Learning-Based Path-Tracking Control for Ground Robots with Discrete Changes in Dynamics

by

Christopher McKinnon

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
University of Toronto Institute for Aerospace Studies

Abstract

Learning-Based Path-Tracking Control for Ground Robots with Discrete Changes in Dynamics

Christopher McKinnon
Doctor of Philosophy
Department of University of Toronto Institute for Aerospace Studies
University of Toronto
2021

Control algorithms such as stochastic model predictive control (SMPC) choose control inputs that guide the robot towards its goal by minimising a cost function while limiting the risk of collision to an acceptable threshold. This ability to manage risk makes it appropriate for autonomous mobile robots to manage the potential for damage to the robot and its environment. A key ingredient in SMPC is a dynamics model that predicts the motion of the robot, including an estimate of uncertainty in that prediction. This thesis focuses on using data to improve this dynamics model in changing conditions with limited prior knowledge about those changes. Existing approaches focus on either a single model that can be adapted to slowly changing dynamics or a fixed number of models that are known ahead of time. In contrast, we focus on the case where the robot dynamics may be subjected to an unknown number of potentially large changes.

First, we develop a method to model the robot dynamics as a Gaussian Process (GP), learning new models when new dynamics are encountered and leveraging existing models when possible. However, the GPs resulting from this method are too computationally expensive for use in closed-loop experiments. To reduce their computational cost, we switch to a repetitive path-following task and an experience-recommendation paradigm, whereby a single, local GP is constructed based on relevant data from previous traversals of the path. We then build on this framework using Bayesian Linear Regression (BLR) to model the robot dynamics which facilitates continuous adaptation to changing dynamics in addition to leveraging data from previous traversals. To make better use of data

available apriori, we develop a method to learn basis functions for BLR from data rather than requiring that they be specified by hand. Finally, we develop a method to learn a correction to the cost function used in SMPC that automatically discourages the robot from entering states where the dynamics model does not accurately predict the cost associated with a sequence of control inputs. These approaches are validated through a series of experiments on ground robots.

Acknowledgements

This thesis summarises the technical contributions of my non-linear journey through the local minima and maxima of robotics research. However, it would not be complete without acknowledging the people who formed the community that made this the unforgettable journey that it was.

First and foremost, to my parents, sister, and partner, thank you for your unconditional support over the years and for sharing the pursuit of your dreams which continues to encourage me to chase mine.

I would also like to extend my deepest gratitude to the students at UTIAS who were the role models, sources of inspiration, and community that made graduate life an enjoyable experience. To members of DSL (Abhi, Adam(s), Andreas, Bhavit, Carlos, Dave, Felix, Julian, Kaizad, Karime, Ke, Keenan, Michael(s), Mario, Melissa, Mohamed, Rikky, Sep, SiQi, Thomas, Wenda, Zach), thank you for all of the insightful discussions about research and interesting lunch-time conversations. A special thanks to Karime for blazing the trail as the first PhD student out of DSL and for sharing her hard-earned wisdom and encouragement during the final stretch of finishing my degree. Also, to the members of STARS and ASRL who extended this commuinty at UTIAS and on adventures around the world at conferences, thank you. A special thanks to Michael Paton, Michael Warren, Kirk MacTavish, and Peter Berczi for helping me get started with Visual Teach & Repeat 2.0.

I must also thank Prof. Joshua Marshall for giving me the incredible experience as a summer student that kicked off my interest in robotics and remains one of my fondest memories of working with robots. I would like to thank my supervisor, Prof. Angela Schoellig, for doing the amazing work during her PhD that inspired me to pursue graduate studies at UTIAS and for providing me with the opportunities and guidance to do so as a member of DSL. In addition, I would like to thank the members of my advisory committee, Prof. Jonathan Kelly and Prof. Tim Barfoot, and the additional members of my examination committee, Prof. Animesh Garg and Prof. Rolf Findeisen, for their time in reviewing this thesis and for asking the insightful questions that make robotics research interesting and meaningful.

To the many more people who helped me in one way or another, including the members and coaches of the U of T Triathlon Club, thank you.

Contents

Acknowledgements iv

Contents v

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Objectives . . . 3
- 1.3 Overview . . . 6
- 1.4 Notation . . . 9

2 Stochastic Model Predictive Control 10
- 2.1 Dynamic Systems . . . 10
- 2.2 Stochastic Model Predictive Control . . . 11
  - 2.2.1 Optimal Control Problem . . . 11
  - 2.2.2 Importance of the Dynamics Model . . . 12
  - 2.2.3 Recursive Feasibility . . . 13
  - 2.2.4 Value Function or Terminal Cost . . . 14
  - 2.2.5 Simplified SMPC Formulation . . . 15
- 2.3 Summary . . . 15

3 Gaussian Process-Based Methods 17
- 3.1 Introduction . . . 17
  - 3.1.1 Gaussian Process Regression . . . 18
  - 3.1.2 Related Work . . . 20
- 3.2 Learning Multi-Modal Models for Robot Dynamics with a Mixture of Gaussian Process Experts . . . 22
  - 3.2.1 Introduction . . . 22
  - 3.2.2 Problem Statement . . . 23
  - 3.2.3 Methodology . . . 24

3.2.4 Gaussian Process Disturbance Model . . . 25
3.2.5 Dirichlet Process . . . 25
3.2.6 Mode Inference . . . 26
3.2.7 Ground Robot Model . . . 26
3.2.8 Experiments . . . 27
3.2.9 Discussion . . . 33
3.2.10 Summary . . . 35
3.3 Experience Recommendation for Robot Modelling in Changing Conditions 35
3.3.1 Introduction . . . 35
3.3.2 Problem Statement . . . 37
3.3.3 Methodogy . . . 38
3.3.4 Experiments . . . 44
3.3.5 Discussion . . . 50
3.4 Summary and Contributions . . . 51

4 Bayesian Linear Regression-Based Methods 53
4.1 Introduction . . . 53
4.2 Problem statement . . . 55
4.3 Bayesian Linear Regression . . . 57
4.3.1 Computational Efficiency . . . 60
4.3.2 Simple Example . . . 61
4.3.3 Application to Repetitive Path-Following . . . 61
4.3.4 SMPC Controller Design . . . 64
4.4 Application to a Ground Robot . . . 67
4.4.1 Robot Model . . . 67
4.4.2 Ancillary Controller Design . . . 68
4.5 Offline Comparison to Gaussian Process Regression . . . 69
4.6 Closed-Loop Experiments with Bayesian Linear Regression . . . 71
4.6.1 Implementation . . . 71
4.6.2 Closed-Loop Tracking Performance with a Path-Dependent Disturbance . . . 72
4.6.3 High Speed Tracking Performance . . . 73
4.7 Closed-Loop Comparison to Gaussian Process Regression . . . 74
4.7.1 Experimental Setup . . . 75
4.7.2 Closed-Loop Comparison . . . 75
4.7.3 Multi-Step Prediction Performance . . . 76

4.7.4 Speed Comparison . . . 77
4.8 Summary . . . 78

5 Meta-Learning with Paired Forward and Inverse Models 80
5.1 Introduction . . . 81
5.2 Related Work . . . 81
5.3 Problem Statement . . . 84
5.4 Methodology . . . 86
5.4.1 Approach . . . 86
5.4.2 Multi-modal Learning . . . 89
5.5 Application to Grizzly . . . 90
5.5.1 Dynamics Model . . . 90
5.5.2 Input-feature Model Accuracy . . . 91
5.5.3 Cost Function . . . 92
5.5.4 Computational Advantage . . . 94
5.6 Experiments . . . 95
5.6.1 Model Accuracy Comparison . . . 97
5.6.2 Closed-Loop Experiments . . . 98
5.7 Discussion . . . 102
5.8 Summary and Contributions . . . 103

6 Cost Learning for Model Predictive Control 104
6.1 Introduction . . . 104
6.2 Related Work . . . 105
6.3 Problem Statement . . . 107
6.4 Methodology . . . 108
6.4.1 Cost Prediction Error . . . 108
6.4.2 Data Management . . . 109
6.4.3 Model for Cost Prediction Error . . . 109
6.4.4 Cost-based Mode Inference . . . 110
6.4.5 Augmented MPC Cost Function . . . 111
6.4.6 Model Learning . . . 111
6.5 Application to a Ground Robot . . . 111
6.5.1 Robot Model . . . 112
6.5.2 Implementation . . . 112
6.6 Experiments . . . 112
6.6.1 Cost Learning vs. Model Learning . . . 112

6.6.2 Cost-based Mode Inference . . . 115
6.6.3 State Dependent Measurement Noise in Simulation . . . 117
6.7 Discussion . . . 119
6.8 Summary and Contributions . . . 120

7 Summary and Future Work **121**
7.1 Summary of Contributions and Publications . . . 121
7.2 Future Work . . . 125

A Path Parametrisation **128**
A.1 Path Parametrization . . . 128

B Exponential Family Distributions **130**
B.1 Likelihood . . . 131
B.2 Prior . . . 131
B.3 Posterior . . . 131
B.4 Relation to Data Weighting . . . 132

Bibliography **132**

Bibliography **133**

Chapter 1

Introduction

1.1 Motivation

Mobile robots are becoming more and more common as advances in actuators, computing, and sensors make it possible to convert almost any vehicle that a human can operate into a robot. For example, farming and construction equipment, cars, trucks, drones, and household vacuum cleaners are acquiring the ability to complete tasks either semi-autonomously or autonomously, without a human operator. Following a path is a significant component of the tasks that many of these robots may be assigned. This requires the robot to know its location in the world (localization), have a route to follow to get from its current location to the completion of its task (planning), and to traverse this route in a reasonable amount of time using the actuation available on the robot given the path and an estimate of the robot's location in the world (control).

Advances in localisation and planning have made it possible for a robot to localise and plan in a wide range of environments. For example, low-cost cameras, lidars, and inertial sensors have enabled flying vehicles to navigate at high speed through outdoor and indoor environments [Mohta et al., 2018]; algorithms using stereo cameras enabled a ground robot to traverse long distances through large, seasonal appearance change [Paton et al., 2017]; and advances in machine learning enabled robots to leverage a deeper understanding of the environment around them, including the presence of pedestrians and other vehicles, and to identify drivable surface and lane markings on public roads [Liang et al., 2019]. The world accessible to robots is now vast.

From a controls perspective, which is the focus of this thesis, this freedom to plan and localise in an unprecedented range of environments means that robots may now encounter a wider variety of conditions than ever before, such as variable terrain, weather, and payload (as the robot performs tasks such as delivery). All of these affect the

robot's dynamics, which changes how the actuators affect the robot's motion through the environment. For example, the stopping distance of a garbage truck can increase by 62% when loaded compared to when it is empty [Hoover et al., 2013]. In addition, the presence of pedestrians, valuable infrastructure (which may include the robot itself), and the social capital of people's trust in autonomous systems are all at stake in this vast new world. This raises safety to the highest level of importance while performance (e.g., timely traversal of a path) also remains vitally important. In addition, these problems cannot be solved by modifying the environment–like in the past for manufacturing robots. Thus, equipping control algorithms to handle changes in robot dynamics caused by factors such as weather, terrain, payload, physical modifications to the robot such as different wheels, or general wear-and-tear are now at the forefront of research for robotic control.

This thesis seeks to address the need for robots to perform safely and effectively in varied, uncontrolled environments by focusing on the dynamics model used for controlling the robot. The dynamics model captures how the control input affects the state of the robot, and it is at the core of many control algorithms in robotics. Specifically, this thesis *(i)* improves model learning techniques for safe controllers to better capture uncertainty in the dynamics model thus improving safety, and *(ii)* improves multi-modal model learning techniques to allow robots to perform effectively given an unknown, variable number of possible dynamics models (caused by large possible variations in the robot's environment).

In terms of safe learning, the dynamics model can be used to optimise a cost function and to ensure the the robot achieves its goals in a safe and reliable way [Kober et al., 2013a, Berkenkamp and Schoellig, 2015]. If the model for the robot is partially unknown, the cost function can incorporate an element to encourage exploration of the system dynamics which establishes a better model for robot dynamics and enables the controller to later exploit well-known, low-cost actions [Moldovan et al., 2015, Xie et al., 2016]. An accurate assessment of the risk associated with each control input is important to ensure safety during this process [Berkenkamp et al., 2016, Ostafew et al., 2015]. Using an assessment of this risk to control the chance of failure during the learning process is known as safe learning.

Safe learning methods generally incorporate an approximate initial guess for the system dynamics with some bounds on the modelling error incurred in the approximation [Ostafew et al., 2016, Aswani et al., 2013]. A learning term then refines this initial guess over time using data gathered by operating the robot to better approximate the true dynamics. The goal is to guarantee that the system does not violate safety constraints (e.g., limits on the control input or tracking error) while achieving the control objective

(e.g., following a path) and at the same time improving the model of the system, and consequently, its task performance over time. This thesis seeks to improve safe learning by developing model learning approaches that provide a better assessment of model uncertainty, which is used to estimate the risk associated with taking a given control input, thus improving safety. Furthermore, the approaches developed in this thesis are designed to be computationally efficient, which benefits stochastic model predictive control–a type of safe controller used in this thesis that is relatively computationally demanding.

In terms of multi-modal learning, most learning algorithms learn a single model for robot dynamics or use multiple models trained ahead of time based on appropriate training data from operating the robot in all relevant conditions [Xie et al., 2016, Williams et al., 2017, Luders et al., 2013, Li et al., 2017, Aoude et al., 2013]. A single model can be made place-specific by including a dependence on, for example, distance along the path [Ostafew et al., 2016]. However other factors such as large payloads can change the dynamics at all places introducing multiple modes in the dynamics at each place. Assuming that a single model can represent the robot dynamics given the chosen dependence presents a challenge for robots that are deployed into a wide range of operating conditions because it may be difficult to anticipate and measure all of the relevant variables. Multimodal learning enables a model to adapt to discrete changes in variables that it does not explicitly depend on. This thesis seeks to improve multi-modal model learning by lifting the requirement that the number of modes be known ahead of time.

1.2 Objectives

This thesis focuses on the controls aspect of enabling robots to function reliably in a wide range of operating conditions. On a high level, the goal is to achieve the following objectives:

Safety and Reliability: A robot must be able to estimate reasonable bounds on error in its dynamics model in partially unknown environments. Bounds on model error are required by safe learning algorithms to ensure that the system does not violate constraints given the range of possible model errors. This thesis focuses on designing models for robot dynamics that achieve this objective and leverage existing control frameworks to perform closed-loop experiments.

Adaptation to New Conditions: This thesis takes the approach that learning a single model for all possible conditions *apriori* is challenging both from the standpoint of gathering enough data to properly fit the model and that such a model would

pose significant computational challenges that would likely require concessions in other parts of the controller (if it is feasible at all). Therefore, this thesis aims to incrementally learn models for robot dynamics relevant to the current operating conditions.

Data Efficiency and Data Capacity: It is reasonable to assume that a robot will be operated in at least one environment prior to deployment for testing purposes. This testing process generates data that may be informative about the dynamics of the robot. It is, therefore, beneficial if the model for robot dynamics can leverage this *apriori* dataset to improve performance when the robot is deployed in novel operating conditions. The amount of data that a robot can leverage to improve performance may be thought of as its *data capacity*. Once a robot is deployed in novel operating conditions, it is beneficial for the model to converge to the new dynamics without requiring large amounts of data to be gathered. This ability to converge with relatively few samples may be thought of as a model's *data efficiency*. Although these properties are sometimes at odds (because simple models are often data efficient but have low data capacity and complex models often have high data capacity but are not data-efficient) this thesis explores methods for achieving a reasonable compromise and demonstrate the effectiveness of that compromise in experiment.

Tolerance to Edge Cases: All models are based on assumptions (e.g., the form of the model or the set of variables on which it depends). Because of the difficulty of anticipating all possible operating conditions, a robot may encounter conditions that violate these assumptions and, consequently, degrade the accuracy of the model for robot dynamics. If this degradation is severe, it can have a large impact on the safety and performance of the control algorithm. Therefore, this thesis aims to enable a robot to automatically recognise cases when the model it is using does not have the capacity to learn the dynamics in the current operating environment and adapt its behaviour accordingly.

Computational Feasibility: Algorithms developed should run in real time on reasonable hardware so that they can be evaluated in closed-loop experiments. For this thesis, reasonable hardware will be defined as one relatively powerful laptop in 2020. The relevant specifications will be given for each experiment.

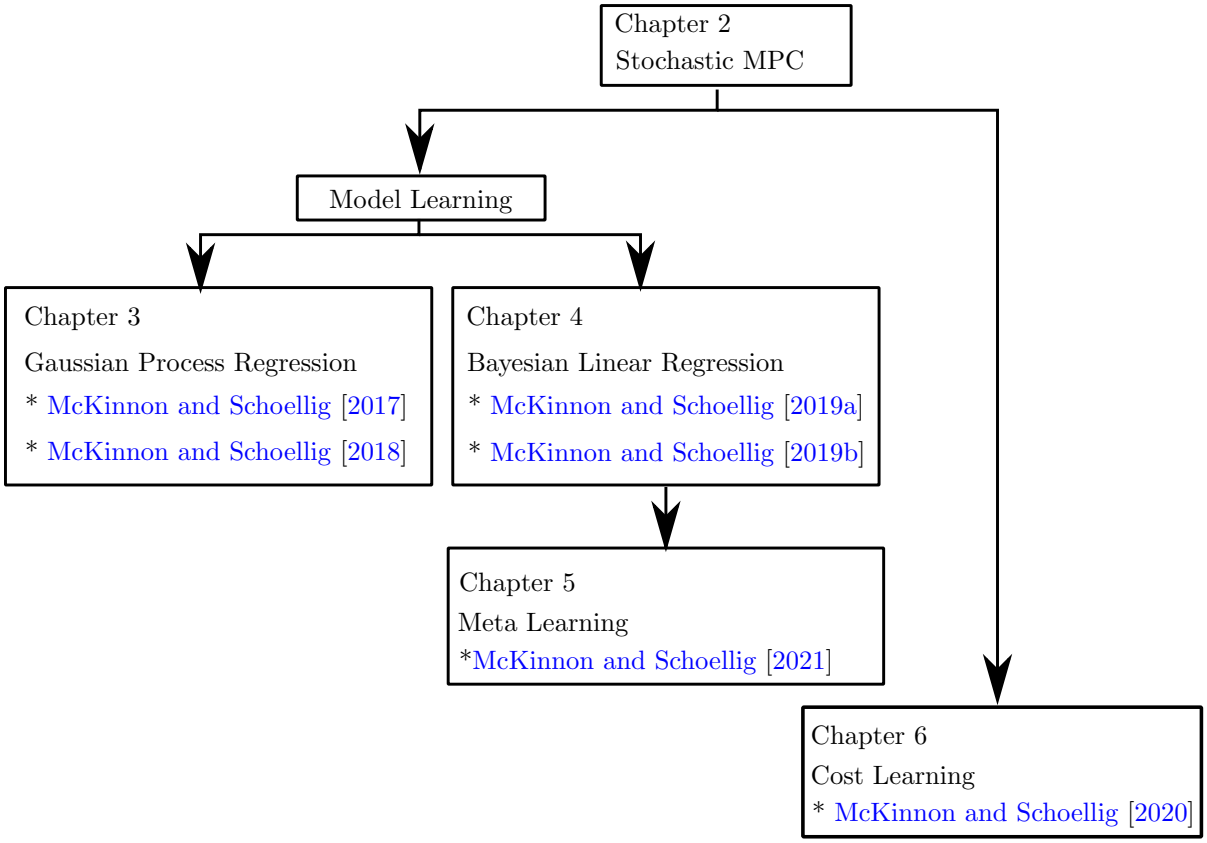


Figure 1.1: This diagram shows the structure and conceptual relation between its content. Items marked with a * refer to full-paper refereed conference and journal papers.

1.3 Overview

This thesis focuses primarily on model learning for Stochastic Model Predictive Control (SMPC), a model-based controller that takes into account uncertainty in the model for robot dynamics in order to limit the probability of a robot entering unsafe states (e.g. colliding with an obstacle). Chapter 2 outlines the important aspects of SMPC for the purpose of this thesis and identifies simplifications made for the purpose of studying models for robot dynamics.

The starting point for this thesis is a state-of-the-art learning-based SMPC Ostafew et al. [2016]. The approach by Ostafew et al. [2016] uses a Gaussian Process (GP) to learn a corrective term for a prior model for robot dynamics. This approach was demonstrated on a Clearpath Grizzly ground robot for repetitive path-following tasks. The first problem that this thesis seeks to address is extending this approach to the case when the dynamics can change depending on a discrete, latent variable; for example, a large payload or other disturbance that changes the robot's dynamics but cannot be measured directly without adding sensors to the robot. This thesis explores two possible methods for doing so.

The first method in Chap. 3 combines GP models in a Dirichlet Process mixture model which enables the model to learn an apriori unspecified number of nonlinear dynamics models. The ability of this method to learn meaningful dynamics models was demonstrated by modelling the dynamics of a ground robot being driven by a human operator while the vehicle dynamics were changed by adding a large payload. The main drawback of this method is that the GPs require many training points to model the robot dynamics accurately, which makes them too computationally expensive to use in closed-loop with SMPC. Chapter 3 also presents the follow-up method, which is an experience recommendation-based approach for constructing local GPs online using relevant data from previous runs for a repetitive path-following task. This method leverages past experience from a large number of runs to model recurring changes in dynamics and safely adapt to novel operating conditions by automatically reverting to a conservative prior. This method enables the controller to improve with each traversal as more experience is gathered in the relevant operating conditions. We demonstrated the effectiveness of this approach in closed-loop experiments on the Clearpath Grizzly with both physical and artificial changes to its dynamics.

A key assumption made by the previous methods, and one that is still common when applying GPs to model robot dynamics in practice, is that the GP hyperparameters can be fixed ahead of time and applied to model the dynamics in all operating conditions. In

our experiments in Chapter 3, we found that the GP-based model's performance varied significantly depending on the configuration of the robot or where it was deployed. Of particular concern was the variable accuracy of the model uncertainty estimates. This is linked to the fixed hyper-parameters in Sec. 4.3.

Chapter 4 presents a novel method to address this. This approach, based on Bayesian Linear Regression (BLR), can adjust model uncertainty to better reflect the distribution of model error. In addition, it can leverage experience from the live run to update the dynamics model continually. This is in contrast to the previous experience experience recommendation-based method, which only uses data from previous runs. We demonstrated, through several experiments, that the proposed method reliably achieved higher model accuracy compared to the GP-based methods and that this resulted in better closed-loop performance. This approach is appropriate for systems where the changing dynamics can be expressed as a linear combination of known basis functions.

A key limitation of the method presented in Chapter 4 is that it requires the dynamics to be close to a linear combination of known basis functions. While this may be reasonable for many robotic systems, it limits the ability of this method to leverage data available apriori to refine the dynamics model.

In Chapter 5, we explore a method use data available apriori to learn a basis function, which we call the input feature, that relates the control input to the time derivative of a particular state of the robot. We show that this basis function can be combined with others (representing prior knowledge about how the dynamics can change) using BLR to generalise to novel operating conditions. In addition, a novel feature of our approach is that we learn the forward and inverse model for the input feature, which enables us to leverage computationally expensive models to learn the input feature while incurring only a marginal increase in computational cost in closed loop. In summary, this method enables us to boost the *data capacity* of the model by learning an input feature while maintaining the *data efficiency* of BLR to adapt to changes in the model for robot dynamics.

Chapter 6 explores an alternative to learning the model for robot dynamics in order to improve the closed-loop performance of SMPC. This was motivated by the fact that model accuracy varied over the length of a path for both GP and BLR-based methods. In general, model accuracy may be limited by our prior knowledge about the robot dynamics, either by our ability to specify the form of the dynamics apriori or our ability to provide data and fit a model that will represent the dynamics while remaining computationally feasibility for SMPC. Furthermore, even with a perfect model, control performance can be affected by the localisation system, which provides the initial condition for predicting

future states in SMPC; if the vehicle's motion affects the performance of the localisation system, then it may be useful for the controller to adjust the vehicles motion to reduce its impact on localisation performance. To address this, we develop a novel method to take into account errors in predicting the cost over the SMPC horizon to improve tracking performance when the accuracy of the dynamics model or the localisation system varies. We demonstrated the utility of this method in the case of errors in the dynamics models in experiment, and show that there is potential for this method to reduce the impact of state-dependent localisation noise in simulation.

Finally, Chapter 7 summarises the contributions of this thesis and outlines potential areas for future research.

1.4 Notation

Symbol	Meaning
v	scalar variable
$\mathbf{v}$	vector variable
$\mathbf{M}$	matrix
$\mathcal{S}$	sets are generally denoted by calligraphic variables
$f(\cdot)$	function that outputs a scalar
$\mathbf{f}(\cdot)$	function that outputs a vector
$\mathcal{A} = \{\mathbf{a}_i\}_{i=1}^n$	a set of n discrete values
$\bar{v}$	mean of a random variable v
$\sigma^2, \mathbf{\Sigma}$	variance, covariance matrix of a Gaussian random variable
$\mathcal{N}(\mu, \sigma^2)$	a Gaussian probability density function with mean μ and variance σ^2

Table 1.1: Symbols

Chapter 2

Stochastic Model Predictive Control

This chapter contains an overview of Stochastic Model Predictive Control (SMPC), the control framework used in this thesis. It is a summary of the relevant background to aid the reader in the following chapters.

