boots. Motion planning as probabilistic inference using gaussian processes and factor graphs. In *Robotics: Science and Systems*, 2016.

Y. Du and I. Mordatch. Implicit generation and modeling with energy based models. In *Advances in Neural Information Processing Systems*, 2019.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

E. Dupont, A. Doucet, and Y. W. Teh. In *Advances in Neural Information Processing Systems*, 2019.

H. F. Durrant-Whyte and T. C. Henderson. Multisensor data fusion. 2008.

P. Dutilleul. The mle algorithm for the matrix normal distribution. *Journal of Statistical Computation and Simulation*, 1999.

D. K. Duvenaud. Automatic model construction with gaussian processes. Technical report, University of Cambridge, 2014.

A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 1987.

P. Englert and M. Toussaint. Learning manipulation skills from a single demonstration. *International Journal of Robotics Research*, 2018.

Federal Highway Administration. Lankershim boulevard dataset. Technical report, 2007a.

Federal Highway Administration. Ngsim peachtree street (atlanta) data analysis. Technical report, 2007b.

P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 1998.

S. R. Flaxman. Machine learning in space and time. Technical report, Carnegie Mellon University, 2015.

P. Florence, C. Lynch, A. Zeng, O. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. In *Conference on Robot Learning (CoRL)*, 2021.

D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, and B. Stewart. Distributed multirobot exploration and mapping. *Proceedings of the IEEE*, 2006.

G. Francis, L. Ott, and F. Ramos. Fast stochastic functional path planning in occupancy maps. *International Conference on Robotics and Automation (ICRA)*, 2019.

M. Füllsack. Systems sciences at isis, 2013.

J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa. Batch informed trees (bit*): Informed asymptotically optimal anytime search. *The International Journal of Robotics Research*, 39(5), 2020.

K. Ganchev, J. a. Graça, J. Gillenwater, and B. Taskar. Posterior regularization for structured latent variable models. *J. Mach. Learn. Res.*, 2010.

F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: continual prediction with lstm. In *International Conference on Artificial Neural Networks*, 1999.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

I. J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, Cambridge, MA, USA, 2016.

K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

J. Gregory, J. Fink, E. Stump, J. Twigg, J. Rogers, D. Baran, N. Fung, and S. Young. Application of multi-robot systems to disaster-relief scenarios with limited communication. In *Field and Service Robotics*, pages 639–653. Springer, 2016.

A. Gupta, J. E. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

B. Haasdonk and C. Bahlmann. Learning with distance substitution kernels. In *DAGM-Symposium, Lecture Notes in Computer Science*, 2004.

D. Hahnel, W. Burgard, D. Fox, and S. Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, 2003.

N. Hansen. The CMA evolution strategy: A tutorial. *CoRR*, 2016.

N. Hansen and S. Kern. Evaluating the cma evolution strategy on multimodal test functions. 01 2004.

N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. GECCO '10, 2010.

J. C. Hayward. Near-miss determination through use of a scale of danger. *Highway Research Record*, 1972.

T. Higuchi. Visuomotor control of human adaptive locomotion: Understanding the anticipatory nature. *Frontiers in Psychology*, 2013.

L. U. Hjorth and I. T. Nabney. Regularisation of mixture density networks. In *International Conference on Artificial Neural Networks*, 1999.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 1997.

T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 2008.

N. Hogan and D. Sternad. Sensitivity of smoothness measures to movement duration, amplitude, and arrests. *Journal of Motor Behavior*, 41(6), 2009.

A. Howard. Multi-robot simultaneous localization and mapping using particle filters. *The International Journal of Robotics Research*, 25(12):1243–1256, 2006.

A. Howard and N. Roy. The robotics data set repository (radish), 2003. URL `http://radish.sourceforge.net/`.

D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proceedings of International Conference on Robotics and Automation*, 1997.

Y. Huang, L. Rozo, J. Silvério, and D. G. Caldwell. Kernelized movement primitives. *The International Journal of Robotics Research*, 2019.

D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1993.