2.1 Dynamic Systems

At the core of most control algorithms is a model $\mathbf{h}(\cdot)$ that explains how the state $\mathbf{s}$ of a robot will change over time when a control input $\mathbf{u}$ is applied. In general, this relation is not known perfectly. The true system's state will evolve according to another function $\mathbf{h}^{true}(\cdot) \neq \mathbf{h}(\cdot)$. One way of representing this difference is to assume that the difference between $\mathbf{h}^{true}(\cdot)$ and $\mathbf{h}(\cdot)$ is a random variable $\boldsymbol{\eta}$. The model for robot dynamics must then include an estimate for the probability density function (pdf) of $\boldsymbol{\eta}$. Throughout this thesis, we will make two simplifying assumptions (among others that will be detailed in the upcoming chapters). First, we make the Markov assumption: that the state at the next sampling time only depends on the state and control input at the current time. Second, we assume that all probability density functions (pdfs) are part of the exponential family (most commonly the Gaussian distribution) to simplify our calculations. In discrete time, this means that the dynamics model can be expressed as:

$$\mathbf{s}_{k+1} = \mathbf{h}(\mathbf{s}_k, \mathbf{u}_k, \boldsymbol{\eta}_k), \tag{2.1}$$

where k is the index corresponding to the current sampling time, and $\boldsymbol{\eta}_k \sim \mathcal{N}(0, \boldsymbol{\Sigma}_k^2)$. The purpose of the dynamics model in an SMPC framework is to predict the pdf of future states, given an estimate of the state at the current time and a sequence of control inputs, so that the controller can choose a control input that minimizes a cost function (which

encodes the desired behaviour) while keeping the probability of violating constraints below an acceptable threshold (maintaining safety).

2.2 Stochastic Model Predictive Control

A control task for a mobile robot can often be expressed as minimizing tracking error while moving the robot along a path at a desired speed subject to constraints on the state and input. State constraints can come from obstacles in the environment and input constraints come from actuator limitations. Combined with uncertainty in the dynamics model, this problem naturally lends itself to SMPC. In SMPC, the state of the robot is treated as a random variable following a known distribution (e.g., Gaussian) and the probability of violating state and input constraints is kept below a certain threshold by keeping the relevant confidence intervals of that distribution inside the relevant constraints [Mesbah, 2016]. To be effective, this approach requires a means to *(i)* accurately predict the cost associated with a sequence of control inputs (to choose the optimal control input) and *(ii)* accurately estimate confidence intervals of the predicted state given a sequence of control inputs (to keep the probability of violating constraints at an acceptable level).

Ideally, the controller would directly minimize the cost of completing the entire task (e.g., driving the entire length of a path). However, since a task such as traversing a long path outdoors can take an arbitrarily long time and require an arbitrarily large number of control inputs, computing this sequence of control inputs can take an arbitrarily long time. Furthermore, disturbances from the environment and model inaccuracies can result in differences between the predicted and actual state. This can change the optimal control inputs for the remaining portion of the task. Therefore, the control inputs must be recomputed at a high rate to account for this. The most common approach to make this computationally feasible is to repeatedly solve for a fixed number of control inputs over a finite horizon, apply the first control input, and repeat this process at the next sampling time. This is known as Model Predictive Control (MPC) [Morari and Lee, 1999]. High frequency replanning in this way and has been shown to improve performance compared to using feedback control and a single, long-horizon plan [Sun et al., 2015].

2.2.1 Optimal Control Problem

We formulate SMPC as an optimisation problem where the prediction horizon has been discretised into H equally spaced timesteps. The robot state $\mathbf{s}$ evolves according to an

approximate model for robot dynamics $\mathbf{h}(\cdot)$ that depends on $\mathbf{s}$ and control inputs $\mathbf{u}$. Let the step cost $\ell(\mathbf{s}, \mathbf{u})$ be the non-negative scalar cost associated with applying $\mathbf{u}$ in state $\mathbf{s}$. The cost associated with the final state in the horizon $V_f(\mathbf{s})$ is the non-negative scalar cost to go from that state to the completion of the task. In addition, let $\mathbf{s}$ and $\mathbf{u}$ be constrained to be within the sets $\mathcal{S}$ and $\mathcal{U}$ with a small, acceptable probability of violating the state constraints $\boldsymbol{\epsilon}^{\mathbf{s}}$. The optimal states and inputs are computed by solving the following optimisation problem at each sampling time:

$$\underset{\mathbf{u}_{k+i},\, i=0,\ldots,H-1}{\arg\min} \quad V_f(\mathbf{s}_{k+H}) + \sum_{i=0}^{H-1} \ell(\mathbf{s}_{k+i}, \mathbf{u}_{k+i}), \tag{2.2}$$

$$\text{s.t.} \quad \mathbf{s}_{k+i+1} = \mathbf{h}(\mathbf{s}_{k+i}, \mathbf{u}_{k+i}, \boldsymbol{\eta}_{k+i}), \qquad i = 0 \ldots H-1, \tag{2.3}$$

$$p(\mathbf{s}_{k+i+1} \in \mathcal{S}) \geq 1 - \boldsymbol{\epsilon}^{\mathbf{s}}, \qquad i = 0 \ldots H-1, \tag{2.4}$$

$$p(\mathbf{s}_H) \in \mathcal{S}_f \geq 1 - \boldsymbol{\epsilon}^{\mathbf{s}}, \tag{2.5}$$

$$\mathbf{u}_{k+i} \in \mathcal{U}, \qquad i = 0 \ldots H-1, \tag{2.6}$$

$$\mathbf{s}_k \sim \mathcal{N}(\bar{\mathbf{s}}_k, \boldsymbol{\Sigma}_k^{\mathbf{ss}}), \tag{2.7}$$

where the state $\mathbf{s}_k$ is given by a localisation system, and model uncertainty is represented as a random, Gaussian variable $\boldsymbol{\eta}_{k+i} \sim \mathcal{N}(0, \boldsymbol{\Sigma}_{k+i}^{\boldsymbol{\eta}\boldsymbol{\eta}})$. The set $\mathcal{S}_f$ is known as the terminal set, and is used to guarantee recursive feasibility which will be discussed in Sec. 2.2.3. In general, the step cost $\ell(\cdot)$ can also depend on other variables including derivatives of the state and input. The control input can become a random variable if an ancillary controller is used, which will be described in Chapter 4. In this thesis, we will refer to a sampling time as an instant in time where a control input was applied (e.g., time k above) and timesteps as time instants along the prediction horizon (e.g., times i above). Since the state (and sometimes the input) are random variables, it is common to either optimise the expected value of (2.2) [Mesbah, 2016] or the cost evaluated at the mean states (and inputs, if they are random variables) [Ostafew et al., 2016, Kabzan et al., 2019]. In this thesis, we take the latter approach for simplicity since it still allows to study the impact of changes in the dynamics model.

2.2.2 Importance of the Dynamics Model

The dynamics model contributes to the optimisation problem in Section 2.2.1 in three important ways. First, it is used to predict future values of the state given a sequence of control inputs which determines the cost associated with that sequence of control inputs. Without accurate predictions of the state, the controller will be unable to choose control

inputs that minimise the cost function when they are applied to the real robotic system.

Second, the dynamics model includes an estimate of the model uncertainty, or the difference between the output of $\mathbf{h}(\cdot)$ and $\mathbf{h}^{true}(\cdot)$ at each point in their domain. This influences uncertainty in predictions about the future state given a sequence of control inputs, or how far the state of the true system is expected to stray from the predicted mean state for a given sequence of control inputs. This uncertainty is used to derive a safety margin between the predicted mean state and the constraints to ensure that the true system does not violate the constraints when inputs calculated based on the approximate model are applied. Without an accurate estimate of model uncertainty, the system will either leave too much safety margin (which may lead to sub-optimal behaviour) or too little (which may elevate the probability of constraint violation above the acceptable level).

Finally, the dynamics model contributes to the computational cost of solving the optimisation problem. The optimisation problem in Section 2.2.1 is usually solved as a sequential quadratic program, whereby the nonlinear dynamics model $\mathbf{h}(\cdot)$ is linearised about an initial guess for the optimal sequence of control inputs and gradient descent is used to refine this initial guess. This requires $\mathbf{h}(\cdot)$ to be evaluated many times for each control applied to the system. For example, suppose that a controller is running at 10 Hz, the prediction horizon is discretised into 30 time instants (i.e. $H = 30$) and the sequential quadratic program is solved three times. In this case, the mean, covariance, and Jacobian of the model will be evaluated 30 timesteps $\times$ 3 re-linearisations $\times$ 10 Hz $= 900$ times per second. This is in contrast to a regular feedback controller as feedback linearisation or PID, where the controller (a relatively simple function) would only be queried 10 times per second. This highlights the elevated importance of computational efficiency when designing models for SMPC.

In light of the critical role the dynamics model plays in safety and performance, this thesis will focus primarily on model learning techniques for SMPC enable the controller to operate safely and with high performance in changing operating conditions.

2.2.3 Recursive Feasibility

Recursive feasibility is the property that (2.2)–(2.7) remain feasible at the next sampling time after applying the first control in the optimal sequence from the finite-horizon problem. One way to do this is by using a terminal safe set $\mathcal{S}_f$, for which we have a controller $\pi_{safe}(\cdot)$ that guarantees that the system will remain within $\mathcal{S}$ by applying $\mathbf{u} \in \mathcal{U}$ for all time [Koller et al., 2018]. This could be, for example, a simple controller that can keep

the vehicle close to the path at low speed or an emergency braking controller that keeps the system in the path constraints while bringing the vehicle to a stop. The design of such controllers has been studied for nonlinear systems with bounded disturbances [Liu et al., 2016, Fisac et al., 2018, Fridovich-Keil et al., 2019]. Having such a controller means that given a sequence of controls from MPC, if $\mathbf{s}_H \in \mathcal{S}_f$, then if at any point no solution to (2.2)-(2.6) can be found, then we can keep the system within the original constraints by applying the remaining sequence of inputs from the previous MPC solution and then using $\pi_{safe}(\cdot)$ until (2.2)-(2.7) can be solved again. A detailed discussion of this approach for nonlinear systems assuming a safety controller may be found in Koller et al. [2018], and Rosolia et al. [2018] shows how to apply this approach to a linear time-invariant system where a safety controller may be derived more easily. A computationally expensive but general approach for generating π_{safe} can be found in Fisac et al. [2018]. For this paper, we found that using a sufficiently long prediction horizon meant that (2.2)-(2.7) remained feasible so we did not include a safety controller or terminal safe set.

2.2.4 Value Function or Terminal Cost

While recursive feasibility is about showing that a sequence of controls to keep the vehicle within the constraints past the prediction horizon *exists*, value function is about ensuring that the controller minimises the total cost to complete the task rather than myopically choosing inputs that lead to short-term gain at the expense of overall task performance. This is known as the terminal cost in controls [Rosolia et al., 2017a] and the value function in reinforcement learning [Koller et al., 2018].

While closed form solutions exist for simple scenarios such as infinite horizon LQR, finding the true value function or for a task is as difficult as finding the globally optimal control policy [Kober et al., 2013b]. For iterative tasks such as repetitive path following, a value function can be computed using the state-input sequences from previous traverses [Rosolia et al., 2017a]. Value function approximation has recently been combined with short-horizon trajectory optimisation to improve performance and prevent short-horizon trajectory optimisation from falling into local minima [Lowrey et al., 2018]. In cases such as repetitive path following where it is more difficult to get stuck in local minima (compared to navigating a maze [Lowrey et al., 2018], for example), good performance can be achieved without a value function [Ostafew et al., 2016]. This is the approach that we take in this thesis because it enables us to investigate changes in the dynamics model without the added complexity of estimating the value function. To differentiate between the value function $V_f(\mathbf{s})$ and the stage cost associated with the final state in the horizon, we denote the latter as $\ell(\mathbf{s}_H)$.

2.2.5 Simplified SMPC Formulation

Neglecting the terminal cost, terminal set, and calculating the step cost over the horizon using the mean states for the reasons stated above, we arrive at the simplified SMPC formulation used in this thesis. Let $\ell_f(\mathbf{s})$ be the non-negative scalar cost of the final state in the horizon. This is not the cost-to-go from the final state to the end of the path, rather the step cost $\ell(\cdot)$ without the components related to the control input. The resulting receding horizon control problem solved at each sampling time k is then:

$$\min_{\mathbf{u}_{k+i}, i=0..H-1} \quad \ell_f(\bar{\mathbf{s}}_{k+H}) + \sum_{i=0}^{H-1} \ell(\bar{\mathbf{s}}_{k+i}, \mathbf{u}_{k+i}) \tag{2.8}$$

$$\text{s.t.} \quad \mathbf{s}_{k+i+1} = \mathbf{h}(\mathbf{s}_{k+i}, \mathbf{u}_{k+i}, \boldsymbol{\eta}_{k+i}), \qquad i = 0...H-1, \tag{2.9}$$

$$p(\mathbf{s}_{k+i+1} \in \mathcal{S}) \geq 1 - \epsilon^{\mathbf{s}}, \qquad i = 0...H-1, \tag{2.10}$$

$$\mathbf{u}_{k+i} \in \mathcal{U}, \qquad i = 0...H-1, \tag{2.11}$$

$$\mathbf{s}_k \sim \mathcal{N}(\bar{\mathbf{s}}_k, \boldsymbol{\Sigma}_k^{\mathbf{ss}}), \tag{2.12}$$

Terms involving higher derivatives of the state and input may be included in the cost function to encourage smoother motions and control inputs. When an ancillary controller is used and the control input becomes a random variable, the mean value of the input is used to calculate the cost and the input constraints becomes probabilistic chance constraints, like the state constraints, (2.10).

Experimental Platform

All of the closed-loop experiments in this thesis were conducted on a laptop with an Intel i7 2.70 GHz 8 core processor with 16 GB of RAM. The optimisation problem above was solved using CPLEX [IBM]. Localisation was provided by Visual Teach and Repeat [Paton et al., 2017], which uses a single stereo camera for localisation and runs on the same laptop as the controller. Closed-loop experiments were conducted on a Clearpath Grizzly.

2.3 Summary

This chapter presented a high-level overview of the predictive control framework that this thesis expands upon, primarily by developing new model-learning methods suited to this approach. We emphasise the system's reliance on an accurate model for robot dynamics to predict the cost associated with a sequence of control inputs as well as to estimate

uncertainty in predicted states to ensure the task is completed safely. Furthermore, we identify simplifications made for the purpose of studying the dynamics model. These simplifications are based on Ostafew et al. [2016], which was the starting point for this thesis.

In the following chapters, we present the current state of the art, show its limitations, and present novel model learning techniques to address their limitations. We remind the readers that this chapter provides background information only and presents no novel contributions.

Chapter 3

Gaussian Process-Based Methods

This chapter contains the methods developed in this thesis that rely on Gaussian Process (GP) regression to learn the unknown dynamics. Section 3.1 provides background on GPs and an overview of related work. Section 3.2 presents the first novel method to automatically learn multiple GP models to adapt to large changes in robot dynamics. Section 3.3 presents a follow-up method to reduce the computational cost of the models learned in the previous section while still retaining the ability to adapt to large changes in dynamics. Section 3.4 summarises the conclusions and contributions from this chapter.

3.1 Introduction

The most common way to improve the performance of SMPC is to leverage data to refine the dynamics model. Most model learning algorithms use a single model for system dynamics or multiple models that are trained ahead of time based on appropriate training data from operating the robot in all relevant conditions. This presents a challenge for robots that are deployed into changing operating conditions which may not be known ahead of time. In this chapter, we present two methods for modelling robot dynamics in changing conditions that are based on GP regression.

The first method presented in this chapter is based on constructing a GP to model the robot dynamics in each discrete mode that a robot may encounter. A mode may represent, for example, a ground robot with a specified payload large enough to change the dynamics. When a new mode is encountered, the method will switch to a 'safe mode' until a new GP can be constructed to model the robot dynamics in the new mode. We show experimental results to demonstrate that this method was effective for generating new models when new modes were encountered and recycling existing models when the dynamics were similar to a previously observed mode. The primary limitation of this

method is that the GPs required to represent the robot dynamics in each mode are too computationally expensive to use in a controller like SMPC, which requires the dynamics model to be evaluated many times at each sampling time. To address this limitation, we developed a second method that we were able to demonstrate in closed-loop experiments.

The second method presented in this chapter also uses a GP to model the robots' dynamics; however, we make the additional assumption that the robot is performing a repetitive path-following task. This makes it possible to learn a local model instead of a global one, which is more computationally efficient. Instead of keeping a library of GP models, one for each mode as our first approach, this method dynamically constructs a local GP using data from previous traversals of the path where the dynamics were similar to the robots dynamics during the current traversal. This method has the same properties that it is able to leverage past experience from multiple modes and revert to a 'safe mode' when no past data is available that matches the robot's current dynamics.

The remainder of this section will introduce the GP regression method used in both of the approaches described above. The following sections describe the novel methods generated as part of this thesis.

3.1.1 Gaussian Process Regression

In modelling robot dynamics, our goal is to make accurate predictions about the robot's motion given its state and input while providing a well-calibrated estimate of uncertainty in the difference between the output of our model and the actual motion of the robot. When discussing model uncertainty, it is helpful to distinguish between two components of model uncertainty [Der Kiureghian and Ditlevsen, 2009]. First, *epistemic* uncertainty is uncertainty that can be reduced by gathering more data. For a parametric model, this is uncertainty about the particular values of the parameters in the model. Second, *aleatoric* uncertainty is the irreducible component that has to do with factors that cannot be captured by our choice of model. Using expressive models such that more of the initial model uncertainty is epistemic and thus reducible given the appropriate training data is usually beneficial. However, it is always important to have an accurate estimate of the total model uncertainty at any time.

In realistic scenarios, there will always be an aleatoric component, either due to lack of knowledge of the system or simplifications necessary for reasons such as computational tractability. Gaussian Process regression is a Bayesian, non-parametric approach for modelling functions in this context, and is popular in the robotics community, in part, since it can be used to model nonlinear functions and its parameters can be deter-

mined largely from data. Since there are many good references on GPs [Rasmussen and Williams, 2006], we provide only a high-level overview.

A GP models functions of the form:

$$g(\mathbf{x}) = \mu(\mathbf{x}) + \eta, \tag{3.1}$$

where $\eta \sim \mathcal{N}(0, \sigma^2)$ and $\mu(\cdot)$ is a potentially nonlinear, deterministic function. A GP is a distribution over functions given past data $\mathcal{D} = \{\mathbf{x}_i, g_i\}_{i=1}^n$, a kernel, and kernel hyperparameters. The posterior distribution is characterized by a mean and variance, which can be queried at any point $\mathbf{x}_*$ using:

$$\mu(\mathbf{x}_*) = \mathbf{k}^T(\mathbf{x}_*)\mathbf{K}^{-1}\mathbf{g}, \tag{3.2}$$

$$\sigma^2(\mathbf{x}_*) = \kappa(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}^T(\mathbf{x}_*)\mathbf{K}^{-1}\mathbf{k}(\mathbf{x}_*), \tag{3.3}$$

where $\mathbf{g}$ is a column vector of g_i, the matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ has entries $\mathbf{K}_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, where $\kappa(\cdot)$ is the kernel function. The column vector $\mathbf{k}(\mathbf{x}_*) = [\kappa(\mathbf{x}_*, \mathbf{x}_1), ..., \kappa(\mathbf{x}_*, \mathbf{x}_n)]^T$ contains the covariances between the new test point $\mathbf{x}_*$ and the observed data points in $\mathcal{D}$. For this work, we use the squared exponential kernel:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}||\mathbf{x}_i - \mathbf{x}_j||^2_{\mathbf{M}^{-2}}\right) + \sigma_\eta^2 \delta_{ij}, \tag{3.4}$$

with δ_{ij} being the Kronecker delta, because of its success in modelling robot dynamics [Ostafew et al., 2016, Bouffard et al., 2012]. The parameters of the kernel, or hyperparameters, are the diagonal matrix $\mathbf{M}$ with so-called length-scales on the diagonal, which are inversely related to the importance of each element of $\mathbf{x}$, the variance of epistemic uncertainty σ_f^2, which is the variance of the prior family of functions represented by the GP, and the variance of aleatoric uncertainty σ_η^2. As training data is added to a GP, epistemic uncertainty is reduced and the posterior distribution of the GP specializes to a particular family of functions that represents the underlying function. An example of functions drawn from the prior and posterior of a GP is shown in Fig. 3.1.

Hyper-parameters can be estimated by maximising the marginal likelihood of the training data. This objective can contain multiple local minima (See Murphy [2012], Fig. 15.5). Since hyperparameter optimisation is often done using gradient descent, it must be repeated multiple times from different initial conditions and validated against a separate partition of the data.

Although it is assumed that the underlying function is deterministic and that we receive noisy observations with a constant noise variance σ_η^2, the predicted variance (3.3)

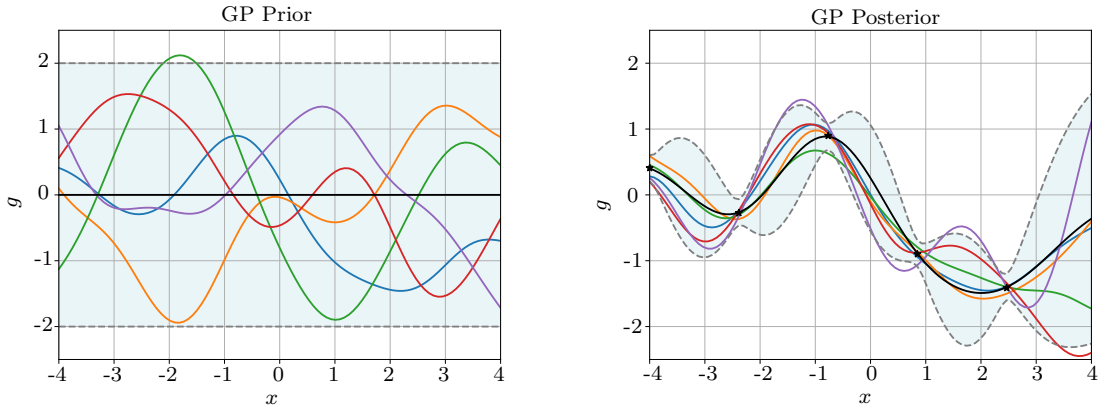


Figure 3.1: Example of a GP with a squared exponential kernel (3.4) when $\sigma_\eta = 0.1, \sigma_f = 1.0$ and the lengthscale is 1. Left: The GP prior and samples of functions drawn from the prior with the 2σ confidence interval shaded in blue. Right: The GP posterior after conditioning on the points shown in black and samples of functions drawn from this posterior. The posterior distribution has concentrated around the observations but residual uncertainty remains because $\sigma_\eta > 0$.

depends on $\mathbf{x}_*$ because uncertainty about the underlying function, or the epistemic uncertainty, depends on the distribution of training data.

3.1.2 Related Work

Learning control has received a great amount of attention in recent years, most notably in the case of single-mode learning control. This is the broad class of learning methods that assumes the true (but initially unknown) robot dynamics will only generate one motion for a given state and control input. Recent developments have contributed safety guarantees [Aswani et al., 2013] and demonstrated impressive results in improved path following [Ostafew et al., 2014a]. Approaches by Ostafew et al. [2014a, 2016], Gillula and Tomlin [2012], Bouffard et al. [2012] and Mahler et al. [2014] use GPs as corrective terms for approximate prior models and update them over time as more experience is gathered.

GPs have been applied to problems such as modelling the dynamics of a chemical reactor [Kocijan et al., 2004], a robot performing repetitive path following [McKinnon and Schoellig, 2018], model car racing [Hewing et al., 2018a], and for a manipulator robot performing a tracking task [Ting et al., 2008]. While a naive implementation of a GP can be computationally intractable since it scales poorly in the number of training points, several solutions have been proposed including local GPs [Meier et al., 2016, F. Meier and S. Schaal, 2016, Lederer et al., 2020, Nguyen-Tuong et al., 2009a], sparse GPs [Hewing et al., 2018a], and GPs using a special set of basis functions [Desaraju et al.,

2017]. These alternatives make GPs computationally tractable in a control loop, even for controllers such as stochastic MPC which require evaluating the model for the robot dynamics many times at each timestep. In addition to being computationally feasible for stochastic MPC, GP-based models have been extended to learn the dynamics of a robot subjected to changes not seen in the training data [McKinnon and Schoellig, 2018]. Comparisons have shown that GPs can achieve higher accuracy and good prediction speed compared to local linear regression when GP approximations are used. These comparisons and the work using GPs for modelling robot dynamics typically consider fairly controlled environments such as a manipulator performing a tracking task [Nguyen-Tuong et al., 2009b] or a ground robot in a parking lot [McKinnon and Schoellig, 2018]. While there are notable exceptions, e.g. Ostafew et al. [2016], where a ground robot was driven over challenging outdoor terrain, there has been no comparison to local linear regression in this case.

One GP-based model that exhibits especially good real-time performance and has been demonstrated in several real-world examples is presented in Ostafew et al. [2016]. This approach continually reconstructs the GP disturbance model based on a fixed number of data points, to ensure the process model can be evaluated in constant time even if new experience is added. Storing the data in first-in-first-out bins of fixed size allows the algorithm to update the data used in the GP in real time. In addition, the application was for repetitive path-following, so including a dependence on distance along the path in the dynamics model enabled this approach to learn different dynamics at each point along the path. This accounts for changes in dynamics due to factors such as the terrain which can affect the dynamics differently at each point along the path but do not change substantially from one run to the next. In contrast, changes caused by factors such as a large payload do not depend on the robots location along the path so can still introduce multiple modes in the dynamics. If the mode changes, the model must un-learn the existing mode by gradually over-writing the data stored in the first-in-first-out bins from the previous mode with data from the new mode. During this process, it suffers from the same problems related to hyper-parameters as mentioned above. This means either requiring over-conservative bounds on model error to accommodate multiple modes or have bounds on model error that are realistic for a single-mode but are optimistic (unsafe) while the model transitions between modes and is using data from more than one mode. The methods presented in this chapter aims to overcome this limitation by learning a separate model for each distinct mode or only selecting experiences from runs where the robots dynamics were similar to the robots current dynamics.

In addition to the single-mode, safe learning controllers, multimodal algorithms exist

which identify a number of dynamic modes ahead of time using labelled or unlabelled training data and switch to the most likely mode during operation [Jo et al., 2012, Luders et al., 2013, Aoude et al., 2013, Calandra et al., 2015a]. This allows them to maintain persistent knowledge of a robots dynamics across a wide range of operating conditions. Inferring the correct mode from measurements during operation allows them to maintain a high level of performance and robustness even when the mode is not directly observed. The method proposed in Fox et al. [2009] for linear systems even infers the number of modes at training time. These approaches do, however, require that the number of modes and/or training data from each mode be available ahead of time, which can be a challenging task in robotics. In contrast, the methods presented in this chapter do not require the number of modes or training data from each mode to be available ahead of time. Rather, new modes will be learned as they are encountered at runtime.

The first method described in this chapter is based on combining GPs and the Dirichlet Process (DP), which is used in Bayesian non-parametric clustering models. GPs have been combined with DPs before to obtain a powerful regression tool [Rasmussen and Ghahramani, 2002]. In Rasmussen and Ghahramani [2002], the posterior is the distribution corresponding to every possible assignment of data points to experts; therefore the likelihood is a sum over (exponentially many) assignments which must be evaluated by sampling. This was an offline method and not designed to be tractable in real-time for a robotics application. The first approach presented in this chapter will improve in computational efficiency by assigning experiences to only one expert which eliminates the need to sum over possible assignments.

3.2 Learning Multi-Modal Models for Robot Dynamics with a Mixture of Gaussian Process Experts

3.2.1 Introduction

This section presents a method to model the dynamics of robotic systems where the dynamics may be subject to large changes that depend on discrete latent variables. The proposed method is shown in combination with a controller in the block diagram in Fig. 3.2. These latent variables reflects a discrete set of operating conditions or physical configurations (dynamic modes) of the robot that change the dynamics. Examples are weather or terrain conditions, or payload configurations. The proposed method uses a Dirichlet Process (DP) to represent the dynamic modes of the system in a way that does not require the number of modes to be specified ahead of time, and Gaussian Process

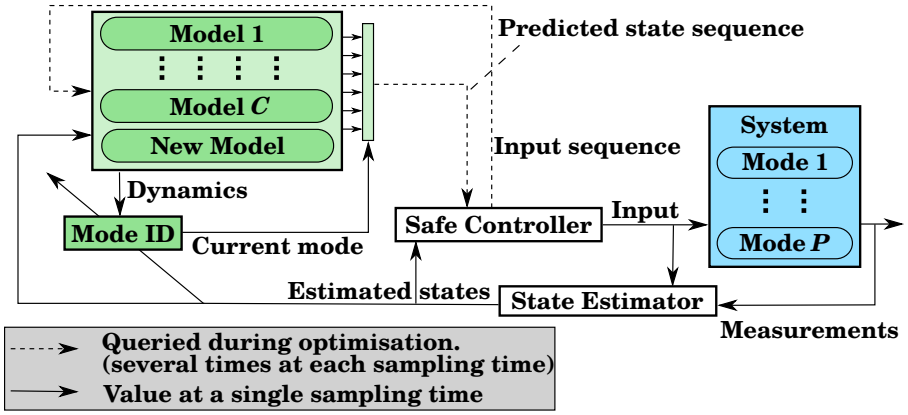


Figure 3.2: Block diagram showing the proposed multimodal model learning in closed-loop with a safe controller. The robotic system dynamics consist of P distinct modes depending on the operating conditions (blue). Our algorithm (green) learns a multimodal model for the system dynamics, and selects the correct model based on recent measurements at run time. The diagonal arrow indicates that a recent history of data is used. The proposed model learning scheme is designed for a safe controller such as the one presented in Ostafew et al. [2016].

(GP) experts to learn the dynamics of the robot in each mode while making only mild, prior assumptions on the robots dynamics in each mode. The GP experts naturally express the uncertainty in the dynamics in regions of the state-input space depending on whether they have been visited before. The DP allows the model to return to a safe mode when the current set of models does not explain current measurements until a new model is established. The result is a mixture model that learns a new model when the current set of models is insufficient to explain current measurements and returns to an existing model when possible. It does not 'forget' the dynamics in previously learned modes when learning new ones which is a significant advantage over previous methods.

3.2.2 Problem Statement

The goal of this work is to learn a dynamic model from data that can predict the future states for a nonlinear, switching dynamic system where the number of modes and dynamics in each mode are not known ahead of time. The algorithm should learn new models when new modes are encountered, and improve existing models when modes are re-visited. The model should also include a reasonable estimate of model uncertainty that acts as an upper bound on model error at all times.

Further assumptions can be summarized as follows:

- The dynamics in each mode can be modelled as a GP with fixed hyper-parameters.
- Data is available to identify the GP hyper-parameters ahead of time.

- The number of modes and the specific dynamics in each mode are not known ahead of time.
- The mode is constant over a short time horizon.

A short time horizon could be similar to the horizon considered for Model Predictive Control (MPC).

We consider systems with dynamics of the form:

$$\mathbf{s}_{k+1} = \mathbf{h}(\mathbf{s}_k, \boldsymbol{\xi}_k) \tag{3.5}$$

$$\boldsymbol{\xi}_{k+1} = \mathbf{g}_0(\boldsymbol{\xi}_k, \mathbf{u}_k) + \mathbf{g}^c(\mathbf{x}_k), \tag{3.6}$$

where we have partitioned the state into $\mathbf{s}$, the elements of the state that have known dynamics $\mathbf{h}(\mathbf{s}_k, \boldsymbol{\xi}_k)$, and $\boldsymbol{\xi}$, the elements of the state that have unknown dynamics. The unknown dynamics are the sum of a known component $\mathbf{g}_0(\boldsymbol{\xi}_k, \mathbf{u}_k)$ and an unknown component $\mathbf{g}^c(\mathbf{x}_k)$. The unknown component is specific to a discrete mode c, depends on features $\mathbf{x}_k$, and is assumed to be a deterministic function with additive, zero-mean, Gaussian noise:

$$\mathbf{g}^c(\mathbf{x}) = \boldsymbol{\mu}^c(\mathbf{x}) + \boldsymbol{\eta}^c, \tag{3.7}$$

where $\boldsymbol{\eta}^c \sim \mathcal{N}(0, \boldsymbol{\Sigma}_n^c)$.

3.2.3 Methodology

In this section, we present our approach to modelling systems with multimodal dynamics using a combination of GPs with the Dirichlet Process (DP). We will consider the case when $g^c(\mathbf{x})$ is scalar. Our approach can be extended to vector $\mathbf{g}^c(\mathbf{x})$ by using a separate GP to model each element of the unknown dynamics as in Ostafew et al. [2016].

Dirichlet Process Mixture of Gaussian Process Experts

The goal is to learn the unknown dynamics, $g^c(\mathbf{x})$, for each dynamic mode from data, and automatically detect the relevant mode or create a new model if necessary. To do this, we propose is a Dirichlet Process Gaussian Process Mixture Model (DPGPMM). The DP is used to learn the number of dynamic modes and the GP is used to model the error between the dynamics of the real system and a prior model in each mode. There are four key properties of this model that make it well suited for modelling the dynamics of multi-modal systems for SMPC:

1. The ability to handle an increasing number modes (DP);

2. The definition of a 'safe mode' when no data is available or there is a poor fit (DP & GP);

3. The quantitative bounds for the model error (GP); and

4. The possibility to improve the model over time (GP).

3.2.4 Gaussian Process Disturbance Model

We model the unknown dynamics $g(\cdot)$ as a GP based on past observations. As training data is added to a particular GP, uncertainty is reduced and the posterior distribution of the GP specializes to a particular family of functions which represents the system dynamics in a particular mode. The DP is then used as a distribution over these families of functions where each family, represented by a GP, is the system dynamics in a particular mode. See Sec. 3.1.1 for a description of the Gaussian Process Regression used for this section.

3.2.5 Dirichlet Process

For the DPGPMM, the DP acts as a distribution over modes, which assumes the number of modes is infinite. In reality, however, only a small number of modes will actually have data. Suppose there are C modes and let $\mathbf{c} = (1, .., C)$ be the vector of indicator variables for the modes. The conditional probability of each mode c when integrating over all possible modes is then

$$p(c = j|\mathbf{c}, \alpha) = \frac{n_j}{N + \alpha} \quad \text{for existing modes} \tag{3.8}$$

$$p(c = C + 1|\mathbf{c}, \alpha) = \frac{\alpha}{N + \alpha} \quad \text{for new modes} \tag{3.9}$$

where $j \in \{1, ..., C\}$ and n_j is the number of points in expert j, N is the total number of data points in the *existing* experts (i.e. $N = \sum_j n_j$), and α is a parameter of the DP called the concentration parameter, which controls the prior probability of new modes [Rasmussen and Ghahramani, 2002]. We have used $\mathbf{c}$ as a shorthand to indicate the existing mixture model, which includes the GP associated with each mode and hence the number of points in that GP.

The important properties of the DP are that modes with more experiences are more likely, and the model always includes an element for a new mode, which is the GP with no data, or the GP prior. The GP prior has the largest variance which results in the most conservative bounds on the disturbance and thus acts as a 'safe' mode.

3.2.6 Mode Inference

During deployment, the goal is to find the best estimate of the current and future model error given all past experiences. We use a recent history of n_w experiences, $\mathcal{D}^- = \{g_i, \mathbf{x}_i\}_{i=k-n_w}^{k}$, to infer the mode at the current timestep, k, and use the most likely mode to predict the model error at future timesteps. We assume the mode is constant over short time periods so all samples in $\mathcal{D}^-$ should come from the same mode. The posterior probability of the j^{th} mode is:

$$p(c = j|\mathcal{D}^-, \mathbf{c}) \propto p(\mathcal{D}^-|c = j)p(c = j|\mathbf{c}, \alpha). \tag{3.10}$$

where $p(c = j|\mathbf{c}, \alpha)$ is the prior probability of mode j calculated using (3.8) or (3.9), and $p(\mathcal{D}^-|c = j)$ is the probability of recent experiences under mode $c = j$:

$$p(\mathcal{D}^-|c = j) = \prod_{i=k-n_w}^{k} p(g_i|\mu^{c_j}(\mathbf{x}_i), \sigma^{c_j}(\mathbf{x}_i)), \tag{3.11}$$

where $\mu^{c_j}(\mathbf{x}_i)$ and $\sigma^{c_j}(\mathbf{x}_i)$ are the mean and standard deviation of the GP representing mode j evaluated at $\mathbf{x}_i$.

A new cluster is created when the probability of $c = C + 1$ is larger than any expert given recent data. Since it is computationally expensive to create a new GP model, which involves inverting an $n_j \times n_j$ matrix, we remain in the safe mode until the model returns to an existing mode. While in the existing mode, the system uses experience gathered while it was in the safe mode to create a new GP model. This approach assumes that the system does not transition between two unknown modes before transitioning back to an existing mode. This was inspired by Linegar et al. [2015] which is about managing experiences for visual navigation in visually changing outdoor environments.

3.2.7 Ground Robot Model

We demonstrate our multimodal learning approach in experiment on a ground robot, namely the Clearpath Husky (see Fig. 3.3). This section outlines the choice of *apriori* model.

We use a similar apriori model to Ostafew et al. [2014a], which uses a unicycle model, but we include a term for translation along the body y-axis ρ_k and use first-order dynamics with a time delay for the forward speed v and the turn rate ω. The addition of the y-component in velocity is to include a learning term to account primarily for offsets between the pivot point and the origin of the body frame. The proposed dynamics

model for the states with known dynamics is:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + dt \begin{bmatrix} \cos\theta_k & -\sin\theta_k & 0 \\ \sin\theta_k & \cos\theta_k & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_k \\ \rho_k \\ \omega_k \end{bmatrix}. \tag{3.12}$$

Through preliminary experiments, we found that the translational and rotational dynamics can be reasonably well approximated by a first order system with time-delay, so we use a first order prior model and a learning term to model the additional, unknown dynamics:

$$\begin{bmatrix} v_{k+1} \\ \rho_{k+1} \\ \omega_{k+1} \end{bmatrix} = \begin{bmatrix} v_k \\ \rho_k \\ \omega_k \end{bmatrix} + dt \begin{bmatrix} \frac{1}{\tau_1}(v^{cmd}_{k-d} - v_k) \\ 0 \\ \frac{1}{\tau_2}(\omega^{cmd}_{k-d} - \omega_k) \end{bmatrix} + \begin{bmatrix} g^c_v(\mathbf{x}_k) \\ g^c_\rho(\mathbf{x}_k) \\ g^c_\omega(\mathbf{x}_k) \end{bmatrix} \tag{3.13}$$

where τ_1 and τ_2 are the time constants for the translational and rotational dynamics, d is the number of timesteps of the delay, v^{cmd} is the commanded forward speed, ω^{cmd} is the commanded turn rate, and $\mathbf{x}$ is the disturbance dependency, which will be defined in Sec. 3.2.8.

Without the learning terms, (3.12)–(3.13) assume that the body frame is centred on the center of rotation of the vehicle and that the velocity and turn rate response are fixed and match the given first order system with fixed parameters. In reality, these assumptions can be violated if the center of mass is offset due to a heavy payload which changes the pivot point, or if a heavy payload changes the turn-rate or velocity response. Many other factors including but not limited to terrain type and tire pressure can also change the dynamics. The goal of the learning term is to correct for these factors using data generated when the robot drives in the relevant conditions.

3.2.8 Experiments

Experimental Setup

Experiments were conducted on a 50 kg Clearpath Husky skid-steer ground robot shown in Fig. 3.3. Weights and tire pressure were varied to produce five dynamic modes while the vehicle was driven manually on a painted concrete surface. The *no mass configuration* was with no mass and high tire pressure. The *centred configuration* was with a 25 lb weight on the front bumper and a 35 lb weight on the rear bumper. The *offset configuration* was tested with a 35 lb weight on the rear bumper and a 25 lb weight

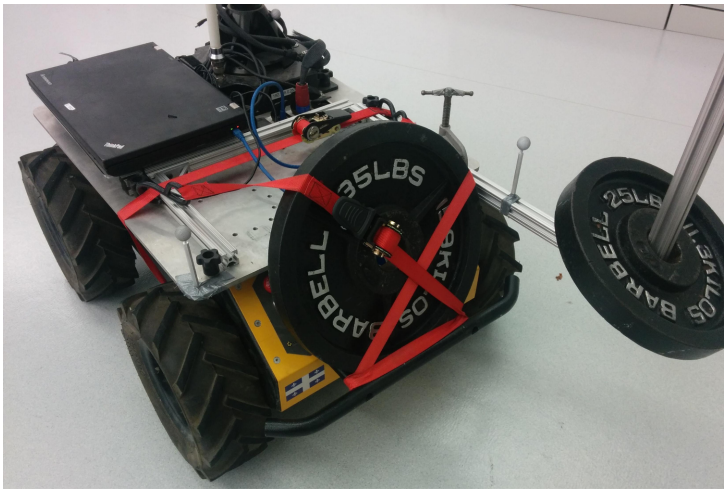


Figure 3.3: The Clearpath Husky robot with additional weights used for experiments. The configuration shown changes the mass and pivot point of the vehicle which drastically changes its rotational dynamics. Different loading configurations result in different dynamics. Our approach detects which dynamic mode is currently active and learns a new model when the existing library of dynamic models is insufficient to explain current measurements.

on an arm extended beyond the body (depicted in Fig. 3.3). The centred and offset configurations were tested with *high and low tire pressure*. Four trials were conducted in each configuration, where the vehicle was driven manually for a total of 1347 s. The motion of the vehicle was captured using a Vicon motion capture system at 200 Hz, and commands were sent to the vehicle at 10 Hz. Below, we focus on the rotational dynamics since the added mass did not have a significant effect on the forward speed dynamics.

Experiences

The learned model depends on observations of prediction error gathered during previous trials. Experiences are calculated using (3.13) and measurements of the translational and angular velocity in the body frame, (v_k, ρ_k, ω_k). We down-sampled measurements from the motion capture system taking the most recent pose measurement for each timestep where a new command was applied.

Choosing the correct dependence for the disturbance is an important and challenging design task. In our experiments, we chose $\mathbf{x}_k = (v_k, \rho_k, \omega_k, \omega_k^{cmd}, \omega_{k-1}^{cmd}, \omega_{k-2}^{cmd})$. This choice for $\mathbf{x}$ assumes that all disturbances act in the body frame because the feature is composed of elements that are aligned with the body frame. This assumption was made based on the intuition that disturbances come from varied interaction between the wheels and the surface; since the wheels are fixed in the body frame, the unmodelled dynamics should be as well. Including several inputs was motivated by an obvious time delay in the vehicles

	centred	centred, low	no mass	offset	offset, low	safe
centred	**0.49**	**0.06**	**0.34**	0.02	0.03	0.06
centred, low	**0.21**	**0.43**	**0.27**	0.00	0.01	0.08
no mass	**0.42**	**0.20**	**0.26**	0.01	0.01	0.10
offset	0.01	0.01	0.01	**0.88**	0.02	0.07
offset, low	0.00	0.02	0.04	0.08	**0.78**	0.08

Table 3.1: The confusion matrix for mode classification based on models trained using the true labels. Coloured boxes indicate modes with similar dynamics.

behaviour of two to three timesteps. Using this approach, the model can learn systematic time delays in the dynamics in the range from one to three timesteps.

Tuning Parameters

Hyper-parameters for the GP were trained using data from the configuration with no mass and fixed thereafter. For this work, we used version 1.0.9 of the GPy package [GPy, since 2012]. The maximum number of points in a GP was fixed to 1000. The concentration parameter of the DP, α, was set to 1. This means that with no other indicative measurements, an existing mode is far more likely than a new mode. The prior function variance, $\sigma_f^2 = 0.25^2$, was chosen as an upper bound on the expected disturbances. The noise variance, $\sigma_\eta^2 = 0.05^2$, was chosen as the upper bound on measurement noise, and the kernel length-scales, with diagonal elements $diag(\mathbf{L}) = (1.48, 685, 0.18, 800, 0.69, 0.64)$ were optimized using training data collected with no mass on the Husky. The large length-scales for ρ_k and ω_k^{cmd} indicates that the GP has learned these elements are not important.

Mode Inference Given Fixed Models

First, we demonstrate the mode inference given a fixed number of models trained using known modes. Four runs of about 90 s were conducted in each mode and one was used for training. The confusion matrix in Table 3.1 shows that classification errors occur primarily between the modes in the first three columns, where the center of mass of the Husky is roughly over the center of the wheels. This suggests that these modes have similar dynamics which matches our observations and means that experience from one of these modes may be relevant to another. In addition, a classification error in these cases would be of little consequence since the predictions made using any of these models will be similar.

Model Learning Example

Figure 3.4 demonstrates how the algorithm learns new models safely and efficiently when confronted with novel configurations. The model was initialized with data from the *no mass* configuration; that is, initially it has no experience related to the *offset* configuration. The mode identification used the most recent 2 s of data to identify the current mode. When it started moving at 9 s and encountered measurements from the *offset* configuration, it reverted to the safe mode. During this time, the learning term, $g_w^c(\mathbf{x}_k)$, is approximately a zero-mean Gaussian with a large variance, which keeps the measurements within 3σ (shaded in red) of the mean (red line). At 18 s, it switches to the learned mode and constantly detects the new mode except when the vehicle is stationary again at 45 s. At this point, all modes predict that vehicle remains stationary so the DP is dominant and the mode with the largest number of points, the initial model trained for no mass, is selected. The model remains in this configuration until 141 s, since the *no mass* configuration is very similar to the *centred* configuration. At 141 s when the Husky is commanded to rotate again, the model switches back to the previously learned model for *offset*, leveraging previous experiences. This demonstrates how the proposed approach safely learns a new model and makes efficient use of experiences by constantly improving existing models.

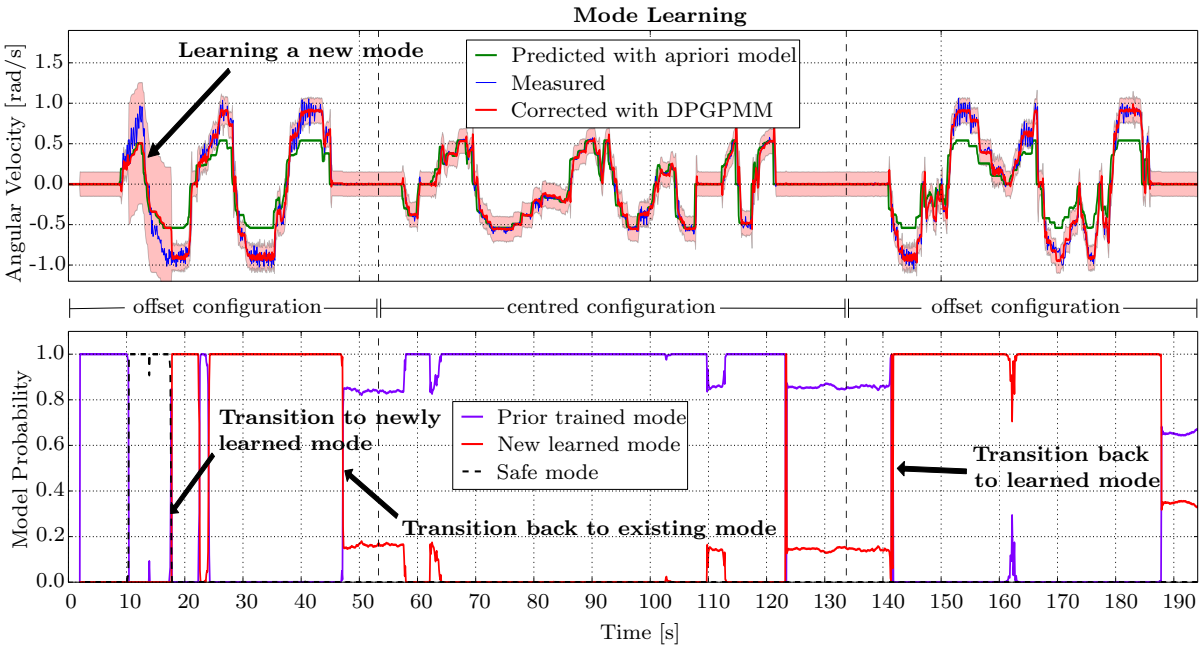


Figure 3.4: During this experiment, the Husky was deployed in three different configurations. First, the offset configuration (0-53 s), then the centred configuration (53-134 s), then the offset configuration again (134 s-194 s). The DPGPMM was initialized with a GP pre-trained using data in the no mass configuration. From 10-18 s, the model cannot explain experiences from the offset configuration, so it is in safe mode, which results in a large uncertainty. At 18 s, the model makes a classification error and switches to its initial mode which triggers the creation of a new learned mode. Immediately after, it switches to the new mode and continues improving this mode until 47 s where it switches back to the prior mode. At this time, the vehicle is stationary, so both models explain the motion well but the initial mode contains more points so it switches to this mode. At 141 s, the DPGPMM switches back to the newly learned mode since it has already learned a model for this configuration, so does not go into safe mode first.

Model Prediction Performance

The method presented in this paper is aimed at improving performance and safety of safe learning controllers such as the one presented in Ostafew et al. [2016]. This safe controller relies on an accurate prediction of the mean state and modelling error bounds over a short time horizon given the current state and a series of inputs, see Fig. 4.1. In our experiments, the Husky ground robot was driven manually. This gives us a series of states and inputs. We measure prediction accuracy of the mean states over the a short horizon of H timesteps using Multi-step RMSE (M-RMSE) and of the uncertainty using multi-step RMS Z-score (M-RMSZ). The M-RMSE at sampling time k is calculated by comparing the predicted mean at each timestep i along the horizon $\mu_{k,i}$ to the measured value at the corresponding sample time $g_{k+i,0}$:

$$\text{M-RMSE}_k = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(\mu_{k,i} - g_{k+i,0})^2}, \tag{3.14}$$

$$\text{M-RMSZ}_k = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\frac{(\mu_{k,i} - g_{k+i,0})^2}{\sigma_{k,i}^2}}. \tag{3.15}$$

M-RMSZ is calculated the same way but each term in the sum is normalized by the predicted variance at that timestep. Ideally, M-RMSE would be low and M-RMSZ would be around one.

Figure 3.5 shows a comparison of the RMS error over 16 trials using three different approaches. First, we use a single GP initialized with data from the no mass configuration. Second, we train a supervised mixture of GPs using perfectly labelled data from all configurations (one for no mass, one for centred, one for offset, and one for offset with low tire pressure) and manually label which mode is active at run time. This model also acts as a measure of the limit of accuracy of the GP with fixed hyper-parameters and perfect mode inference. Third, the proposed DPGPMM is initialized with one GP trained using data from the no mass configuration and learns the remaining modes during the experiment. The algorithm for updating GPs to maintain a constant size is based on continually choosing a random Subset Of Data (SOD) associated with the model. This was chosen for simplicity and good performance relative to other SOD methods [Rasmussen and Williams, 2006].