B. Ichter, J. Harrison, and M. Pavone. Learning sampling distributions for robot motion planning. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 2013.

B. Ivanovic and M. Pavone. The trajectron: Probabilistic multi-agent trajectory modeling with dynamic spatiotemporal graphs. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

B. Ivanovic, E. Schmerling, K. Leung, and M. Pavone. Generative modeling of multimodal multi-human behavior. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

T. Jaakkola and M. Jordan. A variational approach to bayesian logistic regression models and their extensions. In *Sixth International Workshop on Artificial Intelligence and Statistics*, volume 82, page 4, 1997.

M. G. Jadidi, J. V. Miró, R. Valencia, and J. Andrade-Cetto. Exploration on continuous gaussian process frontier maps. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6077–6082, 2014.

N. Jaipuria, G. Habibi, and J. P. How. A transferable pedestrian motion prediction model for intersections with different geometries. *ArXiv*, 2018.

M. Jordan and A. Perez. Optimal bidirectional rapidly-exploring random trees. Technical report, MIT-CSAIL, 2013.

M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. *IEEE International Conference on Robotics and Automation*, 2011.

S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 2011.

I. Karamouzas, B. Skinner, and S. J. Guy. Universal power law governing pedestrian interactions. *Phys. Rev. Lett.*, 2014.

I. Karamouzas, N. Sohre, R. Narain, and S. J. Guy. Implicit crowds: Optimization integrator for robust crowd simulation. *ACM Transactions on Graphics*, 36(4), July 2017. doi: 10.1145/3072959.3073705.

L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 1996.

S. M. Khansari-Zadeh and A. Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 2011.

S. M. Khansari-Zadeh and A. Billard. Learning control lyapunov function to ensure stability of dynamical system-based robot reaching motions. *Robotics Auton. Syst.*, 62:752–765, 2014.

O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *IEEE International Conference on Robotics and Automation*, 1985.

S. Kim and J. Kim. Recursive bayesian updates for occupancy mapping and surface reconstruction. In *Australasian Conference on Robotics and Automation*, 2014.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ArXiv*, 2015.

N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

J. F. Kooij, F. Flohr, E. A. Pool, and D. M. Gavrila. Context-based path prediction for targets with switching dynamics. *Int. J. Comput. Vision*, 2019.

J. F. P. Kooij, F. Flohr, E. A. I. Pool, and D. Gavrila. Context-based path prediction for targets with switching dynamics. *International Journal of Computer Vision*, 2018.

D. Kraft. A software package for sequential quadratic programming. tech. Technical report, DLR German Aerospace Center — Institute for Flight Mechanics, 1988.

T. Krajník, J. P. Fentanes, J. M. Santos, and T. Duckett. Fremen: Frequency map enhancement for long-term mobile robot autonomy in changing environments. *IEEE Transactions on Robotics*, 2017.

T. Kucner, J. Saarinen, M. Magnusson, and A. J. Lilienthal. Conditional transition maps: Learning motion patterns in dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

T. P. Kucner, M. Magnusson, E. Schaffernicht, V. H. Bennetts, and A. J. Lilienthal. Enabling flow awareness for mobile robots in partially observable environments. *IEEE Robotics and Automation Letters*, 2017.

T. Kulak, J. Silvério, and S. Calinon. Fourier movement primitives: an approach for learning rhythmic robot skills from demonstrations. In *RSS 2020*, 2020.

Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. How. Motion planning in complex environments using closed-loop prediction. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 7166, 2008.

T. Lai, W. Zhi, and F. Ramos. Occ-traj120: Occupancy maps with associated trajectories. *CoRR*, 2019.

T. Lai, W. Zhi, T. Hermans, and F. Ramos. Parallelised diffeomorphic sampling-based motion planning. In *Conference on Robot Learning (CoRL)*, 2021.

S. M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Department of Computer Science, Iowa State University, 1998.

S. M. Lavalle. *Planning Algorithms.* Cambridge University Press, 2006.

S. M. LaValle and J. James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 2001.

J. M. Lee. Introduction to smooth manifolds. 2012.

J. Li, H. Ma, and M. Tomizuka. Interaction-aware multi-agent tracking and probabilistic behavior prediction via adversarial learning. *International Conference on Robotics and Automation (ICRA)*, 2019.

Q. Liu and D. A. Pierce. A note on gauss-hermite quadrature. *Biometrika*, 81, 1994.

Y. Luo, P. Cai, A. Bera, D. Hsu, W. S. Lee, and D. Manocha. Porca: Modeling and planning for autonomous driving among many pedestrians. *IEEE Robotics and Automation Letters*, 2018.

K. M. Lynch and F. C. Park. *Modern Robotics: Mechanics, Planning, and Control.* Cambridge University Press, USA, 1st edition, 2017.

B. Majecka. Statistical models of pedestrian behaviour in the forum. Technical report, School of Informatics, University of Edinburgh, 2009.

R. Malladi, J. A. Sethian, and B. C. Vemuri. Shape modeling with front propagation: a level set approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1995.

K. V. Mardia. Statistics of directional data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1975.

Z. Marinho, A. D. Dragan, A. Byravan, B. Boots, S. Srinivasa, and G. J. Gordon. Functional gradient motion planning in reproducing kernel hilbert spaces. *Robotics: Science and Systems*, 2016.

W. McAllister, D. Osipychev, G. Chowdhary, and A. Davis. Multi-agent planning for coordinated robotic weed killing. In *Proceedings 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2018)*, page to appear, 2018.

L. McCalman, S. O'Callaghan, and F. Ramos. Multi-modal estimation with kernel embeddings for learning motion models. In *IEEE International Conference on Robotics and Automation*, 2013.

S. M. Mellado, G. Cielniak, T. Krajník, and T. Duckett. Modelling and predicting rhythmic flow patterns in dynamic environments. In *TAROS*, 2018.

R. Mendonça, M. M. Marques, F. Marques, A. Lourenço, E. Pinto, P. Santana, F. Coito, V. Lobo, and J. Barata. A cooperative multi-robot team for the surveillance of shipwreck survivors at sea. In *OCEANS 2016 MTS/IEEE Monterey*, pages 1–6. IEEE, 2016.

M. Nava, J. Guzzi, R. O. Chavez-Garcia, L. M. Gambardella, and A. Giusti. Learning long-range perception using self-supervision from short-range sensors and odometry. *IEEE Robotics and Automation Letters*, 2019.

K. Neumann and J. J. Steil. Learning robot motions with stable dynamical systems under diffeomorphic transformations. *Robotics Auton. Syst.*, 2015.

M. Neunert, C. de Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli. Fast nonlinear model predictive control for unified trajectory optimization and tracking. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, New York, NY, USA, 2006.

M. Nomura, S. Watanabe, Y. Akimoto, Y. Ozaki, and M. Onishi. Warm starting cma-es for hyperparameter optimization. In *AAAI*, 2021.

S. O'Callaghan, F. T. Ramos, and H. Durrant-Whyte. Contextual occupancy maps using gaussian processes. In *2009 IEEE International Conference on Robotics and Automation*, 2009.

S. T. O'Callaghan, S. P. N. Singh, A. Alempijevic, and F. T. Ramos. Learning navigational maps by observing human motion patterns. *IEEE International Conference on Robotics and Automation*, 2011.

T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. 2018.

T. P. Hill. Conflations of probability distributions. *Transactions of the American Mathematical Society*, 363, 2008.

A. Paraschos, C. Daniel, J. Peters, and G. Neumann. Probabilistic movement primitives. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, 2013.

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

R. Penrose. On best approximate solutions of linear matrix equations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 1956.

F. Pomerleau, P. Krüsi, F. Colas, P. Furgale, and R. Siegwart. Long-term 3d map maintenance in dynamic environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

M. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. *Advances in Optimization and Numerical Analysis*, 1994.

N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 1999.

M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip. Motion planning networks. In *International Conference on Robotics and Automation (ICRA)*, 2019.

A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, 2021.