The results in Fig. 3.5 show how the proposed method quickly approaches the performance of the mixture model trained with perfectly labelled data, and retains this performance regardless of mode switches. The GP mixture trained using labelled data rep-

resents a baseline for 'good' performance using a GP model with fixed hyper-parameters. The single GP can learn to approximate one set of dynamics after a long period of time, but must un-learn and re-learn dynamics each time it encounters a new mode resulting in consistently higher RMSE and RMSZ. During the first run, the proposed method was initially uncertain about the dynamics of the robot in the offset configuration which resulted in higher uncertainty which reduces the RMSZ despite having a higher RMSE (circled in red). This is a good indication of safe performance despite high, initial model error. For runs 8-10, the Husky was in either a centred or no mass configuration. After three repeated runs with the same configuration, the single GP adapts to the dynamics in this mode (circled in purple). On subsequent runs, however, it has unlearned the dynamics in the offset configuration which results in dramatically increased RMSE and RMSZ while the proposed method remains close to the ideal model.

Using a supervised GP mixture model in practice is difficult because it requires perfect training data to be available ahead of time. This is not always possible. Moreover, such a model does not adapt to slight changes between modes (e.g., caused by tire pressure) while the proposed method does. Finally, the pre-trained models do not leverage the fact that dynamics may be similar for multiple modes which may result in redundant models.

3.2.9 Discussion

The prior model and GP hyper-parameters play an important role in determining model accuracy. For the method presented in this section, a good prior model should result in model errors that can be well approximated by a GP. For the squared exponential kernel, this means the error should be smooth, which is why we chose a first-order prior model as opposed to an instantaneous prior as in Ostafew et al. [2014a]. The length-scale hyper-parameters then dictate how smooth the function is expected to be. Since we assumed that training data would not be available from all modes, we chose hyper-parameters based on a configuration with no mass on the Husky. Since the Husky rotates at almost twice the rate in an offset configuration, these length scales were no longer accurate. This results in higher error while the vehicle is in the offset configuration. This could be addressed by separately optimising the hyper-parameters for each model when enough data is gathered; however, that was beyond the scope of this work.

In this section, we assumed that we were given a stream of data from the robot operating in the relevant conditions and our goal was to construct a model for the underlying robot dynamics. In future work, one might consider combining this model learning framework with a model-based reinforcement learning approach. In model-based rein-

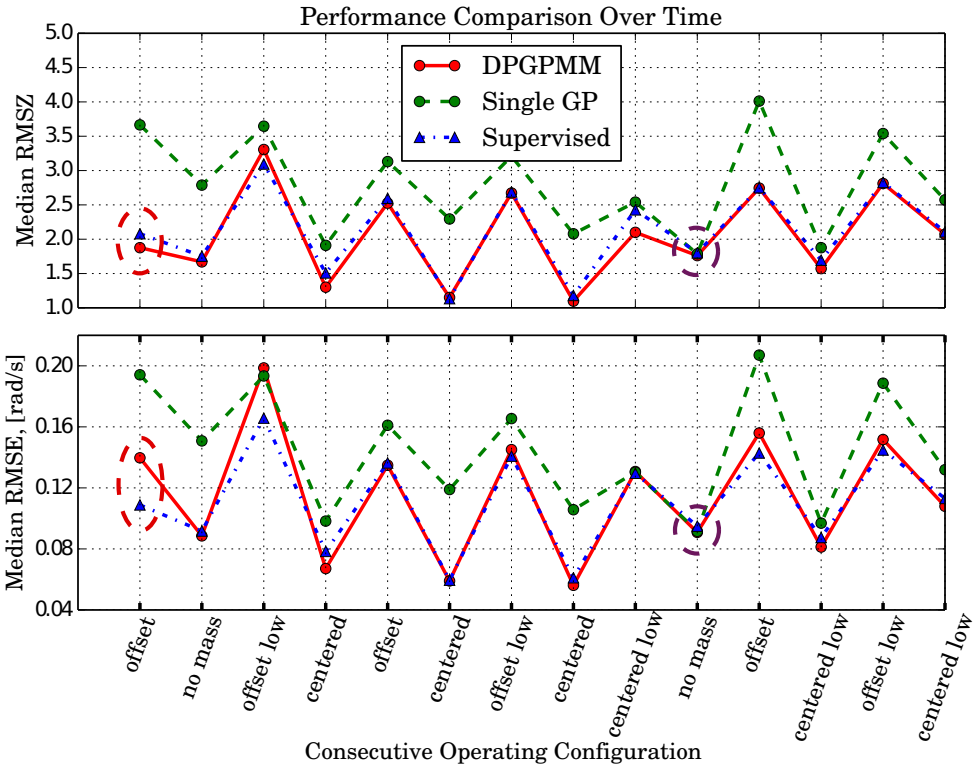


Figure 3.5: This plot shows the median of the RMS error and weighted RMS error for an entire run of consecutively changing configurations. For comparison, we show the proposed method (red), a single GP that is initialized with data from the no mass configuration (green), and an ideal mixture model with static GPs trained on labelled training data for each mode (blue).

forcement learning, a task is defined by a cost function and the goal is to complete the task while incurring the minimum cost. During the execution of the task, an algorithm controlling the robot can either exploit dynamics that are currently well known in order to achieve low cost in the short-term, or explore dynamics that are sub-optimal in the short term but reduce uncertainty in the dynamics model and potentially enable better long-term performance. For an interesting model-based reinforcement learning approach, we refer the reader to Xie et al. [2016]. This and other approaches may be interesting, in particular, to modify the behaviour of a robot when a new mode is detected because of the potentially large changes in model uncertainty and the impact that might have on task performance. Reinforcement learning is a large and active field which is beyond the scope of this thesis. However, for a good overview, we refer the interested reader to Kober et al. [2013b] and Berkenkamp [2019].

3.2.10 Summary

This section has presented a method using Gaussian Processes as experts in a Dirichlet Process mixture model to learn an increasing number of non-linear models for robot dynamics that are affected by different, discrete operating conditions. We have demonstrated in experiments how this approach stores and re-uses past experience from a robot's deployment in an arbitrary number of previous operating conditions, and automatically learns a new model when a new and distinct operating condition is encountered. The proposed method demonstrates significant improvements over single GP models approaching the performance of a model trained using known data association.

3.3 Experience Recommendation for Robot Modelling in Changing Conditions

3.3.1 Introduction

In the previous section, we presented an approach for learning multi-modal dynamics by combining GPs and the Dirichlet Process [McKinnon and Schoellig, 2017]. This allowed the robot to learn a new GP model for novel operating conditions and leverage an existing GP when the robot re-visited an operating condition. The GPs used in McKinnon and Schoellig [2017] represented the dynamics over the entire domain for each mode and therefore required a large number of training points to be effective. As a result, they were too computationally expensive to be used directly in the controller, which is limited

to GPs with only a small number of training points. For example, the GPs in McKinnon and Schoellig [2017] used 1000 points whereas the GPs in Ostafew et al. [2016], which ran in closed loop, used 50 points.

This section builds on work in Ostafew et al. [2016], which proposes a robust, learning-based Model Predicitve Controller (MPC) for repetitive path following using a vision-based localization algorithm [Paton et al., 2017]. The controller developed in Ostafew et al. [2016] uses local GP regression to construct a model for the dynamics at each timestep. These models are only valid around the vehicle's current position along the path. They are based on a fixed number of training points, or experiences, so have fixed computational cost. Our contribution is to generalize this approach for large and small repeated changes in the dynamics that depend on the operating conditions or physical configuration of the robot (see Fig. 3.6). Examples of factors that may change the dynamics include weather, terrain [Angelova, 2008], or payload configuration [McKinnon and Schoellig, 2017].

The specific limitation with Ostafew et al. [2016] that we wish to address is related to how past experiences are managed. In Ostafew et al. [2016], experiences are stored in first-in-first-out bins that can store a fixed number of experiences. Each bin corresponds to a particular location along the path and range of speeds. While this works well for accumulating experiences across the domain of the GP, it will not account for changes in the output of the model for the same input, which is a key property of multi-modal dynamics. If the dynamics change, the experience in each bin must be gradually over-written with experiences from the new mode. During this time, the bin will contain experience from multiple modes. Using these experiences to fit a GP violates the assumption of a GP that the underlying function is the sum of a single deterministic function and additive Gaussian noise. In addition, if the dynamics change back to the original mode, the same process must be repeated since the previous experience in this mode will have been forgotten.

The proposed method builds local GPs of the robot dynamics on each previous run to select experiences from the run with the most similar dynamics. These experiences are used to construct the GP used for control. This allows the GP used for control to leverage knowledge related to changes in dynamics that are both sudden and gradual as long as a similar change has been observed in the past. We use the same GP hyper-parameters as the GP in the controller. Accordingly, we introduce no additional model parameters and are able to assess how likely it is that the GP constructed from previous experiences will satisfy the assumptions of the safe controller, namely that the 3σ bounds on uncertainty are an accurate upper bound on model error. The local GPs used by our

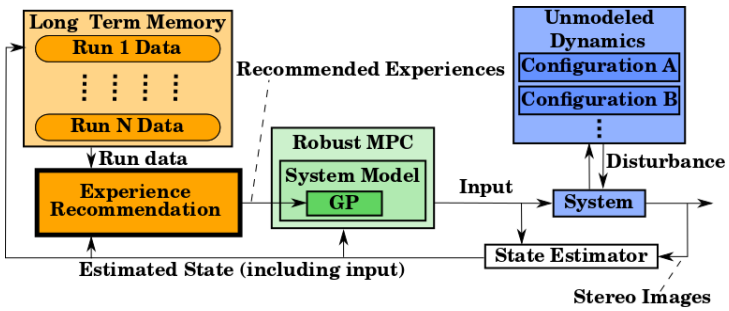


Figure 3.6: Block diagram showing the proposed experience recommendation in closed-loop with a safe controller. The system dynamics can change from one run to another. The proposed experience recommendation chooses the experiences from previous runs that best match the current dynamics. These experiences are used in a Gaussian Process (GP) to model the system dynamics. The robust Model Predictive Controller (MPC) is the controller from Ostafew et al. [2016].

method are inexpensive to compute enabling the robot to learn over and leverage data from a large number of runs without having to 'forget' previous experiences. This is a significant advantage over previous methods.

3.3.2 Problem Statement

The goal of this work is to learn a model for the dynamics of a ground robot performing a *repetitive, path-following task*. The robot may be subjected to large changes in its dynamics due to factors such as payload, terrain, weather, or tire pressure changes. We assume that these factors cannot be measured directly. The algorithm should scale to long-term operation and take advantage of repeated runs in the same operating conditions. The model should also include a reasonable estimate of model uncertainty that acts as an upper bound on model error at all times.

Further assumptions can be summarized as follows:

- The dynamics in each mode can be modelled as a GP with fixed hyper-parameters.
- Data is available to identify the GP hyper-parameters ahead of time.
- The number of operating conditions and the dynamics in each operating condition are not known ahead of time.
- The operating conditions are constant over a short time horizon.

A short time horizon could be similar to the horizon considered for MPC.

The system can be modelled by some nominal dynamics $\mathbf{g}_0(\mathbf{s}_k, \mathbf{u}_k)$ with additive, initially unknown dynamics $\mathbf{g}^c(\mathbf{x}_k)$ that are specific to discrete or continuous operating

conditions c and depend on features $\mathbf{x}_k$, so

$$\mathbf{s}_{k+1} = \mathbf{g}_0(\mathbf{s}_k, \mathbf{u}_k) + \mathbf{g}^c(\mathbf{x}_k). \tag{3.16}$$

The unknown dynamics are assumed to be a deterministic function with additive, zero-mean, Gaussian noise,

$$\mathbf{g}^c(\mathbf{x}_k) = \boldsymbol{\mu}^c(\mathbf{x}_k) + \boldsymbol{\eta}_k^c, \tag{3.17}$$

where $\boldsymbol{\mu}^c(\cdot)$ is a deterministic mean function, $\boldsymbol{\eta}_k^c \sim \mathcal{N}(0, \boldsymbol{\Sigma}_\eta^c)$, and $\boldsymbol{\Sigma}_\eta^c$ is a fixed noise covariance matrix. As before, we make the further assumption that each element of the unknown dynamics is independent so that each element can be modelled as a separate GP.

In contrast to the previous section, we do not separate states with known dynamics from states with unknown dynamics. All states are assumed to have some unknown dynamics. This formulation was chosen in this section because we build directly on work by Ostafew et al. [2016] which also used this formulation. Note that this is the special case of the previous formulation, assuming that the whole dynamics model is represented by (3.6).

3.3.3 Methodogy

In this section, we present our approach for long-term, safe learning control. Our approach makes extensive use of local GPs to model the robot dynamics.

Gaussian Process (GP) Disturbance Model

We model each element of the unknown dynamics, $g(\cdot)$, as a GP based on past observations (see Sec. 3.1.1). We drop the $(\cdot)^c$ for notational convenience because we learn a GP for each operating condition separately. As training data is added to a particular GP, uncertainty is reduced and the posterior distribution of the GP specializes to a particular family of functions which represents the system dynamics in a particular operating condition. The job of the experience recommendation, which is the contribution of this section, is to choose training data that results in the GP specializing to the functions that represent the robot dynamics in the current operating condition.

Controller and System Model

For this work, we use the robust model predictive controller from Ostafew et al. [2016] and refer the reader to this paper for details.

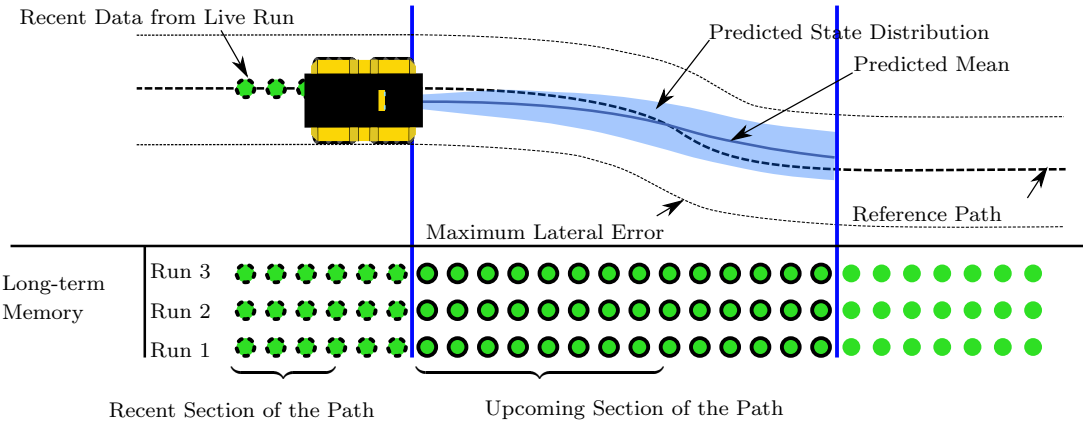


Figure 3.7: The predicted trajectory (shaded blue) is shown superimposed over the reference path in parallel with the storage structure for data from previous runs (green circles) that is indexed by run and location along the path. Data along the recent section of the path (circles with dotted outlines) is used to estimate the similarity between the current run and each previous run. This similarity is used to weight data from the upcoming section of the path (circles with solid outlines) and construct the predictive model used in MPC. We also use recent data from the current run to recursively update the model and adapt quickly to novel operating conditions and non-repetitive changes. The size of the regions of the path considered *upcoming* and *recent* may be considered hyperparameters that are linked to the MPC problem.

Data Management

The purpose of our method is to construct the best possible model of the system dynamics for MPC. MPC uses the dynamics over the upcoming section of the path to compute the control. We use data from the recently traversed section of the path to determine which past runs are relevant and can be used to construct the MPC prediction model. Referring to Fig. 3.7, for each previous run, i, we use data, $\mathcal{D}_i^- = \{g_{i,j}, \mathbf{x}_{i,j}\}_{j=1}^{m_i}$, from the recently traversed section of the path to construct a local GP, $g_i^-(\cdot)$. Each of these GPs is then used to generate predictions for the mean and standard deviation at each of the m_n recent experiences from the current run n, $\left\{\mu_i^-(\mathbf{x}_{n,j}), \sigma_i^-(\mathbf{x}_{n,j})\right\}_{j=1}^{m_n}$. These estimates are used to identify the past run with the most similar dynamics to the current run and reject runs where the dynamics were substantially different.

To update the GP used in the controller (see Fig. 3.6) with new points to model the dynamics on the upcoming section of the path, we randomly draw ten experiences (if available) from the most similar run in a window ahead of the vehicle overlapping with the MPC prediction horizon. These experiences are added to the set of fifty experiences already in the control GP. Fifty experiences (if available) are then randomly chosen from this combined set to get a new set of experiences for the control GP [McKinnon and Schoellig, 2017]. If no runs are recommended, we randomly remove ten data points from the set in the GP used for control. If this happens several times in a row, it will quickly

revert to the GP prior which acts as a 'safe mode' since the prior uncertainty bounds are the most conservative. We only use experiences from the most similar run, however, we could easily make use of experiences from multiple runs if not enough experience was available from one run.

The GP uses 50 points to allow the controller to run at 10 Hz with a 1.5 second prediction horizon. Updating the GP incrementally helps the model change smoothly which in turn results in smoother control inputs. We found this to be helpful during our experiments. The number of new points considered for updating the GP at each time (10 in our experiments) controls how quickly the set of points in the GP changes. Considering fewer points will result in slower/smoother changes to the GP while considering more points will result in faster changes. For example, the combined set of experiences before sampling to form a new GP in our case consists of 10/60 new points and 50/60 points from the existing GP. Since we sample 50 points uniformly to form the new GP, we expect the proportion of new and existing points to be preserved after sampling (5/6 existing and 1/6 new). By changing the proportion of new and existing points before sampling (by either varying the number of points in the GP or the number of new points considered), it is possible to change how quickly points in the GP are updated. We found a ratio of 1/6 new points to be a good compromise between fast adaptation and smooth change in our experiments given the 2 Hz rate at which the GP is updated (updating the GP at a higher rate will also increase the rate at which points in the GP change).

Run Rejection Criterion

Our first step is to eliminate runs where the dynamics are so different from the current dynamics that using data from these runs is likely to result in model errors that violate assumptions made by the safe controller. In particular, we must ensure that the 3σ bounds on the prediction from the GP are a reasonable upper bound on the model error [Ostafew et al., 2016].

To do this for candidate run i, we test whether the number of samples from $\mathcal{D}_n^-$ that lie further than three standard deviations from predictions made using the GP from run i is significantly higher than would be expected by chance. We do this using the binomial test.

Let m_n be the number of input-output pairs in $\mathcal{D}_n^-$ and N_{out} be the number of outliers. Outliers are defined as points outside of the predicted 3σ bounds, according to the predictions from $g_i^-(\cdot)$. The binomial distribution $B(n, m_n, p)$ describes the probability of drawing exactly n outliers from m_n independent samples where the probability

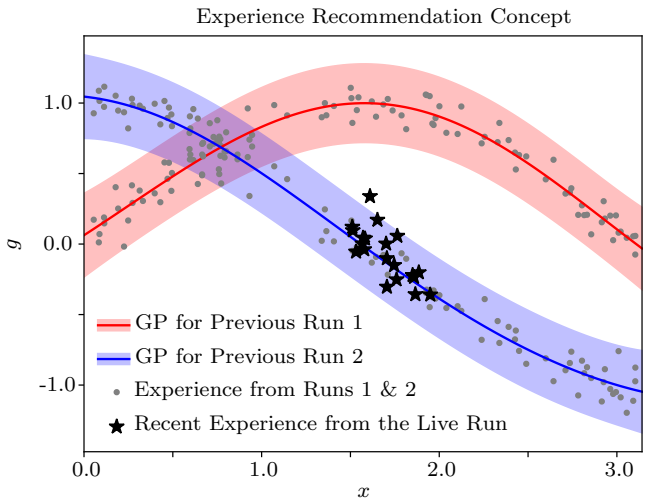


Figure 3.8: This figure shows a conceptual outline of the experience recommendation method. Experiences from past runs $\mathcal{D}_i^-$ is shown as grey dots. Recent experiences from the live run $\mathcal{D}_n^-$ are shown as black stars. Local GPs (red and blue) are fit to experience from previous runs. The shaded region represents the same confidence interval of these GPs that is used by the controller to compute margin on constraints, e.g., 3 standard deviations. First, we check if it is likely that the number of outliers observed (one here) could have occurred by chance. In this case, run 1 would be eliminated because all recent experiences are outliers (i.e. they all lie outside of the chosen confidence interval). Second, the recommended run is chosen as the run with the local GP that best explains recent experience from the live run. In this case, run 2.

of drawing an outlier is p. We calculate the probability of N_{out} or more outliers using

$$p(N_{\text{out}} \text{ or more}) = \sum_{n=N_{\text{out}}}^{m_n} B(n, m_n, p). \tag{3.18}$$

We reject the run if $p(N_{\text{out}} \text{ or more}) < \alpha$ where α is the significance level, or the probability of falsely rejecting a run. We chose a 5% significance level for our experiments. This may be reduced to avoid falsely rejecting runs, or increased to be more conservative. Since $3\sigma_{i,j}^{-}$ is such a conservative bound, changing α only changes the allowed number of outliers by one or two for $m_n = 100$ so the algorithm is not very sensitive to this parameter.

Any runs that make it past this step are considered as candidates for drawing experiences to model the dynamics over the upcoming section of the path.

Note that this will only be effective where the GP has data in the same input region as the run being tested. Otherwise the variance of the GP for the live run will be close to the GP prior which does not have strong discriminating power. In other words, this test only rules out runs where we have strong evidence to support that the two runs are different.

Run Similarity Measure

Our next step is to identify which runs are most similar to the current run given recent data from the live run, $\mathcal{D}_n^-$ and corresponding predictions $\{\mu_i^-(\mathbf{x}_{n,j}), \sigma_i^-(\mathbf{x}_{n,j})\}_{j=1}^{m_n}$ from the local GP for each candidate run, i.

We assume that the vehicle can be in a different operating condition for each run, and that one set of GP hyper-parameters is sufficient to describe the robot dynamics in each operating condition when operating conditions are considered separately. We can then compute the posterior probability that the current dynamics are from the same operating condition, c, as run i using:

$$p(c = i|\mathcal{D}_n^-) \propto \prod_{j=1}^{m_n} p(g_{n,j}|\mu_i^-(\mathbf{x}_{n,j}), \sigma_i^-(\mathbf{x}_{n,j}))p(c = i). \tag{3.19}$$

The first term on the right is the likelihood recent experiences if the operating conditions are the same as run i assuming each experience is independent. The second term on the right is the prior, which we assume to be equal for all runs. However, it could be informed by other sources such as computer vision, a weather report, or user input. Similar to our previous work [McKinnon and Schoellig, 2017], we reject any run that has lower

probability than the GP prior of generating $\mathcal{D}_n^-$. This is to ensure that experience added to the GP for control is likely to improve the performance beyond what could be achieved with no experience at all.

In our implementation, we use the log-probability to avoid numerical issues and do not normalize the posterior since we are only interested in finding the most likely run and not the actual probability distribution. We denote the un-normalized log-probability for run i with L_i. The run with the largest L_i is chosen as the recommended run.

Overview of the Algorithm

Putting the components above together, we arrive at the experience recommendation algorithm, Alg. 1.

Algorithm 1 The experience recommendation algorithm.

$n \leftarrow$ live run number
$\alpha \leftarrow$ significance level for binomial test
$scored\_runs = \{\}$
for $i = 1...n-1$ **do**
  $g_i^-(\cdot) \leftarrow$ Fit a GP to $\mathcal{D}_i^-$
  $N_{\text{out},i} \leftarrow$ number of outliers in $\mathcal{D}_n^-$ according to $g_i^-(\cdot)$
  $p_{b,i} \leftarrow$ Binomial test probability given $N_{\text{out},i}$ (3.18)
  if $p_{b,i} < \alpha$ **then**
    continue
  end if
  Compute L_i as the log of (3.19)
  Append (i, L_i) to $scored\_runs$
end for
$\mathcal{D}^{ctrl} \leftarrow$ experiences in the GP used for control
if $scored\_runs$ is empty **then**
  Remove 10 experiences (if available) from $\mathcal{D}^{ctrl}$
  $\mathcal{D}^{rec} + \leftarrow \{\}$
else
  $i^* \leftarrow$ run with largest L_i
  $\mathcal{D}^{rec} \leftarrow$ 10 randomly selected experiences from the upcoming section of the path in run i^*
end if
Construct a new GP for control using up to 50 points (if available) from $\left\{\mathcal{D}^{ctrl}, \mathcal{D}^{rec}\right\}$

3.3.4 Experiments

Experiments were conducted on a 900 kg Clearpath Grizzly skid-steer ground robot shown in Fig. 3.9. We tested our algorithm with the Grizzly in three configurations. First, the *nominal* configuration, with no changes to the vehicle. Second, the *loaded* configuration, with six bags of gravel, weighing approximately 30 kg each, in a cargo carrier mounted on the Grizzly (see Fig. 3.9). Finally, the *altered* configuration, where the rotational rate commands were multiplied by 0.7. Compared to the *nominal* configuration, the *loaded* configuration results in over-steer and the *altered* configuration results in under-steer.

Our first experiment was conducted in a parking lot on a 42 m long course. During this experiment, the configuration was switched between the *nominal* and *altered* configurations. We compare our proposed method to a baseline method, which uses only experiences from the most recent run. We conducted three runs in each configuration to allow the baseline method to converge to the dynamics in each configuration before switching. This serves as a simple case to demonstrate a few features of our algorithm.

Our second experiment was also conducted in a parking lot on a similar course. However, we switched between all three configurations after only two runs in each over a total of 30 runs to compare our method to the baseline method during long term operations. This was to demonstrate that the proposed method continues to learn during long term operation and maintains high performance in all three configurations regardless of the order of configuration switches.

Implementation

Our algorithm was implemented in C++ and can process up to 300 runs in a single thread at 2 Hz. This number is extrapolated based on the fact that the computational cost of the proposed method scales linearly with the number of runs. We consider the last three seconds of data (30 samples) from the live run for $\mathcal{D}_n^-$. The experience recommendation process runs in a separate thread to the controller. Therefore, it does not add any computation time to the control loop and can run at a lower rate to process more runs.