F. Ramos and L. Ott. Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent. *International Journal Robotics Research*, 2016.

M. A. Rana, M. Mukadam, S. R. Ahmadzadeh, S. Chernova, and B. Boots. Towards robust skill generalization: Unifying learning from demonstration and motion planning. In *Proceedings of the 1st Annual Conference on Robot Learning*, 2017.

M. A. Rana, A. Li, D. Fox, B. Boots, F. Ramos, and N. Ratliff. Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems. In *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, 2020.

N. Ratliff. Learning geometric reductions for planning and control. In *ICML 2013 Workshop on Robot Learning*, 2013.

N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation*, 2009.

N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox. Riemannian motion policies, 2018.

N. D. Ratliff, K. V. Wyk, M. Xie, A. Li, and M. A. Rana. Generalized nonlinear and finsler geometry for robotics. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

X. Rong Li and V. P. Jilkov. Survey of maneuvering target tracking. part i. dynamic models. *IEEE Transactions on Aerospace and Electronic Systems*, 2003.

A. Rudenko, T. P. Kucner, C. S. Swaminathan, R. T. Chadalavada, K. O. Arras, and A. J. Lilienthal. Thör: Human-robot navigation data collection and accurate motion trajectories dataset. *IEEE Robotics and Automation Letters*, 2020a.

A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras. Human motion trajectory prediction: a survey. *The International Journal of Robotics Research*, 2020b.

A. Sadeghian, V. Kosaraju, A. Sadeghian, N. Hirose, H. Rezatofighi, and S. Savarese. Sophie: An attentive gan for predicting paths compliant to social and physical constraints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

M. Saveriano. An energy-based approach to ensure the stability of learned dynamical systems. *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. 2001.

R. Schubert, E. Richter, and G. Wanielik. Comparison and evaluation of advanced motion models for vehicle tracking. *International Conference on Information Fusion*, 2008.

J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: Science and Systems*, 2013.

R. Senanayake and F. Ramos. Bayesian hilbert maps for dynamic continuous occupancy mapping. In *Conference on Robot Learning (CoRL)*, 2017.

R. Senanayake and F. Ramos. Directional grid maps: modeling multimodal angular uncertainty in dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018.

V. Sindhwani, S. Tu, and M. Khansari. Learning contracting vector fields for stable imitation learning, 2018.

S. L. Smith, M. Schwager, and D. Rus. Persistent robotic tasks: Monitoring and sweeping in changing environments. *IEEE Transactions on Robotics*, 28(2):410–426, 2012.

E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*. 2006.

K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems*, pages 3483–3491, 2015.

I. standard 9899:1999. Programming language c., 1999.

I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, (4):72–82, December 2012.

U. Tan, O. Rabaste, C. Adnet, and J.-P. Ovarlez. On the eclipsing phenomenon with phase codes. In *2019 International Radar Conference (RADAR)*, 2019.

G. Tanzmeister, J. Thomas, D. Wollherr, and M. Buss. Grid-based mapping and tracking in dynamic environments using a uniform evidential environment representation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

S. Thrun. A probabilistic on-line mapping algorithm for teams of mobile robots. *The International Journal of Robotics Research*, 2001.

S. Thrun and Y. Liu. Multi-robot slam with sparse extended information filers. In P. Dario and R. Chatila, editors, *Robotics Research. The Eleventh International Symposium*, pages 254–266, 2005.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT Press, 2005.

T. Tieleman, G. Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 2012.

V. Tresp. A bayesian committee machine. *Neural Computation*, 12(11):2719–2741, 2000.

J. Urain, M. Ginesi, D. Tateo, and J. Peters. Imitationflows: Learning deep stable stochastic dynamic systems by normalizing flows. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.

A. Valada, P. Velagapudi, B. Kannan, C. Tomaszewski, G. Kantor, and P. Scerri. Development of a low cost multi-robot autonomous marine surface platform. In *Field and service robotics*, pages 643–658. Springer, 2014.