Figure 3.9: Clearpath Grizzly in the *loaded* configuration with six bags of gravel in the cargo carrier. Each bag weighs approximately 30 kg for a total payload of 180 kg.

System Model and Controller Parameters

Our process model is the unicycle with an additive GP learning term [Ostafew et al., 2016],

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + dt \begin{bmatrix} \cos\theta_k & 0 \\ \sin\theta_k & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_k^{cmd} \\ \omega_k^{cmd} \end{bmatrix}}_{\text{unicycle}} + dt \underbrace{\begin{bmatrix} g_x(\mathbf{x}_k) \\ g_y(\mathbf{x}_k) \\ g_\theta(\mathbf{x}_k) \end{bmatrix}}_{\text{learning term}}, \tag{3.20}$$

where v_k^{cmd} and ω_k^{cmd} are the commanded speed and turn rate, x and y are the position in the reference frame, θ_k is the orientation, and dt is the timestep of the controller.

The cost function for MPC is a quadratic penalty on lateral error, heading error, ω_k^{cmd}, $(v_k^{cmd} - v_k^d)$ where v_k^d is the desired speed, $\dot{\omega}_k^{cmd}$, and $\dot{v}_k^{cmd}$. The respective weights are 500, 35, 5, 4, 1000 and 500. The desired speed was set at 1.5 m/s for all of our experiments.

Model Predictive Performance

In order to evaluate the quality of the models constructed using our method, we first compared the multi-step prediction performance of a GP constructed using our experience recommendation method to a GP constructed using experiences from the most recent run. We consider the rotational dynamics, because they differ the most between configurations.

To measure the accuracy of the prediction of the mean, we use the M-RMSE and M-RMSZ. The lower the M-RMSE, the better. An M-RMSZ around one is ideal, a larger M-RMSZ around two indicates that the model is over-confident and M-RMSZ less than

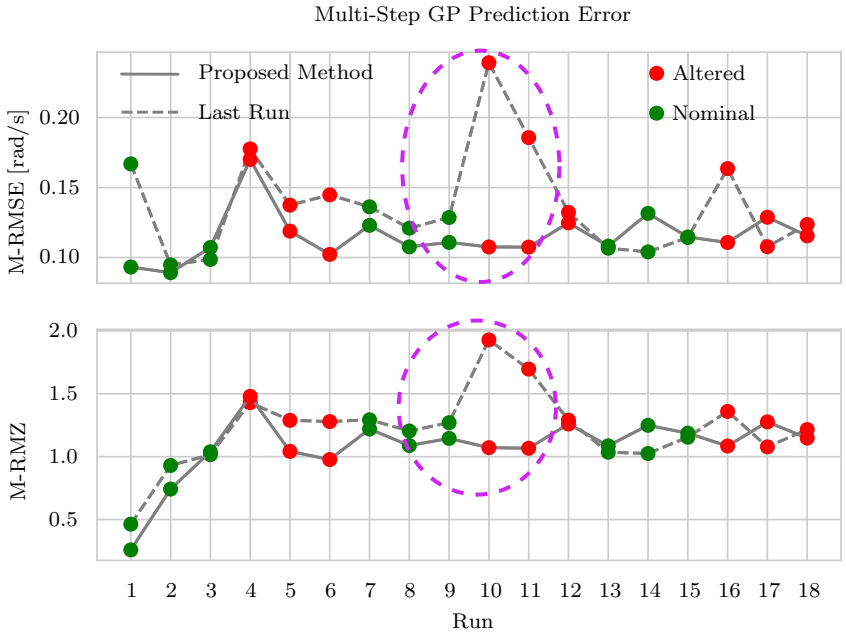


Figure 3.10: This figure shows how the proposed method improves the GP multi-step prediction performance compared to using experiences from the last run, especially during runs 8-11, indicated by the purple dotted circle. The configuration is switched between the ***nominal*** configuration (green) and the ***altered*** configuration (red). The lower M-RMSZ and higher M-RMSE on run the third transition during run 16 for the Last Run method indicates that the controller exploited actions where the model was more uncertain for this run.

one indicates that the model is conservative.

Figure 3.10 shows that the M-RMSE after a transition is up to 2.5 times higher when using experiences from the last run compared to the proposed method. The M-RMSZ shows a similar trend with the proposed method continually closer to one than the baseline method. We ignore run one for this comparison because the robot did not have any experience.

Closed-Loop Performance

To assess the impact of our method on closed-loop performance, we compare the speed, control cost, and tracking error using our method compared to when the GP is constructed using experiences from the last run.

Figure 3.11 shows that after transitions from the nominal configuration to the altered configuration, the proposed method significantly lowers the control cost compared to the baseline method. This cost comes from large lateral tracking errors due to under-steering. The vehicle under-steers because the model is based on the dynamics before the turn rate commands applied to the vehicle were multiplied by 0.7. Thus, the controller expects

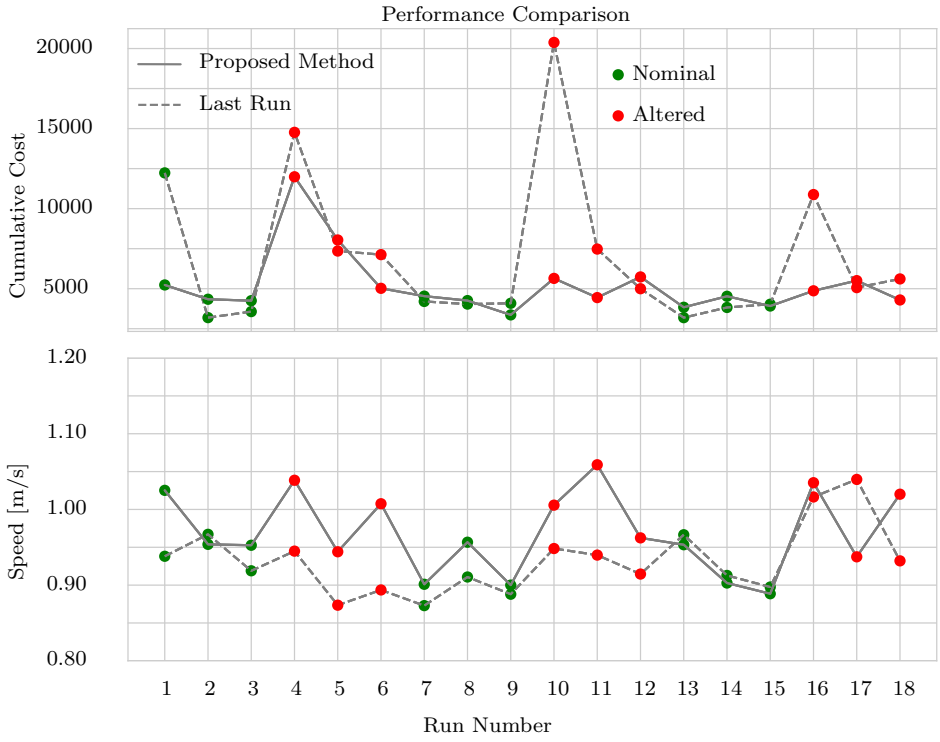


Figure 3.11: The cumulative step cost calculated using the MPC penalty function on the states and inputs the system actually visited during a run, and the average speed during each run. Runs in the ***nominal*** configuration are green and runs in the ***altered*** configuration are red. The proposed method reduces the control cost compared to using experiences from the last run. The average speed when using the proposed method is slightly higher because the controller does not have to slow down to make large corrections as often.

the vehicle to respond as if the turn rate commands were larger than what is being sent to the vehicle. We do not observe the same difference when transitioning to the nominal configuration because the vehicle will tend to over-steer in this case, and the penalty function in SMPC penalizes the magnitude of the turn rate commands, which naturally prevents the vehicle from over-steering. Choosing a different control cost may result in an increased cost for transitioning the configuration either way.

Experience Recommendation by Configuration

Figure 3.12 shows that the proposed method prefers experiences from the same configuration as the vehicle's current configuration and relies heavily on experiences several runs in the past. This demonstrates that it is beneficial to store not only the most recent experiences, but also many experiences from the past in order to leverage experiences from the same configuration.

Some of the time, experiences are chosen from runs with a different configuration.

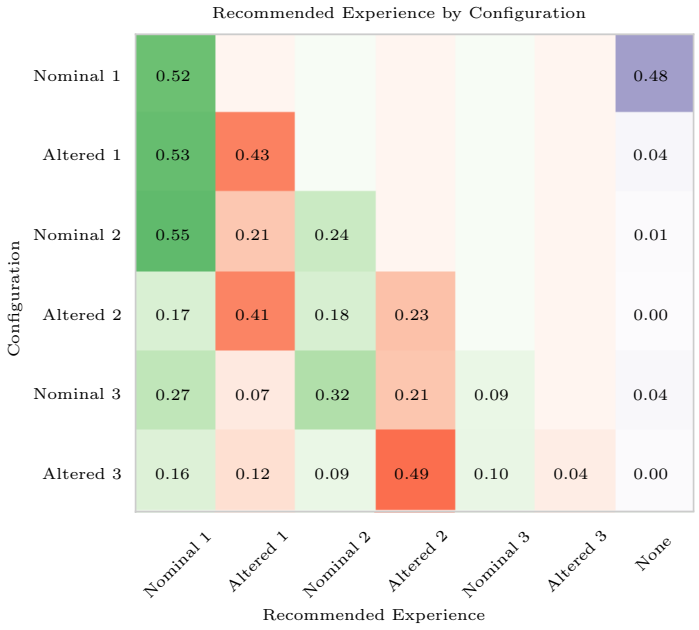


Figure 3.12: The distribution of recommended experiences by configuration when the vehicle was in each configuration. Each row corresponds to three runs conducted in the same configuration. Each column shows the proportion of experiences recommended from each previous set of runs. The color of the column indicates the vehicle configuration for those runs. Green is for the ***nominal*** configuration, red is for the ***altered*** configuration, and purple is for when ***none*** of the previous runs were recommended.

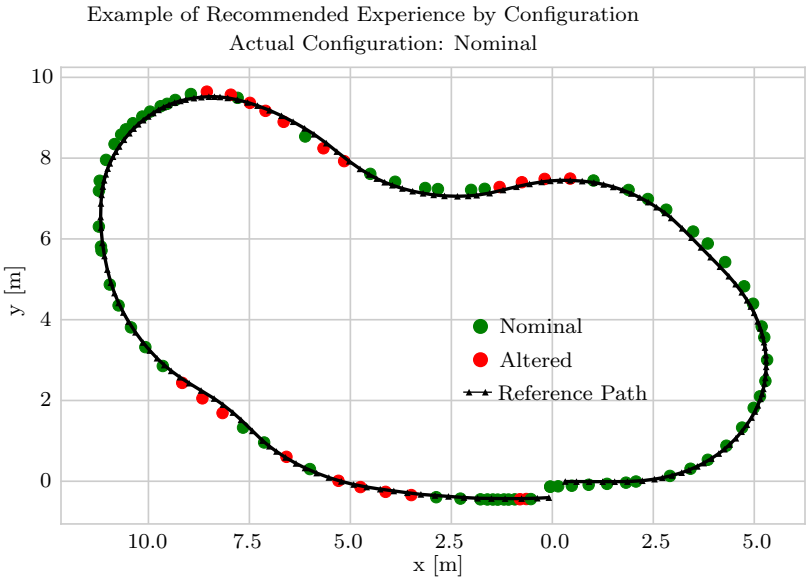


Figure 3.13: A top-down view of the path showing the experiences recommended by configuration as the vehicle moves along the path while the vehicle is in the ***nominal*** configuration. Red and green markers indicate when experiences were recommended from the ***altered*** and ***nominal*** configurations, respectively. The reference path is shown in black. This shows that experiences are primarily recommended from a different configuration along straight sections of the path which is when the dynamics in the two configurations are the most similar.

Figure 3.13 shows that this is primarily along straight sections of the path where the vehicle is not turning and therefore the dynamics are similar. On all sections of the path with sharp corners where the dynamics are the most different between configurations, the proposed method prefers experiences from the same configuration.

The proportion of time that the proposed algorithm rejects all previous runs decreases rapidly over time. The highest proportion is during the first three runs (Nominal 1 in Fig. 3.12) where during the first run, there are no previous runs to draw experiences from, and during the second and third run, the algorithm rejects all runs 17% and 11% of the time, respectively. After this initial phase of adaptation, the algorithm manages to find relevant experience over 89% of the time for each run.

Closed-Loop Performance, Longer Experiment

Our second experiment was to demonstrate the benefit of the proposed method during a longer operation with frequent changes in the operating conditions. The configuration was switched every two runs giving the baseline method one opportunity to adapt before changing the configuration again.

Figure 3.14 shows the cumulative control cost over thirty runs of a similar course to the first experiment. On average, the proposed method results in a 37% lower cumulative control cost compared to the baseline method primarily due to transitions to the *altered* configuration, which is the most distinct of the three.

In addition, the number of times the algorithm rejected all candidate runs decreased dramatically after the initial adaptation just like in the first experiment. For the first three pairs of runs, all runs were rejected 55%, 19% and 4% of the time, respectively. After this, the algorithm found matching experiences 97% of the time with the exception of the second pair of runs in the altered configuration, where it rejected all runs 9% of the time.

3.3.5 Discussion

Our method requires that the GP is a good model for the dynamics in each configuration to work well. We conducted experiments in more challenging off-road environments and the GP did not achieve lower M-RMSE than the prior even when learning only from runs in the same configuration. In these cases, our approach could not improve performance simply by changing the experiences in the model. It is important to note that during these runs, the vehicle remained within the lateral error constraints so the model error did not jeopardize the safety of the vehicle, it just did not improve the performance.

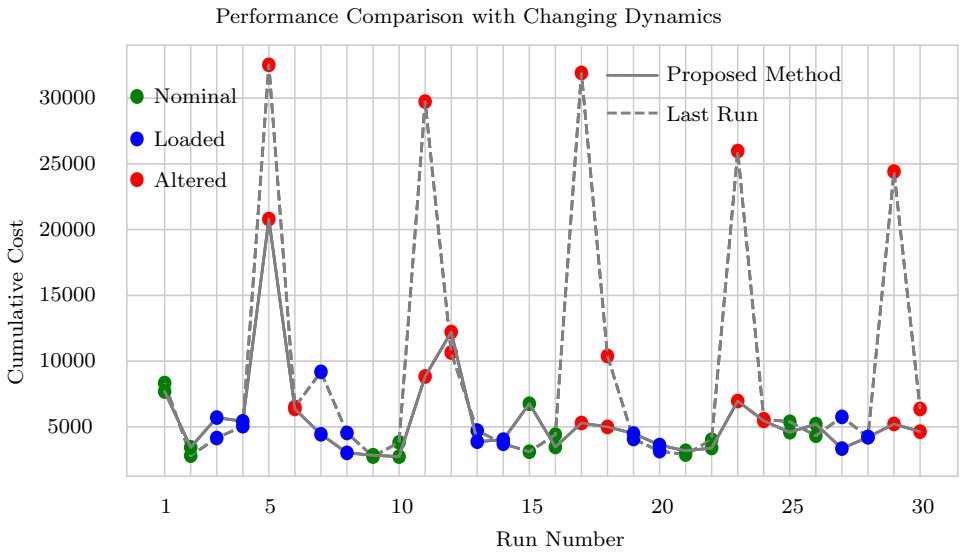


Figure 3.14: This figure compares the sum of the control cost for the actual states/inputs over 30 runs in varying configurations when using the proposed method compared to the baseline method, which is representative of Ostafew et al. [2016]. Colored markers indicate the configuration for each run. The vehicle cycled between the ***nominal*** configuration (green), the ***loaded*** configuration (blue), and the ***altered*** configuration (red).

Improving the underlying controller in this way is beyond the scope of this thesis, however our method is applicable to any model-based controller that relies on local probabilistic models of the dynamics and is performing a repetitive path-following task.

In addition, our method requires that the dynamics over the previous section of the path are a good indicator of the dynamics on the upcoming section of the path. The operating conditions are always sampled discretely (by run) but can be continuous (e.g. the factor we multiply turn rate commands). Although the method infers by run, which is discrete, it will still work for continuous variables that vary consistently between runs.

Our method requires large changes in the dynamics to produce a noticeable improvement. The dynamics in the *loaded* and *nominal* configurations were actually quite similar so using data from one or the other did not produce large enough changes to be noticeable over the course of a run. If a new configuration with half the load was added, our method would automatically determine that it was similar to the loaded or nominal configurations and leverage data from runs in those configurations.

3.4 Summary and Contributions

In this chapter, we presented two novel methods for modelling robot dynamics in changing conditions using GPs. A key contribution of these algorithms is that the number of

operating conditions or the number of distinct dynamics they induce do not have to be known ahead of time. The first method used a Dirichlet Process mixture of GP experts to construct a mixture model for robot dynamics where the number of components does not have to be specified ahead of time. Through experiments on a Clearpath Husky, we showed that the proposed method could automatically recover multiple models for robot dynamics when presented with a stream of data generated by a human driving the robot. Further, we showed that the predictive performance of these models was comparable to a model trained using the appropriate data for each mode. While successful at modelling robot dynamics, it was not computationally feasible to use this approach in closed loop with SMPC. This triggered the development of a second method that made the further assumption that the vehicle was performing a repetitive path-following task in order to construct local GPs that were more computationally efficient. This method was able to search past experiences for those relevant to the robots current operating conditions or revert to a safe mode if none were found. We demonstrated this method on the Clearpath Grizzly and showed that it could be integrated with a state-of-the-art SMPC algorithm imposing minimal additional computational cost on the control loop while searching up to 300 previous traversals of the path for relevant experience. The first method was was published in the International Conference on Robotics and Automation in 2017 [McKinnon and Schoellig, 2017] and the second in the proceedings of the International Conference on Intelligent Robots and Systems in 2018 [McKinnon and Schoellig, 2018].

In summary, the contributions of this early work are:

- A multi-modal model learning framework capable of learning an apriori un-specified number of non-linear models for robot dynamics and will automatically revert to a conservative 'safe mode' when the model has insufficient data to explain the current robot dynamics.

- An experience recommendation system to modify an existing learning-based SMPC framework to learn multi-modal dynamics that both recognised relevant past experience and when the current dynamics are different from previous runs so the model can smoothly transition to a conservative 'safe mode'.

Chapter 4

Bayesian Linear Regression-Based Methods

In Chapter 3, we presented a method to adapt to changes in robot dynamics using GPs with fixed hyperparameters. We observed that the accuracy of the models varied depending on particular robot dynamics, even if the GP models were trained with relevant data (Figure 3.5). The method that was computationally feasible in closed loop with SMPC relied primarily on data from previous runs to model the robot dynamics. This meant it took one full traversal of the path to learn new dynamics (Figure 3.14). In this chapter, we present a novel method, based on Bayesian Linear Regression (BLR), to overcome some of the drawbacks of fixed hyperparameters and use data from the current run to adapt continually to changing dynamics.

4.1 Introduction

In the previous chapter, we presented two methods using GPs to model the robot dynamics in changing conditions and were able to demonstrate the second method, presented in Sec. 3.3, in closed loop. The primary contribution of these methods was to develop a model that could represent nonlinear dynamics in a number of discrete modes without requiring the number of discrete modes to be known ahead of time.

In both of these methods, we used GPs with fixed hyper-parameters to model the robot dynamics which is a common approach when using GPs in practice [Hewing et al., 2018b, Ostafew et al., 2016]. This is primarily because hyperparameter optimisation is computationally expensive and requires a validation dataset to avoid overfitting. The requirement that a validation set be gathered in each new operating condition extends the period of time that the robot would have to operate with a conservative model (referred

to as the 'safe' mode in the previous chapter) before being able to leverage new data to improve the model for robot dynamics. In one method where the hyperparameters were recomputed during an experiment, the controller frequently had to revert to a conservative behaviour because model error bounds estimated by the GP were over-confident [Akametalu et al., 2014].

For a mobile ground robot, fixed hyperparameters correspond to the assumption that the GP will be able to predict the dynamics of the robot equally well whether the robot is stationary or moving at its top speed over rough terrain given sufficient training data. While it is possible to imagine that such a model might exist, it presents the designer with the daunting task of identifying all the variables that this model depends on and providing enough training data to properly fit the model. In addition, even if this model could be identified, the designer would still be faced with the challenge of making it computationally efficient enough to leverage the predictive power of SMPC. In the face of this challenge, there are two practical approaches.

First, the GP model could use an upper bound for the variance of the aleatoric uncertainty so that model error bounds derived from this uncertainty would be an upper bound in the entire domain of the model and thus maintain safety. The drawback of this approach would be that the model error bound would be conservative over large regions of its domain leading to conservative behaviour.

The second option, which is the focus of this chapter, is to use a model that adjusts the aleatoric uncertainty online without the computationally and data-intensive hyper-parameter optimisation required by GPs. This chapter explores the use of BLR to model the robot dynamics. Bayesian linear regression allows us to model functions that are a linear combination of basis functions with additive, Gaussian noise. Compared to GP regression, this requires additional prior knowledge to choose appropriate basis functions (also known as features). While this restricts the family of functions that can be represented using a BLR-based model, it also means that the basis functions can be chosen to limit the model's ability to overfit, thus reducing the importance of a large validation set which was required for reliable GP hyper-parameter optimisation. For mobile robots, where we often have good prior knowledge about how to choose these basis functions, BLR has strong potential to provide a reasonable estimate of robot dynamics while being able adjust the aleatoric uncertainty online in response to factors that cannot be represented by the choice of basis functions.

Local Linear Models: Linear regression tends to be used in one of two ways to model robot dynamics. First, a mixture of local linear models (i.e., linear in a set of

model coefficients) can be used to approximate a nonlinear function that cannot be well approximated globally by a linear combination of the chosen basis functions [Jamone et al., 2014, Ting et al., 2008]. In these methods, each local model is responsible for modelling the underyling function over a 'local' region of the function's domain. What is defined as 'local' is a parameter that must be identified from data, similar to the kernel length-scales in a GP. A GP can be linearized to get a local linear model that is valid about a specific operating point. However, local linear models have the advantage that the aleatoric uncertainty can be specified separately for each local region, e.g., Ting et al. [2008]. In our approach, we leverage the length of the MPC prediction horizon to define a 'local' region along the path and assume that a single set of model parameters is sufficient to describe the robot dynamics over the MPC prediction horizon. Continually re-constructing a single local model allows us to avoid the complexity of maintaining a globally consistent multi-modal model as in Jamone et al. [2014] while maintaining the ability to leverage data from previous traversals of the path.

Global or Time-varying Linear Models: Alternatively to fitting a global model composed of many local linear models, the dynamics of some robots can be represented by a single model that is linear in a set of coefficients. These parameters can be estimated efficiently using least squares [Xie et al., 2016]. This process can be repeated if the parameters change over time [Rosolia et al., 2017b]. Furthermore, if the basis functions are not known from a physical model, then trigonometric [Lázaro-Gredilla et al., 2010, Gijsberts and Metta, 2013] or polynomial [Murphy, 2012] basis functions can be used where more basis functions can be used to get a more accurate approximation (if sufficient data is available to avoid over-fitting). Like these approaches, we use one set of coefficients to model the robot dynamics at any time instant and update them continually using the live stream of data from the robot. What is unique is that we assume that aleatoric uncertainty will change over time and we incorporate data from past runs based on how similar the robot's dynamics were during those runs to the current robot dynamics.

4.2 Problem statement

The goal of the work presented in this chapter is to learn a model for the dynamics of a ground robot performing a repetitive, path-following task. The robot may be subjected to large changes in its dynamics due to factors such as payload, terrain, weather, or tire pressure changes. We assume that these factors cannot be measured directly. The algorithm should scale to long-term operation and take advantage of repeated runs in

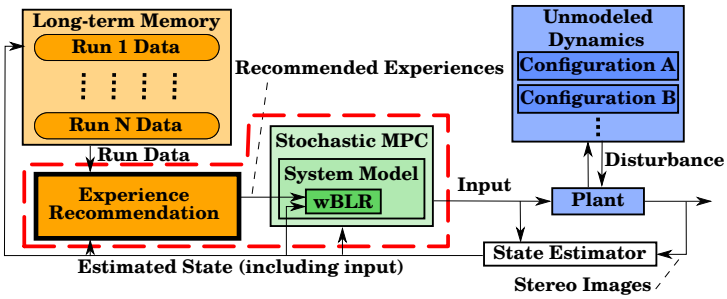


Figure 4.1: Block diagram showing the proposed model learning method in closed loop with a safe controller (red dashed box). The system dynamics can change from one run to another and over the course of a run. We use weighted Bayesian Linear Regression (wBLR) to learn the actuator dynamics of the plant. This approach, which enables fast adaptation and long-term learning, is shown to be highly effective in experiment. We encourage the reader to watch our video showing the experiments and datasets used in this section: http://tiny.cc/fast-slow-learn.

the same operating conditions. The model should also include a reasonable estimate of model uncertainty that acts as an upper bound on model error at all times.

Further assumptions can be summarized as follows:

- The unknown dynamics can be factored such they are close to a linear combination of known basis functions with additive Gaussian noise.
- The number of operating conditions and the dynamics in each operating condition are not known ahead of time.
- The operating condition is constant over a short time horizon.

A short time horizon could be similar to the horizon considered for MPC.