A. van den Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *CoRR*, 2018.

K. Van Wyk, M. Xie, A. Li, M. A. Rana, B. Babich, B. Peele, Q. Wan, I. Akinola, B. Sundaralingam, D. Fox, B. Boots, and N. D. Ratliff. Geometric fabrics: Generalizing classical mechanics to capture the physics of behavior. *IEEE Robotics and Automation Letters*, 2022.

D. Vasquez, T. Fraichard, and C. Laugier. Growing hidden markov models: An incremental tool for learning and predicting human and vehicle motion. *The International Journal of Robotics Research*, 2009.

W. Wang, M. Zhu, X. Wang, S. He, J. He, and Z. Xu. An improved artificial potential field method of trajectory planning and obstacle avoidance for redundant manipulators. *International Journal of Advanced Robotic Systems*, 2018.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, 2011.

G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. Theodorou. Aggressive driving with model predictive path integral control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

Z. Yang, S. Liu, H. Hu, L. Wang, and S. Lin. Reppoints: Point set representation for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

T. J. Ypma. Historical development of the newton-raphson method. *SIAM Review*, 1995.

S. Zernetsch, S. Kohnen, M. Goldhammer, K. Doll, and B. Sick. Trajectory prediction of cyclists using a physical model and an artificial neural network. In *IEEE Intelligent Vehicles Symposium (IV)*, 2016.

P. Zhang, W. Ouyang, P. Zhang, J. Xue, and N. Zheng. Sr-lstm: State refinement for lstm towards pedestrian trajectory prediction. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Y. Zhang, Z. Zhai, X. Nie, C. Ma, and F. Zuo. An extended self-adaptive kalman filtering object motion prediction model. In *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2008.

W. Zhi, L. Ott, and F. Ramos. Kernel trajectory maps for multi-modal probabilistic motion prediction. Conference on Robot Learning (CoRL), 2019.

W. Zhi, L. Ott, R. Senanayake, and F. Ramos. Continuous occupancy map fusion with fast bayesian hilbert maps. In *International Conference on Robotics and Automation (ICRA)*, 2019a.

W. Zhi, R. Senanayake, L. Ott, and F. Ramos. Spatiotemporal learning of directional uncertainty in urban environments with kernel recurrent mixture density networks. *IEEE Robotics and Automation Letters*, 2019b.

W. Zhi, T. Lai, L. Ott, and F. Ramos. Trajectory generation in new environments from past experiences. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021a.

W. Zhi, T. Lai, L. Ott, and F. Ramos. Anticipatory navigation in crowds by probabilistic prediction of pedestrian future movements. In *IEEE International Conference on Robotics and Automation, ICRA*, 2021b.

W. Zhi, L. Ott, and F. Ramos. Probabilistic trajectory prediction with structural constraints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2021c.

W. Zhi, T. Lai, L. Ott, E. V. Bonilla, and F. Ramos. Learning efficient and robust ordinary differential equations via invertible neural networks. In *International Conference on Machine Learning, ICML*, 2022a.

W. Zhi, T. Lai, L. Ott, and F. Ramos. Diffeomorphic transforms for generalised imitation learning. In *Learning for Dynamics and Control Conference, L4DC*, 2022b.

W. Zhi, K. van Wyk, I. Akinola, N. Ratliff, and F. Ramos. Global and reactive motion generation with geometric fabric command sequences. In *To appear in: IEEE International Conference on Robotics and Automation, ICRA*, 2023.

B. Zhou, W. Schwarting, D. Rus, and J. Alonso-Mora. Joint multi-policy behavior estimation and receding-horizon trajectory planning for automated urban driving. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

D.-X. Zhou. Derivative reproducing properties for kernel methods in learning theory. *Journal of Computational and Applied Mathematics*, 2008.

H. Zhu and J. Alonso-Mora. Chance-constrained collision avoidance for mavs in dynamic environments. *IEEE Robotics and Automation Letters*, 2019.

B. D. Ziebart, N. D. Ratliff, G. Gallagher, C. Mertz, K. M. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. S. Srinivasa. Planning-based prediction for pedestrians. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

M. Zucker, N. D. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 2013.