We consider systems with dynamics of the form:

$$\mathbf{s}_{k+1} = \mathbf{h}(\mathbf{s}_k, \boldsymbol{\xi}_k), \tag{4.1}$$

$$\boldsymbol{\xi}_{k+1} = \mathbf{g}^0(\boldsymbol{\xi}_k, \mathbf{u}_k) + dt\mathbf{g}_k(\mathbf{x}_k), \tag{4.2}$$

where the state of the system $\mathbf{s}$ evolves according to known dynamics $\mathbf{f}(\cdot)$ that depend on $\mathbf{s}$ and the state of the actuators $\boldsymbol{\xi}$. We assume that our control input $\mathbf{u}$ affects the actuator dynamics which consist of a known part $\mathbf{g}^0(\cdot)$ and an unknown and potentially changing part $\mathbf{g}_k(\cdot)$ that we wish to learn. We assume that the dynamics are well approximated by a linear combination of known basis functions, which make up the elements of the feature $\mathbf{x}_k$, and are corrupted by additive, zero-mean, Gaussian noise. In contrast to the previous chapter, we assume that the variance of this noise can change over time. The

subscript refers to the sampling time and dt is the sampling period. If we have no prior knowledge about the system then (4.1) can be omitted.

The system is constrained by known state and input constraints:

$$\mathbf{s}_k \in \mathcal{S}, \tag{4.3}$$

$$\boldsymbol{\xi}_k \in \boldsymbol{\Xi}, \tag{4.4}$$

$$\mathbf{u}_k \in \mathcal{U}. \tag{4.5}$$

4.3 Bayesian Linear Regression

Bayesian Linear Regression can be used to model functions that are a linear combination of known basis functions with additive, Gaussian noise. In this section, we present weighted BLR (wBLR) where each data point, or experience, can be assigned a weight indicating its importance. The weight can be used to vary its contribution to the current regression. These weights are determined in a separate step and are useful for tasks that involve learning from multiple sets of experience where some may be more relevant than others.

Suppose we are given a *weighted* dataset $\mathcal{D}^l = \{\mathbf{x}_i,\ g_i,\ l_i\}_{i=1}^n$ with scalar weights $l_i \in [0, 1]$ that determine the importance of each data point. If $l_i = 0$, the point has no influence on the regression, and if $l_i = 1$, the point is fully included. In a simple scenario, all weights can be set to 1, in which case we recover regular BLR. We assume that the dynamics of interest depend on a vector of model parameters $\mathbf{w}$ and are of the form:

$$g(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + \eta, \tag{4.6}$$

where $\eta \sim \mathcal{N}(0,\ \sigma^2)$. The goal of wBLR is to determine the distribution of both $\mathbf{w}$ and σ^2 given $\mathcal{D}^l$.

We assume that all observations in $\mathcal{D}^l$ are independent so that their likelihood can be factored as a product of the individual likelihoods. The expression we use for the likelihood of data in $\mathcal{D}^l$ is:

$$p(\mathbf{g} \,|\, \mathbf{X}, \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} \mathcal{N}(g_i \,|\, \mathbf{x}_i^T\mathbf{w},\ \sigma^2)^{l_i}, \tag{4.7}$$

where $\mathbf{g}$ is a column vector of g_i, and rows of $\mathbf{X}$ are $\mathbf{x}_i$.

The main difference between our approach and regular Bayesian Linear Regression our use of the weight l_i for each term in the likelihood. This is based on the intuition

that in the case of independent observations and the weighting below, the likelihood term for two identical observations weighted by half ($l_i = 0.5$) is equivalent to a single observation weighted fully ($l_i = 1$). Since the likelihood term is what contributes evidence to the posterior, we can use this to adjust how much each observation contributes to the posterior model parameters.

This intuition can be understood mathematically by looking at the canonical form of distributions in the exponential family. In this form, the posterior of the distribution's parameters is related to the weighted sum of the average sufficient statistics of the prior and posterior where the weights are the number of observations attributed to the prior and posterior respectively. The data weights l_i gives us the means to adjust the contribution of each point to the posterior individually. For more detail, we refer the interested reader to Appendix B.

Expanding (4.7), we get:

$$
\begin{aligned}
& p(\mathbf{g} \mid \mathbf{X}, \mathbf{w}, \sigma^2) \\
& = \prod_{i=1}^{n} \left[(2\pi)^{-\frac{1}{2}} (\sigma^2)^{-\frac{1}{2}} \exp\left(-\frac{1}{2\sigma^2} (g_i - \mathbf{x}_i^T \mathbf{w})^2 \right) \right]^{l_i} && (4.8) \\
& = (2\pi)^{-\frac{tr(\mathbf{L})}{2}} (\sigma^2)^{-\frac{tr(\mathbf{L})}{2}} \exp\left(-\frac{1}{2\sigma^2} ||\mathbf{g} - \mathbf{X}\mathbf{w}||^2_{\mathbf{L}} \right), && (4.9)
\end{aligned}
$$

where $\mathbf{L}$ is a diagonal matrix with diagonal elements l_i. Now we factor the likelihood to get it in terms of the parameters of the distribution rather than the data. Ignoring constants (which do not depend on $\mathbf{w}$ and σ):

$$
\begin{aligned}
& p(\mathbf{g} \mid \mathbf{X}, \mathbf{w}, \sigma^2) \\
& \propto (\sigma^2)^{\frac{-tr(\mathbf{L})}{2}} \exp\left(\frac{-1}{2\sigma^2} (||\mathbf{g}||^2_{\mathbf{L}} - 2\mathbf{g}^T \mathbf{L}\mathbf{X}\mathbf{w} + ||\mathbf{X}\mathbf{w}||^2_{\mathbf{L}}) \right) \\
& \propto (\sigma^2)^{\frac{-tr(\mathbf{L})}{2}} \exp\left(\frac{-1}{2\sigma^2} ||\mathbf{w} - \bar{\mathbf{w}}||^2_{\mathbf{\Sigma}^{-1}} \right) \\
& \qquad \exp\left(\frac{-1}{2\sigma^2} (||\mathbf{g}||^2_{\mathbf{L}} - ||\bar{\mathbf{w}}||^2_{\mathbf{\Sigma}^{-1}}) \right), && (4.10)
\end{aligned}
$$

where $\bar{\mathbf{w}}^T = \mathbf{g}^T \mathbf{L}\mathbf{X}(\mathbf{X}^T \mathbf{L}\mathbf{X})^{-1}$ and $\mathbf{\Sigma}^{-1} = (\mathbf{X}^T \mathbf{L}\mathbf{X})$. This is of the form of a Normal Inverse Gamma (NIG) distribution:

$$
\begin{aligned}
p(\mathbf{g} \mid \mathbf{X}, \mathbf{w}, \sigma^2) = & \mathcal{N}(\mathbf{w} \mid \bar{\mathbf{w}}, \sigma^2 \mathbf{\Sigma}) \\
& IG(\sigma^2 \mid tr(\mathbf{L})/2, (||\mathbf{g}||^2_{\mathbf{L}} - ||\bar{\mathbf{w}}||^2_{\mathbf{\Sigma}^{-1}})/2), && (4.11)
\end{aligned}
$$

where $IG(\cdot)$ is the Inverse Gamma distribution. We can then choose an NIG distribution as a conjugate prior [Murphy, 2012]. The prior represents our belief about the distribution of $\mathbf{w}$ and σ^2 before observing the data $\mathcal{D}^l$. The posterior joint distribution for $\mathbf{w}$ and σ^2 is then:

$$p(\mathbf{w}, \sigma^2 \,|\, \mathcal{D}^l) = NIG(\mathbf{w}, \sigma^2 \,|\, \mathbf{w}_N, \mathbf{V}_N, a_N, b_N) \tag{4.12}$$

$$\triangleq \mathcal{N}(\mathbf{w} \,|\, \mathbf{w}_N, \sigma^2 \mathbf{V}_N) IG(\sigma^2 \,|\, a_N, b_N), \tag{4.13}$$

where:

$$\mathbf{V}_N = (\mathbf{V}_0^{-1} + \mathbf{X}^T \mathbf{L} \mathbf{X})^{-1}, \tag{4.14}$$

$$\mathbf{w}_N = \mathbf{V}_N (\mathbf{V}_0^{-1} \mathbf{w}_0 + \mathbf{X}^T \mathbf{L} \mathbf{g}), \tag{4.15}$$

$$a_N = a_0 + tr(\mathbf{L})/2, \tag{4.16}$$

$$b_N = b_0 + \frac{1}{2}(\mathbf{w}_0^T \mathbf{V}_0^{-1} \mathbf{w}_0 + \mathbf{g}^T \mathbf{L} \mathbf{g} - \mathbf{w}_N^T \mathbf{V}_N^{-1} \mathbf{w}_N). \tag{4.17}$$

The subscripts 0 and N indicate parameters of the prior and posterior, respectively. The prior parameters (and their posterior equivalents) have the following interpretation: $\mathbf{w}_0$ is the prior mean for $\mathbf{w}$; a_0 proportional to the effective number of observations attributed to the prior; $b_0 = \sigma_0^2 a_0$, where σ_0^2 is the prior estimate of σ^2; and $\mathbf{V}_0 = 1/\sigma_0^2 \mathbf{\Sigma}_0^{\mathbf{ww}}$ where $\mathbf{\Sigma}_0^{\mathbf{ww}}$ is the covariance matrix of $\mathbf{w}_0$. For example, if we have prior knowledge suggesting that each element of $\mathbf{w}$ is independent and has a mean value of zero and variance τ_0^2, then $\mathbf{V}_0 = \tau_0^2/\sigma_0^2 \mathbf{I}$. These parameters can also be identified from data using (4.14)-(4.17) and a non-informative prior, i.e. $\mathbf{w}_0 = \mathbf{0}$, $\tau_0^2 = \infty$, $b_0 = 0$, and $a_0 = 0$, and σ_0^2 large.

Following Murphy [2012], the posterior marginals are then

$$p(\sigma^2 \,|\, \mathcal{D}^l) = IG(\sigma^2 \,|\, a_N,\, b_N), \tag{4.18}$$

$$p(\mathbf{w} \,|\, \mathcal{D}^l) = \mathcal{T}(\mathbf{w} \,|\, \mathbf{w}_N,\, \frac{b_N}{a_N} \mathbf{V}_N,\, 2a_N), \tag{4.19}$$

where $\mathcal{T}$ is a Student's t-distribution. The Student's t-distribution arises because of marginalizing out σ^2 and rapidly converges to a Gaussian distribution for $2a_N \gg 5$ [Murphy, 2012], which is useful for common tools in robotics for uncertainty propagation, which often assume Gaussian probability distributions.

Recursive Updates When dealing with streaming data such as the data generated by a robot driving, it is useful to continually update the model in order to adapt quickly to new scenarios. To do this while ensuring the model stays flexible enough to adapt to

sudden changes, we recursively update the prior parameters while keeping the strength of the prior fixed at a pre-determined value n_0. The value of n_0 determines how many effective data points we attribute to the prior. A large value for n_0 results in smoother estimates for the $\mathbf{w}$ and σ^2 while a smaller value for n_0 allows them to vary more quickly. If we start with fewer than n_0 points in the prior, e.g., $a_0 < n_0/2$, we update the prior using (4.14)-(4.17) with the weight for the new point set to one, and use this posterior as the new prior until a_0 reaches $n_0/2$. Once a_0 reaches $n_0/2$, we use (4.14)-(4.17) with the weight for the new point set to one and then use the following re-weighting to keep n_0 constant:

$$\mathbf{V}_{0^*} = \frac{n_0+1}{n_0}\mathbf{V}_N, \qquad \mathbf{w}_{0^*} = \mathbf{w}_N, \tag{4.20}$$

$$a_{0^*} = \frac{n_0}{n_0+1}a_N, \qquad b_{0^*} = \frac{n_0}{n_0+1}b_N. \tag{4.21}$$

The parameters $(\cdot)_{0^*}$ are the re-weighted parameters which become the new prior. This is equivalent to assigning the prior *and* the new point a weight of $n_0/(n_0+1)$ and carrying out a weighted update using (4.14)-(4.17).

Compared to the experience recommendation method for GPs in Sec. 3.3, changing n_0 has a similar effect to the number of points in the GP used for control relative to the number of new points that are considered each time the GP is updated, as described in Alg. 1. The main difference is that updating wBLR is computationally cheap and wBLR can be evaluated in constant time regardless of the number of data points used to fit it. Because of this, every data point generated by the robot can be used to continually update the model parameters as it drives in contrast to the GP-based method Sec. 3.3 where we chose to rely solely on data from past runs because of the limited number of points in the GP.

4.3.1 Computational Efficiency

There are three main operations that determine the computational efficiency of a model: *(i)* fitting the model to training data, *(ii)* evaluating the mean at a test point and *(iii)* evaluating the variance at a test point. Fitting a model can be done in parallel to the control loop to mitigate some of the computational cost of this step [McKinnon and Schoellig, 2018]. However, querying the mean and variance are part of computing the control which makes these operations time-critical. This is especially true for SMPC, which queries the model several times at each sampling time.

The time it takes to fit a GP to N data points, query the variance, and query the mean

scales with $O(N^3)$, $O(N^2)$, and $O(N)$, respectively. This is because of the computation of the matrix inverse $\mathbf{K}^{-1}$, the matrix product in (3.3), and the dot product in (3.2). For wBLR, the same operations scale with $O(N)$, $O(1)$, and $O(1)$ because of the dot products in (4.14)-(4.17) and because the output equations are parametric.

4.3.2 Simple Example

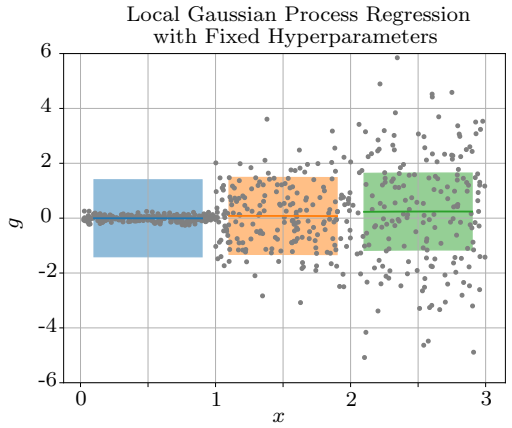
Safe control algorithms require an accurate estimate of the model uncertainty to guarantee constraint satisfaction even if the mean prediction from the model is inaccurate. One key difference between wBLR and GP regression (for fixed GP hyperparameters) is that the uncertainty estimate from a GP depends only on the similarity between the query point and training points in the *input* space as measured by a kernel (see (3.3), which only depends on $\mathbf{x}$). In contrast, the wBLR parameters are updated using (4.14)-(4.17) which depend on both $\mathbf{g}$ and $\mathbf{x}$. This enables the BLR-based models to adjust their aleatoric uncertainty estimate online.

Figure 4.2 shows a simple example comparing a GP and the proposed BLR-based method on data that is a zero-mean Gaussian noise with a standard deviation of 0.1 in $x \in (0, 1)$, 1.0 in $x \in (1, 2)$ and 2.0 in $x \in (2, 3)$. This change in variance represents a change in the robot dynamics that cannot be modelled exactly by the mean function of either method. The GP hyper-parameters and the BLR prior were set based on data from the entire domain. The key is that the standard deviation of aleatoric uncertainty (i.e. σ_η in (3.4)) is fixed after hyper-parameter optimisation so it is unable to adjust its uncertainty estimate in response to changes in the standard deviation of g in each local region. BLR, on the other hand, is able to adjust this, leading to an estimate of 0.35, 1.05, and 2.09 in each region respectively. The estimate in the first region is larger than 0.1 because it has been inflated by the prior.

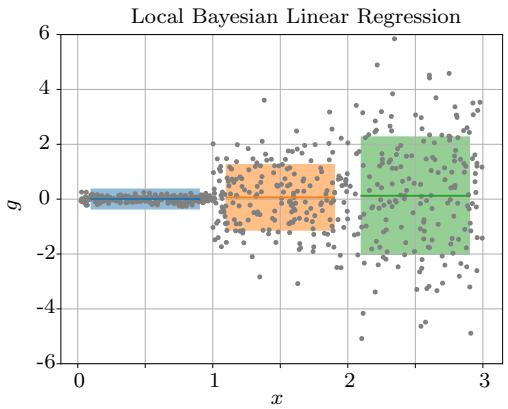
4.3.3 Application to Repetitive Path-Following

We consider a repetitive path following task with SMPC where data from each run is indexed by location along the path as depicted in Fig. 3.7. In this section, we describe how wBLR leverages this structure to efficiently model robot dynamics for repetitive path following.

Since the controller only makes use of the dynamics of the vehicle over the upcoming section of the path, our primary focus, as before, is to model the dynamics over the upcoming section of the path. The advantage for the GP-based method in the previous chapter was that we needed fewer points to model the dynamics over this restricted region



(a) This figure shows the posterior mean (solid line) and 1-sigma bounds (shaded region) for a GP with fixed hyper-parameters conditioned on data from each region in the domain.



(b) This figure shows the posterior mean (solid line) and 1-sigma bounds (shaded region) for a BLR model conditioned on data from each region in the domain.

Figure 4.2: This figure shows a simple comparing a GP and the proposed BLR-based method on data that is a zero-mean Gaussian noise with a standard deviation of 0.1 in $x \in (0, 1)$, 1.0 in $x \in (1, 2)$ and 2.0 in $x \in (2, 3)$. The uncertainty estimate of the GP with fixed hyper-parameters does not depend in the input x so is constant over the entire domain. For local Bayesian linear regression, however, the uncertainty estimate depends on both x and g and so it is able to adjust the uncertainty estimate appropriately.

of the state space. This made using GP models computationally feasible with SMPC. The advantage for wBLR is that it increases the chance that the relatively simple parametric model will be a good approximation for the dynamics.

Fast Adaptation

In order to adapt quickly to new scenarios, we use the most recent data pair $\{g_{k-1}, \mathbf{x}_{k-1}\}$ generated by the robot to update the model at every timestep. We use the *recursive update* explained in the previous section. These parameters are used as the prior at each timestep. In our previous work using GPs [McKinnon and Schoellig, 2018], the model reverted to a conservative prior when the current dynamics did not match the dynamics in any previous run. While this preserved safety, it took one full traverse of the path before the robot could adapt to a novel change in the dynamics. Leveraging data from the current run directly to update the dynamics model enables adaptation to new conditions as they arise, which will be demonstrated in Sec. 4.6.2.

Long-term Learning

To improve controller performance in the face of repetitive changes, we leverage data from previous runs in similar operating conditions. We consider data from all previous runs because the model update is efficient and the cost to evaluate the model does not depend on the number of points used to construct it. Our approach relies on the same steps as in the previous chapter. We construct local models using data from each previous run over the recently traversed section of the path, reject runs for which the local models predictions result in too many outliers, and estimate the probability that the robot dynamics in the current run are the same as the dynamics from each previous run. The difference is that rather than drawing experiences from the most similar run to fit a local model, we weight past data from each run according the probability that the dynamics are the same as the current run.

Suppose that $p(c = j|\mathcal{D}_n^-)$ is the probability that the robot dynamics in run j were the same as the current robot dynamics. Let $\mathcal{D}_j^+ = \{g_{j,i}, \mathbf{x}_{j,i}\}_{i=1}^{m_j}$ is data from the upcoming section of the path from run j. Then, all points in $\mathcal{D}_j^+$ are weighted using:

$$l_{j,i} = p(c = j|\mathcal{D}_n^-)/p(c = j^*|\mathcal{D}_n^-), \tag{4.22}$$

where j^* is the run with maximum posterior probability. This satisfies $l_{i,j} \in [0, 1]$ and normalising means that the effective number of points can increase with each additional run. With these weights, we use (4.14)-(4.17) to compute the posterior parameters of

the predictive model. This update (based on data from previous runs) is considered to be location specific and therefore discarded after computing the control. That is, the recursively updated prior becomes the prior for the next sampling time.

4.3.4 SMPC Controller Design

This section outlines the SMPC formulation used for experiments in this chapter including the path parametrization, cost function, ancillary control design, and uncertainty propagation. The formulation in this chapter differs from the one used in the previous chapter primarily in *(i)* how the cost function encourages progress along the path and *(ii)* our use of an ancillary controller to slow the growth of uncertainty in predicted states.

Model Predictive Contouring Control

In the previous implementation of SMPC, a target speed was specified for each waypoint along the path and the target heading was based on the heading of the closest waypoint. Separate controllers were used to handle manoeuvres like turns on the spot, directions switches, where the robot goes from forward to reverse, and the end of the path, where the robot should come to a stop close to the end-point of the path.

We use a Model Predictive Contouring approach, based on Lam et al. [2010], which expresses position error as lag error (parallel to the path) and contouring error (perpendicular to the path) and uses a virtual input to drive reference states along the path. Progress along the path can be traded off against other metrics such as tracking error by tuning the cost function. Since this is not our novel contribution we have included the details of our adaptation of path following in Appendix A.

Uncertainty Propagation

In contrast to the previous section, we linearise the dynamics model to propagate uncertainty instead of using the Sigma Point transform [Ostafew et al., 2016]. This is more computationally efficient because we have to compute the Jacobians anyway for SMPC. To improve the accuracy, alternatives to Euler's method such higher order Runge Kutta methods can be used [Van Loan, 1997, Kabzan et al., 2019].

Ancillary State Feedback Controller

An important difference between the SMPC framework in this chapter and the previous chapter is our use of an ancillary controller to bound the growth of predicted uncertainty.

The dynamics model on its own does not take into account the fact that the controller can take corrective actions at future times. Predicting the future states assuming that the control inputs are applied open-loop (as was done in Ostafew et al. [2016], which was used in the previous chapter) means that the predicted uncertainty can grow quickly and without bound. These conservative bounds result in conservative control inputs [Hewing and Zeilinger, 2017]. Our primary concern for the purpose of this thesis is that a conservative uncertainty propagation method could mask over-confident model uncertainty estimates making closed-loop results more difficult to interpret.

A computationally efficient approach to account for feedback when predicting uncertainty is to use make use of an ancillary controller as in Tube MPC [Aswani et al., 2013, Hewing and Zeilinger, 2017]. The ancillary controller is a state-feedback controller designed to reduce uncertainty in a particular state s. In this chapter, we use ancillary controllers of the form $\pi(s) = K_s(s - \bar{s})$ where K_s is a negative constant and $\bar{s}$ is the mean of state s. Section 4.4.2 shows how we apply this to a unicycle-type robot.

Constraint Tightening

Since our predictive model has uncertainty, we must tighten the constraints on the state and input to make sure the true system respects the true constraints (with high probability) when we use inputs that were calculated based on an approximate model. We must also ensure that the ancillary control policy remains feasible for our choice of control inputs (with high probability). Since enforcing these constraints jointly can lead to undesirable, conservative behaviour, we enforce them individually. Our treatment of the constraint tightening is not a novel contribution of this work and follows Hewing and Zeilinger [2017], to which we refer the reader for further details.

As before, we assume the state has a Gaussian pdf at all times. We also assume that the reference values for the state are perfectly known. Therefore, any error e which is a linear transformation of the state the reference also follows a Gaussian pdf. Let σ_e be the standard deviation of error e and ϵ_e be the small acceptable probability of error e violating a maximum value e^{max}. This probabilistic chance constraint can be converted into a deterministic one using:

$$p(e \leq e^{max}) \geq 1 - \epsilon_e \Leftrightarrow \bar{e} + \underbrace{r_e \sigma_e}_{\text{safety margin}} \leq e^{max}, \tag{4.23}$$

where r_e is the number of standard deviations of a Gaussian pdf containing $1 - \epsilon_e$ of the probability mass (e.g., 2.0 for $\epsilon_e = 0.05$).

Since we make use of an ancillary controller, we must also treat the input constraints as chance constraints. The ancillary controller acts on a state $s \sim \mathcal{N}(\bar{s}, \sigma_s^2)$ which is a random variable. As such, uncertainty in the state introduces uncertainty in the control inputs that might be applied. Let $\bar{u}$ be the input applied to the system by SMPC. The input used to predict the state is then $u = \bar{u} + \pi(s)$. Treating input constraints the same way as the state constraints, we get that:

$$p(u < u^{max}) \leq 1 - \epsilon_u \Leftrightarrow \bar{u} + \underbrace{r_u |K_s| \sigma_s}_{\substack{\text{input allocated for} \\ \text{the ancillary controller}}} \leq u^{max}, \tag{4.24}$$

where $|K_s|$ is the absolute value of K_s. Here, we can see that while the ancillary controller reduces the predicted uncertainty in s it will also reduce the control input available for controlling the nominal state.

The feedback gain K_s can be chosen as an infinite horizon LQR controller with the same cost function as MPC [Hewing and Zeilinger, 2017, Gao et al., 2014] or included in the optimisation problem [Vitus and Tomlin, 2011], but we found that a wide range of gains worked for our system so left the ancillary controller gain as a tuning parameter.

(a) The Grizzly loaded with gravel bags in the UTIAS Mars Dome. Data from this configuration was used to compare the predictive performance of GPs and wBLR in Sec. 4.6.

(b) The Grizzly on the muddy test track used for the closed-loop comparison between GPs and wBLR in Sec. 4.7.

Figure 4.3: The algorithms in this section were evaluated on the Clearpath Grizzly, a 900 kg ground robot. Estimates of the robots pose and velocity came from using Visual Teach and Repeat 2.0 [Paton et al., 2017].

4.4 Application to a Ground Robot

In this section, we show an example of how to apply wBLR in combination with SMPC to a ground robot, the Clearpath Grizzly depicted in Fig. 4.3. In parallel, we show how we apply GP regression to the same task for the purpose of comparison to the GP-based method presented in Sec. 3.3.

4.4.1 Robot Model

Let $\mathbf{s} = [x, y, \theta]^T$, the 2D position and heading of the robot, $\boldsymbol{\xi} = [v, \omega]^T$, the speed and turn rate of the robot, and $\mathbf{u} = [v^{cmd}, \omega^{cmd}]^T$, the commanded speed and turn rate of the robot. We assume that the dynamics of $\mathbf{s}$ are well approximated by a unicycle:

$$\underbrace{\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix}}_{\mathbf{s}_{k+1}} = \underbrace{\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + dt \begin{bmatrix} v_k \cos\theta_k \\ v_k \sin\theta_k \\ \omega_k \end{bmatrix}}_{\mathbf{h}(\cdot)}, \tag{4.25}$$

which is of the form (4.1). For wBLR, we will model the dynamics of $\boldsymbol{\xi}$ as:

$$\underbrace{\begin{bmatrix} v_{k+1} \\ \omega_{k+1} \end{bmatrix}}_{\boldsymbol{\xi}_{k+1}} = \underbrace{\begin{bmatrix} v_k \\ \omega_k \end{bmatrix}}_{\mathbf{g}^0(\cdot)} + dt \underbrace{\begin{bmatrix} [v_k^{cmd}, v_k]\mathbf{w}_k^v + \eta_k^v \\ [\omega_k^{cmd}, \omega_k]\mathbf{w}_k^\omega + \eta_k^\omega \end{bmatrix}}_{\mathbf{g}_k(\cdot)}, \tag{4.26}$$

which is of the form (4.2). For the model using GPs, we model the speed and turn rate dynamics as:

$$\underbrace{\begin{bmatrix} v_{k+1} \\ \omega_{k+1} \end{bmatrix}}_{\boldsymbol{\xi}_{k+1}} = \underbrace{\begin{bmatrix} v_k^{cmd} \\ \omega_k^{cmd} \end{bmatrix}}_{\mathbf{g}^0(\cdot)} + dt \underbrace{\begin{bmatrix} \mu_k^v(v_k, v_k^{cmd}) + \eta_k^v \\ \mu_k^\omega(\omega_k, \omega_k^{cmd}) + \eta_k^\omega \end{bmatrix}}_{\mathbf{g}_k(\cdot)}. \tag{4.27}$$

We use different models for the GP and wBLR so that the speed and turn rate for the GP-based model depends on the control inputs before data is added to the GP. If it did not, MPC would not be able to drive the vehicle for the first run. For wBLR, we keep $\mathbf{g}^0(\cdot) = \boldsymbol{\xi}_k$ which is a more natural choice for modelling dynamic systems. It should be noted that the wBLR prior is equivalent to the GP prior for a certain choice of $\mathbf{w}$. In both cases, the input to the learned model $\mathbf{x}$ is composed of elements of $\boldsymbol{\xi}$ and $\mathbf{u}$.

4.4.2 Ancillary Controller Design

We use a linear state feedback controller as an ancillary controller for the learned dynamics. The main benefit of this is that predicted uncertainty in the error the controller reacts to can be straightforwardly used to tighten the constraints on the control inputs using the method described in Hewing and Zeilinger [2017]. Here, we present an ancillary controller design for both wBLR and GP-based models and show simulation results demonstrating the effectiveness of this approach.

We use the following ancillary controllers to reduce the predicted speed and heading uncertainty and, as a consequence, position uncertainty, which is linked to the maximum path tracking error constraint. The turn rate and speed dynamics with the ancillary controllers are:

$$\begin{bmatrix} v_{k+1} \\ \omega_{k+1} \end{bmatrix} = \mathbf{g}_0(\cdot) + dt \begin{bmatrix} g_k^v(v_k, v_k^{cmd} + K_v(v_k - \bar{v}_k)) \\ g_k^\omega(\omega_k, \omega_k^{cmd} + K_\theta(\theta_k - \bar{\theta}_k)) \end{bmatrix}, \tag{4.28}$$

where $g_k^v(\cdot)$ and $g_k^\omega(\cdot)$ are the respective wBLR or GP-based models and $\mathbf{g}_0(\cdot)$ is the corresponding prior, either (4.26) or (4.27). The ancillary controller gains were chosen manually because it was easy to find values that worked well in practice.

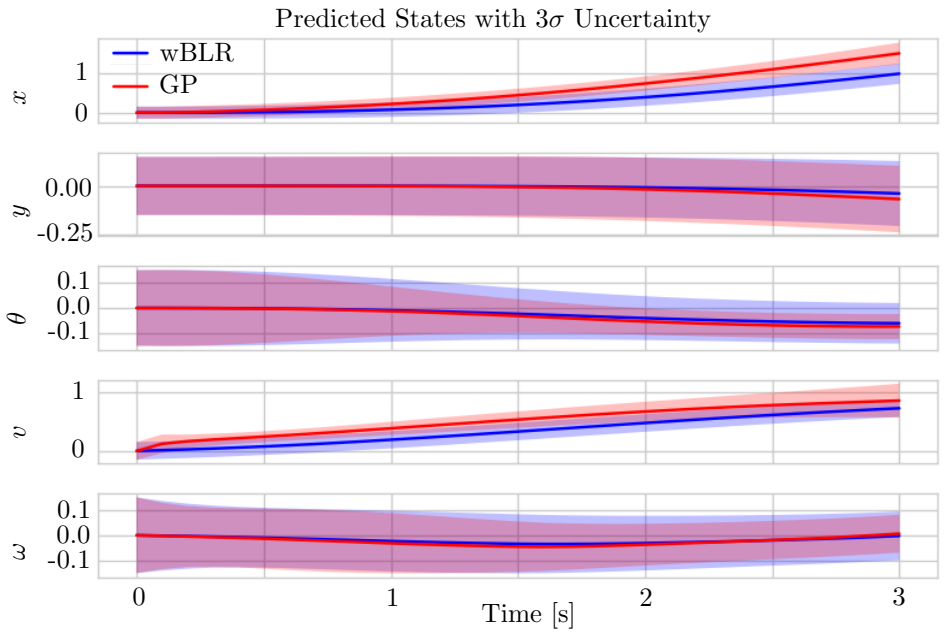


Figure 4.4: This figure shows the predicted state distribution from both the GP-based (red) and wBLR-based (blue) models using the proposed ancillary controller. The shaded regions are the predicted 3σ bounds. The proposed ancillary controller is able to keep the 3σ uncertainty in y (roughly equivalent to lateral uncertainty) at about 25 cm at the end of the prediction horizon for both models. This is well below the maximum lateral uncertainty used in our experiments.

Figure 4.4 shows the predicted uncertainty using the GP and wBLR models with the proposed ancillary controller. Each learning model was trained using data from the same experiment and K_v and K_θ were set to -5. Here, we can see that the proposed method is able to keep the uncertainty from growing substantially over the duration of a typical MPC prediction horizon for both GP- and wBLR-based models.

4.5 Offline Comparison to Gaussian Process Regression

In order to evaluate the suitability of the proposed method for predictive control, we compare the predictive performance of the BLR-based method proposed in this chapter to a context-aware GP with fixed hyperparameters (GP-Fixed-Rec) and a context-aware GP with hyperparmeters optimised using MLE and a sliding window of the last 100 datapoints (GP-MLE-Rec). The context-aware GP uses the experience recommendation method of the previous chapter to choose relevant experience from past runs.

This comparison was conducted on data collected in the UTIAS Mars Dome. The

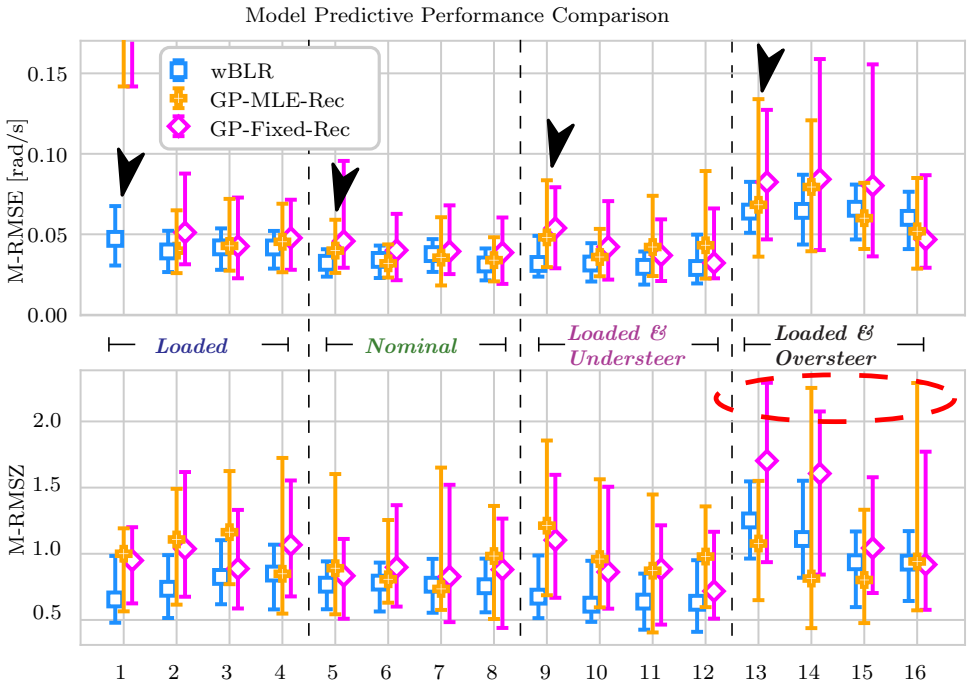


Figure 4.5: A comparison of M-RMSE and M-RMSZ for the rotational dynamics with the vehicle in four different configurations for two context-aware GP-based methods and the proposed method. The error bars indicate the 25th and 75th percentiles and the marker indicates the median. The 65 m path traversed sand, gravel, and concrete. Runs 1-4 are in the ***Loaded*** configuration, with 6 gravel bags in the rear of the Grizzly (see Fig. 3.9), runs 6-8 are with the vehicle in the ***Nominal*** configuration (no modification), runs 9-12 are with the vehicle in the ***Loaded & Understeer*** configuration, where it is loaded and the turn rate commands are multiplied by 0.7, runs 13-16 are in the ***Loaded & Oversteer*** configuration, where the vehicle is loaded and the turn rate commands are multiplied by 1.2. The black arrows indicate the first time the vehicle is driven in a new operating condition. The red circle indicates where the GP-based methods were over-confident, frequently producing M-RMSZ values above 2.0.

dynamics of the Grizzly were modified by adding heavy weights and artificially modifying the commands sent to the vehicle. We consider the rotational dynamics because they differ the most between configurations. We compare the model predictions given the inputs that were actually applied to the vehicle over the SMPC prediction horizon to the actual state of the vehicle recorded at the corresponding times. To measure the accuracy of the prediction of the mean and uncertainty, we use the M-RMSE and M-RMSZ respectively. An accurate model uncertainty estimate is important to ensure that the probability of violating the chance constraints formulated in Sec. 4.3.4 is kept at an acceptable level.

Figure 4.5 compares the proposed method to GP-Fixed-Rec and GP-MLE-Rec. The proposed method consistently achieves lower M-RMSE, especially during run 1 before the GP-based methods have data, and the first time the system encounters a new configura-

tion as indicated by the black arrows. This is the proposed method is able to incorporate relevant data from the current run using *fast adaptation*. While online hyperparameter optimisation generally improves the M-RMSE, it causes the GP overfit in the most challenging scenario, ***Loaded & Oversteer***, which can be inferred by the M-RMSZ value exceeding 2.0 during runs 14 and 16. In contrast, the proposed method is much more consistent and the M-RMSZ stays between 0.5 and 1.5, indicating the model has a reasonable estimate of model uncertainty.

4.6 Closed-Loop Experiments with Bayesian Linear Regression

In this section, we present experiments to demonstrate the performance of SMPC in closed loop using the wBLR-based method proposed in this chapter. These experiments are designed to *(i)* highlight the combination of fast adaptation and long-term learning, and *(ii)* demonstrate the controller on a 175 m outdoor course at speeds of up to 2.7 m/s.

Experiments were conducted on a 900 kg Clearpath Grizzly skid-steer ground robot shown in Fig. 4.3. First, we compare the predictive performance of a GP to our proposed method on a dataset with varied payload and terrain type. Second, we demonstrate the effectiveness of each component of our algorithm in closed loop. Finally, we demonstrate the path tracking performance of our algorithm at high speed on a 175 m off-road course.

4.6.1 Implementation

Our algorithm was implemented in C++ and the controller runs at 10 Hz with a three second look-ahead discretized by 30 points. The optimisation problem (2.8)-(2.11) is solved as a sequential quadratic program and re-linearized three times, taking an average of 70 ms to compute the control. The model updates (Sec. 4.3.3 and 4.3.3) are executed at every time step.

We consider the last three seconds of data (30 samples) from the live run for $\mathcal{D}_n^-$. The penalties on lag, contouring, heading, speed, and turn rate error are 50, 200, 200, 2, and 2 respectively. The penalties on commanded speed, turn rate, and reference speed from their references are 1, 1, and 50 respectively. The penalties on rate of change of commands in the same order are 10, 15, and 5. The maximum lateral error is 2 m, r^c is 1, and the ancillary controller gains are both -5. The prior strength, n_0, was set to 100. For the high speed experiment, we increased the penalty on commanded turning acceleration from 15 to 20 to achieve smoother performance on the rough terrain.

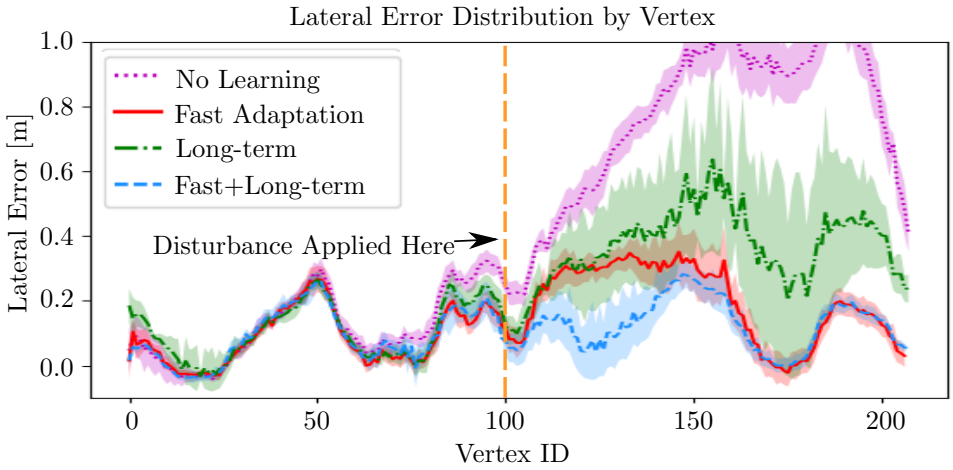


Figure 4.6: This figure shows the lateral error (measured using the position estimate from the vision-based localisation system) in a closed-loop experiment when we introduce a large, repetitive disturbance at vertex 100 by multiplying the turn rate commands by 0.5 after this point. This introduces a large, repeatable disturbance such as one might expect if the vehicle was traversing a patch of ice. The solid line indicates the median lateral tracking error over eight runs and the shaded region indicates the 50th and 75th percentiles. The proposed method with both long-term and fast adaptation learning achieves the lowest error and fastest convergence. No learning is when the controller uses a fixed wBLR model to compute the controls.

4.6.2 Closed-Loop Tracking Performance with a Path-Dependent Disturbance

To demonstrate the impact of each component of our method in closed loop and show that it can adapt to repetitive model errors, we drive the vehicle around two laps of a circular course and apply an artificial disturbance by multiplying the turn rate commands by 0.5 at the start of the second lap (vertex 100 in Fig. 4.6). An artificial disturbance was chosen over a large physical disturbance, such as inducing a flat tire, because it does not risk damaging the robot and is easily repeatable, which facilitates the comparison. We compare the tracking performance of each component of our algorithm over eight repeats of the path. For this experiment, the desired speed was 2 m/s.

Figure 4.6 shows that all methods achieve similar performance before the disturbance is applied because the model for all methods was a good representation of the vehicle dynamics over this portion of the path. After this point, the non-learning controller incurs a large lateral error because the model is no longer accurate. Long-term learning (Sec. 4.3.3) similarly incurs a large path tracking error on the first run (see Fig. 4.7) since there are no previous runs with experience. However, after the first run, it improves greatly but then converges slowly because it is constantly working against a static

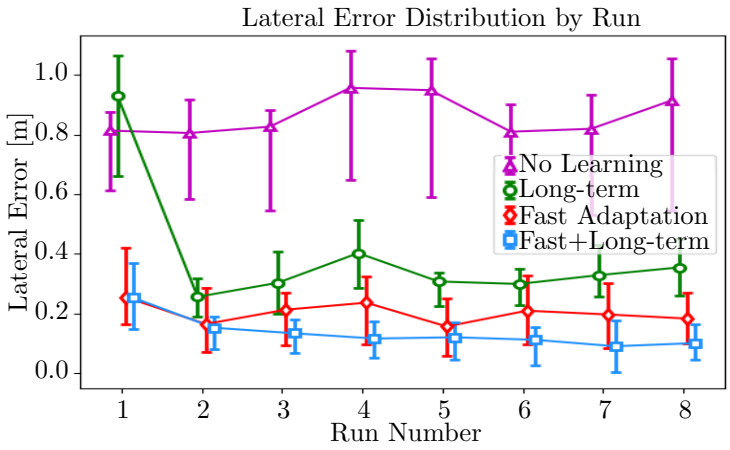


Figure 4.7: Figure showing the 25th, 50th, and 75th percentiles of lateral error after the large, repetitive disturbance described in Sec. 4.6.2 was applied. This figure shows that the proposed algorithm was able to quickly adapt to the disturbance and that the combination of fast adaptation (Sec. 4.3.3) and long-term learning (Sec. 4.3.3) achieves the best performance. No learning is when the controller uses a fixed, prior model to compute the controls. The horizontal position of each point is offset slightly for clarity.

prior (the same model used for the non-learning comparison), that is incorrect after the disturbance is applied. When fast adaptation (Sec. 4.3.3) is enabled, the controller incurs a large tracking error at the moment the disturbance is applied but adapts quickly to the new robot dynamics to achieve low error as expected. When both fast adaptation and long-term learning are enabled, the fast adaptation keeps the prior close to the true dynamics such that the long-term learning is able to reduce the transient error by leveraging data from the upcoming section of the path. This combination achieves the lowest path tracking error and the fastest convergence (see Fig. 4.7).

4.6.3 High Speed Tracking Performance

Finally, we evaluated the performance of our controller on a 175 m off-road course with tight turns and fast straights. The desired speed was 3 m/s and the controller achieved an average speed of 1.6 m/s with a top speed of 2.7 m/s and a RMS lateral error of 0.25 m. This is a 60% improvement over our previous work, where the controller achieved an average speed around 1.0 m/s on pavement [McKinnon and Schoellig, 2018].

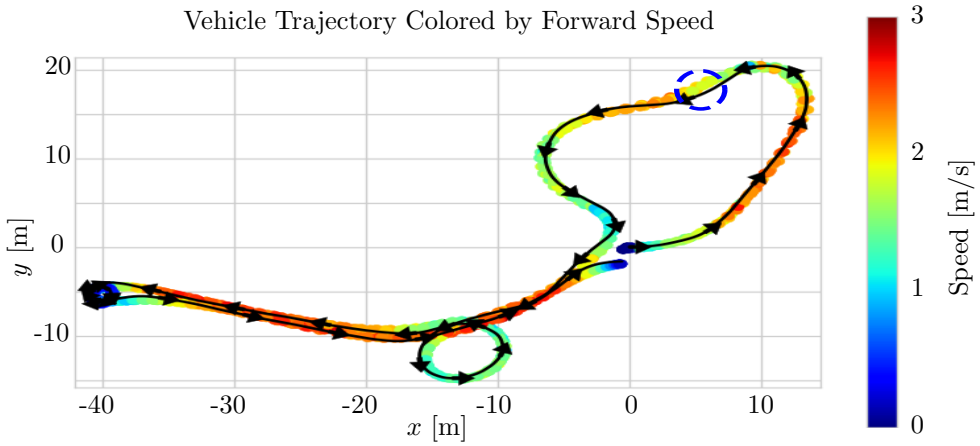


Figure 4.8: This figure shows the path taken by the vehicle on five traverses of a 175 m course. The direction of travel is indicated by the black arrows. The maximum path tracking error is 0.7 m when the controller cuts a corner (dashed blue circle). The vehicle was in the *Nominal* configuration.

4.7 Closed-Loop Comparison to Gaussian Process Regression

The experiments in this section were conducted on the same platform as the previous section. This section contains:

- A comparison of the closed-loop tracking performance of SMPC using a GP-based model and a wBLR-based model.
- An comparison of the model predictive performance linking this to the difference in closed-loop tracking performance.

The purpose of this section is to compare the performance of the stochastic MPC when using the GP- and wBLR-based models in *closed loop*. To test both methods in a challenging environment where the robot dynamics change, we drove the vehicle around a 43 m muddy path (depicted in Fig. 3.9) with varied target speed. As the target speed was increased, the vehicle incurred more slip which noticeably changed the dynamics. To simplify our implementation, we use the experience from the last run to construct the local GP used in the controller. To ensure that the most relevant data was from the last run, we increased the target speed gradually.

First, we compare the tracking performance at various target speeds. Next, we compare the model performance to link the mean and uncertainty estimates from each model to the resulting closed-loop performance.

4.7.1 Experimental Setup

Our experimental platform was the Clearpath Grizzly depicted in Fig. 4.3b. The controller was the SMPC described in Sec. 4.3.4 and McKinnon and Schoellig [2019a]. The controller parameters were kept constant and only the learning model and target speed were changed during the experiment. Two sets of GP hyperparameters were determined from a dataset where the wBLR-based controller drove the vehicle on the muddy test track: one with the vehicle driving with a target speed of 1.0 m/s (GP 1); and the other from the vehicle driving with a target speed varied between 1.0 and 2.5 m/s in increments of 0.5 m/s (GP 2). The hyperparameters were chosen to maximize the likelihood of a randomly chosen subset of data from each dataset. A constant prior was used to initialize wBLR that was identified from another dataset. This prior only matters for the first few seconds because it is constantly updated using data from the current run.

Our algorithm was implemented in C++ on an Intel i7 2.70 GHz 8 core processor with 16 GB of RAM. The solver used in SMPC was IBM's CPLEX [IBM]. The controller runs at 10 Hz with a look-ahead of 2 seconds, which was discretised with 20 points. This was reduced from the 3 s look-ahead used in the previous section to accommodate the more computationally expensive GP-based model. The local GP was constructed using $n_{gp} = 50$ past data points and $n_p = 5$. For wBLR, n_0 was set to 200. The maximum lateral error was set to 1.0 m and K_v and K_θ were set to -5. We rely on a vision-based system for localization [Paton et al., 2017], which runs on the same laptop and is accurate to approximately 5 cm (see Paton [2018], Fig. 4.17)..

4.7.2 Closed-Loop Comparison

Figure 4.9 shows that both models enabled the controller to perform well at low speed. However, for the GP-based models, the tracking error increased quickly with the target speed. GP 1 failed during run 6 at a target speed of 1.5 m/s and GP 2, trained on data from higher speeds, failed during run 7 with a target speed of 2.0 m/s. When the controller used the wBLR-based model, the tracking error increased with the desired speed, but the vehicle was able to traverse the path safely and reliably for all target speeds. In the next section, we analyse the prediction accuracy of the two models and show how it is linked to the difference in performance.

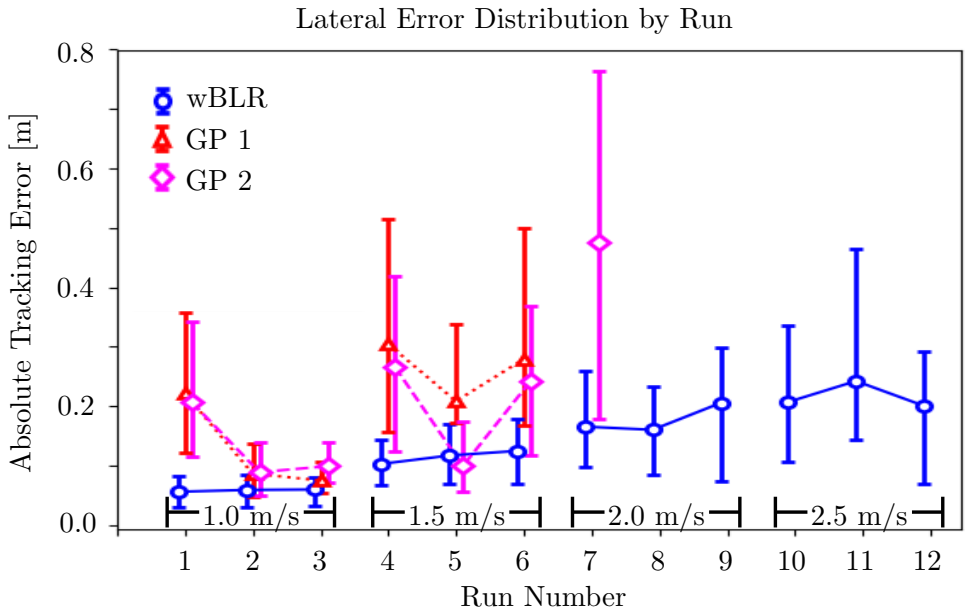


Figure 4.9: This figure shows the 25th, 50th, and 75th percentiles of absolute lateral tracking error when the controller used each model as the target speed was increased. When the controller used the wBLR-based model, it was able to drive the vehicle at a significantly higher target speed while maintaining low path tracking error. While the controller performed well at low speed using the GP-based models, the performance quickly degraded as the target speed was increased leading to failures on runs 6 for GP 1 and 7 for GP 2.

4.7.3 Multi-Step Prediction Performance

In this section, we compare the prediction performance of the best GP-based model (GP 2) and the wBLR-based model offline. Our performance metrics are the Multi-step RMSE (M-RMSE), which is the RMS prediction error over the MPC look-ahead horizon, and the Multi-step RMSZ (M-RMSZ), which is the RMS Z-score over the MPC look-ahead horizon [McKinnon and Schoellig, 2018]. Ideally, the M-RMSE should be low and the M-RMSZ should be close to 1.

Figure 4.10 shows that the predictive performance of both models is comparable when the vehicle drives up to 2.0 m/s. The GP takes one run to adapt because it only uses data from previous runs. This delay is likely what caused the failure for GP 2 at 2.0 m/s. Both metrics increase gradually with speed for wBLR, but the M-RMSZ is around 1.0 indicating that the model uncertainty estimate is still realistic. This indicates that while the wBLR-based model appears to make stronger assumptions on the form of the model, being able to adapt all model parameters including the uncertainty online and use data from the current run enables this model to adapt to novel scenarios better than the GP-based models.

While it may be possible to devise a strategy for the GP-based model to leverage

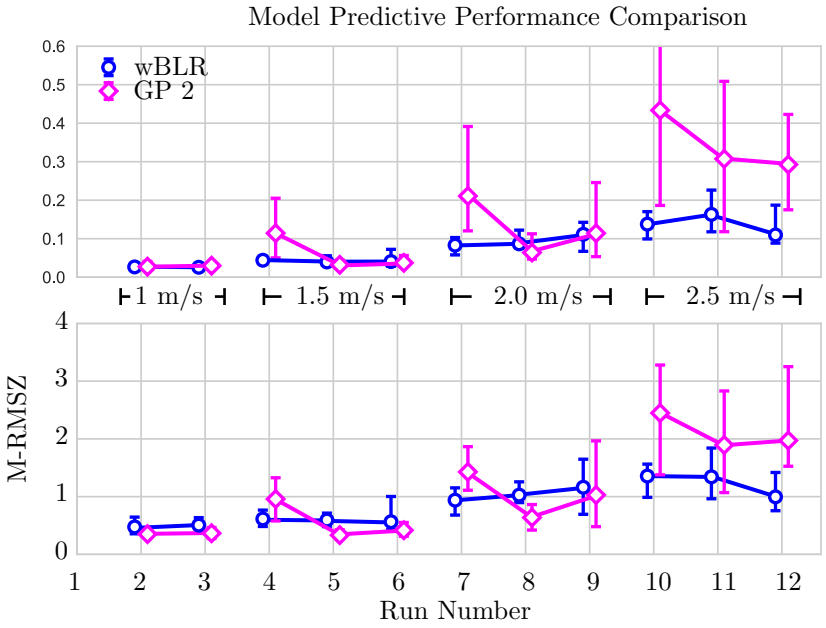


Figure 4.10: This figure shows the 25th, 50th, and 75th percentiles of M-RMSE and M-RMSZ for the GP and wBLR-based models at varied target speed. These values are calculated offline based on the data from the wBLR-based controller driving the vehicle. This shows that while both methods perform well at low speed, the wBLR-based model continues to perform well at high speed indicating better generalization to new conditions. The GP-based model produces higher error and more overconfident error estimates when the speed changes and at higher speeds because it does not have a fast adaptation term or adjust the estimate of σ^2 like wBLR. This likely contributed to the controller using the GP-based model to fail during run 7 (see Fig. 4.9).

data from the current run and avoid the large increases in prediction error when the speed changes, the fact that the wBLR-based model achieves similar accuracy to the GP during runs where the GP was using data from a run at the same speed (e.g., runs 5, 6, 8 and 9) indicates that the difference in accuracy between the two methods would be small. This, combined with the computational efficiency of wBLR and its ability to automatically adjust the aleatoric uncertainty online make it an attractive alternative to GPs for systems where the dynamics can be well approximated by a linear combination of basis functions.

4.7.4 Speed Comparison

Figure 4.11 shows the *actual* speed of the vehicle when the controller used the GP 2- and wBLR-based models. The average speed achieved by the better of the GP-based controllers (GP 2) during runs 4-6 was 1.00 m/s. The wBLR-based controller was slightly

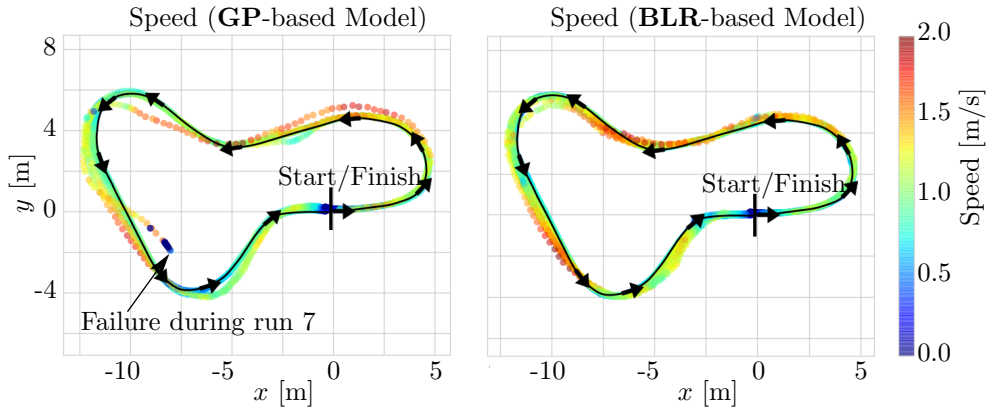


Figure 4.11: A top-down view of the path taken by the vehicle coloured by speed for each run over our 43 m test track when using GP 2 (left) and wBLR (right) to model the robot dynamics. This shows that the wBLR-based method was able to drive the vehicle faster than the GP-based method consistently as well as maintaining low path-tracking error. The black line indicates the reference and the arrows along the path indicate the direction of travel.

slower during runs 4-6 at 0.92 m/s, but increased this to 1.08 and 1.09 m/s during runs 7-9 and 10-12 respectively (9 % faster than the GP). The fact that the speed did not increase much between runs 7-9 and 10-12 indicates that the controller is making use of the uncertainty estimate to limit the maximum speed of the vehicle in order to maintain safety. This shows that the wBLR-based model enabled the controller to drive the vehicle *safely* and *reliably* at equal or higher speeds than the GP-based models in these challenging conditions.

4.8 Summary

In this chapter, we presented a novel method for modelling robot dynamics in changing conditions using a new form of Bayesian linear regression. This novel method improves on the work in the previous chapter in two important ways. First, when novel dynamics are encountered, the method presented in this chapter uses the live stream of data being generated by the robot to adapt the dynamics model continually rather than reverting to a conservative prior and waiting to leverage this new data on the subsequent run. Second, the BLR-based model can adjust the aleatoric model uncertainty online without computationally- and data-intensive hyper-parameter optimisation required by GPs. Through experiments on the Clearpath Grizzly, we showed that the proposed method could achieve comparable or better predictive performance compared to our previous GP-based algorithm, and that this improvement translated into better closed-loop tracking performance. The BLR-based method was initially presented at International

Conference on Robotics and Automation and published in the Robotics and Automation Letters [McKinnon and Schoellig, 2019a]. A follow-up study comparing the closed-loop performance of SMPC using a BLR-based model and a GP-based model was published at the European Controls Conference [McKinnon and Schoellig, 2019b].

Chapter 5

Meta-Learning with Paired Forward and Inverse Models

In the previous chapter, we presented a method for learning robot dynamics based on a novel form of Bayesian linear regression. This enabled us to adjust the aleatoric uncertainty online and adapt to new conditions continuously. The main drawback compared to GP-based methods is that they require the unknown dynamics to be close to a linear combination of known basis functions. While this does make good use of prior knowledge we might have about the system, it leaves a limited scope to leverage data that might be available beforehand to learn better basis functions to represent the robots dynamics. The work in this chapter is intended as an initial investigation into an approach for solving this issue. The scope was chosen in order to make it possible to conduct this investigation in the remaining time for this thesis.

In this chapter, we present a novel method for learning the basis function that relates the control input to the robots motion with an expressive function approximator, which may be relatively computationally expensive to evaluate. We show how this approach can be used in an SMPC framework with only a small incremental increase in computational cost. This method can leverage data and/or prior knowledge from multiple sources by choosing the one that best explains the current dynamics of the robot. This method retains the ability to adjust the aleatoric uncertainty online and adapt to new conditions continuously. Although we do not leverage the data from multiple repeats of the path once the robot is deployed, this is not a limitation of this method. Rather, it was a choice made to focus our experiments on the novel component of the model learning algorithm.

5.1 Introduction

The method in this chapter is based on the assumption that the control input affects the robot dynamics through an unknown (but invertable) nonlinear function. By learning this unknown function and its inverse, we can use the output of the function as a new input (which we call the input feature) that is optimised by SMPC in place of the original control input. The input feature can be used as a basis function in addition to others that represent our prior knowledge about how the robot dynamics might change. The key advantage of learning a forward and inverse model for the input feature is that this removes the need to evaluate a function approximator for the unknown function during optimisation in SMPC (where it would be evaluated many times at each sampling time). This substantially reduces the computational cost. The learned inverse is evaluated only once at each sampling time to convert the optimal input feature from SMPC to a control input to apply to the system. We assume that the remaining unknown dynamics can be accurately represented as a model that is linear in a set of coefficients, which enables fast adaptation to new conditions. Partitioning of the model in this way enables us to match the complexity each part of the model to the availability of data and computational power before and during deployment. We use expressive forward and inverse input-feature models that have large *data capacity* and are trained *offline*, and a model that is linear in a set of coefficients that is *data efficient* and trained *online*. A simplified block diagram of the proposed architecture is shown in Fig. 5.1.

This chapter is structured as follows: Sec. 5.2 discusses related work, Sec. 5.3 presents the problem statement, Sec. 5.4 details our approach, Sec. 5.6 presents experimental results showing our method on a Clearpath Grizzly ground robot, Sec. 5.7 discusses the practical benefits and remaining challenges of our approach, and Sec. 5.8 briefly summarises the novel contributions of this chapter.

5.2 Related Work

The most popular approach to improve SMPC is to learn the forward model for the robot dynamics directly. To increase the accuracy of this model, researchers leverage various function approximators from machine learning, such as Gaussian process regression [Hewing et al., 2018b, McKinnon and Schoellig, 2018, F. Meier and S. Schaal, 2016, Niekerk et al., 2017, Kabzan et al., 2019], local linear regression [Jamone et al., 2014, Desaraju et al., 2017, Ting et al., 2008, McKinnon and Schoellig, 2019b], and neural networks [Williams et al., 2017, Mohajerin and Waslander, 2019] to learn an unknown function

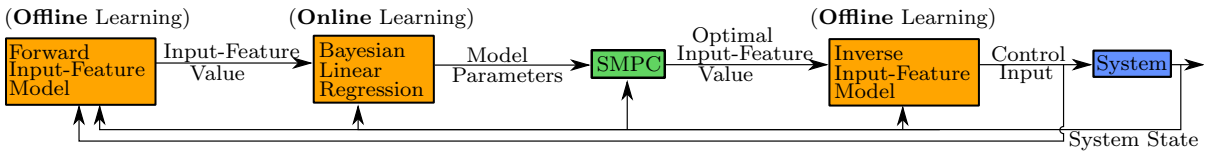


Figure 5.1: A block diagram of the proposed approach. The forward and inverse input-feature models are learned offline and Bayesian Linear Regression is used to adapt to changes online and estimate model uncertainty. The input feature is the value of an unknown function in the dynamics for which we have learned an approximate forward input-feature model. The value of this unknown function is optimised in SMPC and then converted into a control input that can be applied to the system using a learned inverse for the forward input-feature model. The main benefits of this approach are that *(i)* the expressive but computationally expensive forward input-feature map does not have to be evaluated during optimisation in SMPC, and *(ii)*, multiple forward and inverse input-feature maps from different sources can be considered simultaneously. A detailed version of this block diagram that includes online model selection is shown in Fig. 5.2.

in the forward dynamics model. However, SMPC requires these models to be evaluated many times to compute each control input. For example, for a prediction horizon of 3 seconds and a sampling period of 0.1 seconds, the forward model must be evaluated at least 30 times to compute each control input. Because of this, there is a trade-off between the computational cost of these function approximators and their expressiveness. In our approach, we only require the inverse input-feature model to be evaluated once for each control input, mitigating this problem. Additionaly, we use a simple model for robot dynamics in SMPC that can quickly adapt to changes in the robot dynamics. Partitioning the model into an expressive component learned offline and a simple component learned online to adapt to changes is inspired by a recent trend known as meta learning.

In meta learning, algorithms leverage large amounts of data available ahead of time to learn complex functions that can be adapted to new scenarios with a small amount of data and computation. An expressive model, such as a neural network, generates features [Harrison et al., 2018, O'Connell et al., 2020] or an initial guess [Finn et al., 2017] that can be adapted to changes using linear regression [Harrison et al., 2018, O'Connell et al., 2020] or a small number of gradient descent steps [Finn et al., 2017]. Functions learned using these methods adapt to changes efficiently because linear regression or a small number of gradient steps require a relatively small amount of data and computing power. These algorithms have been applied to reinforcement learning on real robotic platforms, used to learn forward models, and these forward models have been combined with robust control [O'Connell et al., 2020] and SMPC [Lew et al., 2019]. The main difference in our approach is that in addition to partitioning the model, we learn an inverse input-feature map, so we do not have to use the full forward model during optimisation in SMPC.

While a forward model predicts the motion of a robot given the current state and control input, an inverse model predicts the control input that will produce the desired motion given the current state. Inverse models have long been used for controlling platforms such as robotic manipulators [Spong et al., 2020]. These models can be derived from physics [Spong et al., 2020], learned [Zhou et al., 2017, Meier et al., 2016, Calandra et al., 2015b, Polydoros et al., 2015], or some combination [Nguyen-Tuong and Peters, 2010, Sun de la Cruz, 2011, Helwa et al., 2019]. The advantage is that they simplify complex, nonlinear dynamics into a simple linear system. However, they typically do not leverage an MPC framework, which requires a forward model, and they adapt the inverse term (a complex, nonlinear function) directly to adapt to changing conditions. In contrast, by learning both a forward and inverse input-feature map and combining this with linear regression, we simplify the online learning process to linear regression (which keeps the potentially challenging step of learning the inverse as an offline process where it can be validated before deployment) and leverage an SMPC framework.

As an alternative to learning one model that adapts efficiently to new scenarios, several forward/inverse model pairs can be trained ahead of time. The one that performs the best in the robot's current operating conditions is selected at runtime [Aoude et al., 2013]. This can be combined with online learning to adapt further to specific operating conditions [Pautrat et al., 2017, Sorocky et al., 2020, McKinnon and Schoellig, 2017]. While multiple paired forward and inverse models have been used for control before [Wolpert and Kawato, 1998], our approach is, to the best of our knowledge, the first that combines this capability with probabilistic, online model learning and SMPC.

Finally, another alternative to making the dynamics model more computationally efficient is to leverage advances in parallel computing hardware. Graphics Processing Units (GPUs) are becoming increasingly available on embedded computers because of their utility for perception algorithms that use deep learning and can accelerate computations that can be executed in parallel. While predicting a trajectory (and the associated uncertainty) is sequential in nature because the state and input at one sampling time affects the state at the following sampling time, it is often the case that evaluating the mean of a model is computationally cheaper than querying the model uncertainty (or its Jacobian). For example, evaluating the mean of a GP fit to N data points is $O(N)$, while evaluating the variance is $O(N^2)$. Therefore, it may be beneficial to predict the mean states in the trajectory sequentially and then, given the sequence of inputs and mean states, compute the model uncertainty and Jacobians at each timestep in parallel on a GPU. Given the model uncertainty and Jacobian at each timestep in the prediction horizon, uncertainty propagation (by linearisation) only involves a sequence of low-dimensional matrix mul-

tiplications, which is computationally inexpensive. The downside of utilising the GPU for control is that it is the dominant computing resource required by many perception algorithms that use deep learning. Using a computationally efficient approach that can run on a CPU will keep these resources available for other tasks that are essential for a mobile robot to operate in a complex environment. In addition, a more expressive model may be less data efficient and therefore require more data to adapt to changes in robot dynamics. Nevertheless, we would encourage a reader of this thesis to consider this direction as a possibility for future research.

5.3 Problem Statement

The goal of this work is to learn a model for robot dynamics where the robot is performing a path following task in changing conditions and the control approach is SMPC. The key requirements for the model are: *(i)* high accuracy predictions of the robot motion given the current state and control input, *(ii)* realistic bounds on modelling error to maintain safety, and *(iii)* computational efficiency to allow for long prediction horizons in SMPC. We assume that a geometric path and input and state constraints (e.g., maximum speed and lateral error) are given.

We consider the robot dynamics to be of the form where part of the state $\mathbf{s}$ evolves according to known dynamics $\mathbf{h}(\mathbf{s}, \xi)$ and part of the state ξ evolves according unknown dynamics. We assume the unknown dynamics are a linear combination of a known set of features $\boldsymbol{\phi}(\mathbf{s}, \xi)$ and an unknown but smooth and invertible function $f(\mathbf{s}, \xi, u)$ that depends on the states and the control input u:

$$\dot{\mathbf{s}} = \mathbf{h}(\mathbf{s}, \xi) \tag{5.1}$$

$$\dot{\xi} = [f(u, \mathbf{s}, \xi),\ \boldsymbol{\phi}^T(\mathbf{s}, \xi)]\ \mathbf{w} + \eta, \tag{5.2}$$

where $\mathbf{w}$ is a vector of unknown weights, and $\eta \sim \mathcal{N}(0, \sigma^2)$ where σ^2 is unknown and slowly changing. The reason for including $\boldsymbol{\phi}(\cdot)$ in addition to $f(\cdot)$ is to allow parts of the dynamics that depend on each element in $\boldsymbol{\phi}(\cdot)$ and $f(\cdot)$ to vary independently through the associated weight in $\mathbf{w}$ (facilitating online adaptation). That is, $\boldsymbol{\phi}(\cdot)$ represents prior knowledge about ways that the dynamics can change in new operating conditions and $f(\cdot)$ allows us to leverage prior data to learn unknown dynamics in a particular set of operating conditions. In the case where we have no prior knowledge, $\boldsymbol{\phi}(\cdot)$ can be omitted. There may be multiple states with unknown dynamics (e.g., ξ may be a vector), however we assume that each one depends on a separate control input u. In this case, there would

be one independent instance of (5.2) for each control input.

Simple Example To further motivate this problem setup, we present a simple example. Consider a cart on a rail with forward velocity v, mass m, viscous drag with drag coefficient b, input u, which is related to a force through an unknown function $q(u)$, and a disturbance with units of force which takes the form of zero-mean Gaussian noise $\eta_0 \sim \mathcal{N}(0, \sigma_0^2)$:

$$m\dot{v} = q(u) - bv + \eta_0. \tag{5.3}$$

Let $f(u, v) = \frac{1}{m}q(u) - \frac{b}{m}v$. This is the acceleration of the cart in terms of the velocity and the control input. Then the dynamics of a cart with any mass and drag coefficient can be expressed as:

$$\dot{v} = [f(u, v),\ v]\ \mathbf{w} + \eta. \tag{5.4}$$

This is of the form (5.2) with $\xi = v$. Here, including v in addition to $f(\cdot)$ enables changes in both m and b to be captured by $\mathbf{w}$ rather than just changes in m. Learning the entire right hand side of (5.3) (mass-normalised) as $f(\cdot)$ means that we do not have to know the initial value of m or b. While $f(\cdot)$ can be learned from data available ahead of time, changes in m, b, and the variance of η can be learned online efficiently through Bayesian linear regression.

The model above can be extended to a unicycle using (5.1) as follows. Let ω be the turn rate and assume that it is constant in the example. Let x and y be the position of the robot and θ be its heading. Then:

$$(5.1)\ \begin{cases} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix} \end{cases} \tag{5.5}$$

$$(5.2)\ \begin{cases} \dot{v} = [f(u, v),\ v]\ \mathbf{w} + \eta. \end{cases} \tag{5.6}$$

which is of the form (5.1) and (5.2) with $\mathbf{s} = [x, y, \theta]^T$ and $\xi = v$. For this example, changes in the m and b will only affect $\mathbf{w}$ and σ^2. As a result, it is sufficient to learn $\mathbf{w}$ and σ^2 online.

Additional examples of systems with dynamics that can be factored such that model parameters appear linearly include serial manipulators, pendulums, cartpoles [Xie et al., 2016], and quadrotors with unknown mass and inertia properties [Hoffmann et al., 2007] or quadrotors which have nested control loops that have dynamics close to first order systems [Augugliaro and D'Andrea, 2013]. The nonlinearity $q(\cdot)$ could come from, for

example, a DC motor with a nonlinear saturation curve (see Hagenmeyer and Delaleau [2003] Sec. 8, Fig 4).

5.4 Methodology

Our method partitions learning into two phases: an offline phase, where a forward and inverse model for the input feature are learned; and an online phase, where a simple model uses the input feature to adapt to changes in the robot dynamics and estimate model uncertainty. From a model learning perspective, the advantage of our approach is that the computationally and potentially data-intensive step of learning the forward and inverse input-feature models is done offline and the data-efficient and computationally-inexpensive step of updating the simple model is done online. From a computational perspective for MPC at runtime, trajectory optimisation (which requires many queries to the full model for robot dynamics) relies on the simple model, which is computationally inexpensive, and only optimises the *value* of the unknown function (i.e. the input feature). Generating the original control input using the more computationally expensive inverse input-feature model is done only once for each control input applied to the vehicle.

5.4.1 Approach

To present our approach, we focus on the unknown dynamics and assume $\xi \in \mathcal{R}$ for clarity of presentation. Our method can be applied to systems with $\boldsymbol{\xi} \in \mathcal{R}^{n_\xi}$ so long as each element depends on a separate input which allows them to be treated independently using our approach.

Forward Input-feature Model (Trained Offline)

We assume that we are given a dataset $\mathcal{D} = \left\{u_i, \mathbf{s}_i, \xi_i, \dot{\xi}_i\right\}_{i=1}^{n_\mathcal{D}}$ of $n_\mathcal{D}$ samples of the states, input, and time derivative of ξ at $n_\mathcal{D}$ different sampling times. Suppose that we have a system where the unknown dynamics are:

$$\dot{\xi} = [q(u, \mathbf{s}, \xi),\ \boldsymbol{\phi}^T(\mathbf{s}, \xi)]\mathbf{w}_0 + \eta_0. \tag{5.7}$$

where $\eta_0 \sim \mathcal{N}(0, \sigma_0^2)$, $q(u, \mathbf{s}, \xi)$ is an unknown function, and $\mathbf{w}_0$ is unknown but fixed. This is the same form as (5.2) except that we have $q(u, \mathbf{s}, \xi)$ instead of $f(u, \mathbf{s}, \xi)$. We can use $\mathcal{D}$ to fit a function $f(u, \mathbf{s}, \xi) \approx [q(u, \mathbf{s}, \xi),\ \boldsymbol{\phi}^T(\xi, \mathbf{s})]\mathbf{w}_0$, which we assume to be invertible. Learning the mean of the right hand side of (5.7) means that we do not have

to know the value of $\mathbf{w}_0$. For any changes to the weight or the variance of the additive, Gaussian noise, the dynamics will remain of the form:

$$\dot{\xi} = [f(u, \mathbf{s}, \xi),\ \boldsymbol{\phi}^T(\mathbf{s}, \xi)]\mathbf{w} + \eta. \tag{5.8}$$

Compared to (5.7), we have $f(u, \mathbf{s}, \xi)$, which is known instead of $q(u, \mathbf{s}, \xi)$, which is unknown. The weight $\mathbf{w}$ and the variance of η can be recovered online through Bayesian linear regression.

Inverse Input-feature Model (Trained Offline)

Let $\dot{\xi}^f \equiv f(u, \mathbf{s}, \xi)$. At a given time instant, ξ and $\mathbf{s}$ are fixed so the only parameter that we can change to influence $\dot{\xi}^f$ is u. Therefore, we can define:

$$u = f^{-1}(\mathbf{s}, \xi, \dot{\xi}^f). \tag{5.9}$$

which generates u that results in a desired $\dot{\xi}^f$ for a given $\mathbf{s}$ and ξ. To fit this function, we can use $f(\cdot)$ from the previous section to generate $\dot{\xi}^f$ for each sample in $\mathcal{D}$ yielding a new dataset $\mathcal{D}^f = \left\{u_i, \mathbf{s}_i, \xi_i, \dot{\xi}_i^f\right\}_{i=1}^{n_\mathcal{D}}$. This dataset can be used to fit $f^{-1}(\cdot)$ using standard regression techniques. Since $\mathcal{D}^f$ is generated using a deterministic function, this dataset is noise-free which facilitates learning $f^{-1}(\cdot)$. In cases where the inverse is not unique, approaches such as the distal-teacher approach can be employed to learn a particular inverse [Jordan and Rumelhart, 1992].

Simple Forward Model (Trained Online)

Given $f(\cdot)$ from the previous section, a tuple $(u_i, \mathbf{s}_i, \xi_i, \dot{\xi}_i)$, and $\boldsymbol{\phi}(\cdot)$, we can generate a corresponding tuple $(\dot{\xi}_i^f, \boldsymbol{\phi}_i, \dot{\xi}_i)$, where $\dot{\xi}_i^f = f(u_i, \mathbf{s}_i, \xi_i)$ and $\boldsymbol{\phi}_i = \boldsymbol{\phi}(\mathbf{s}_i, \xi_i)$ which can be used to fit a model of the form:

$$\dot{\xi}_i = [\dot{\xi}_i^f,\ \boldsymbol{\phi}_i^T]\mathbf{w} + \eta \tag{5.10}$$

which is linear in coefficients $\mathbf{w}$ with additive, Gaussian noise $\eta \sim \mathcal{N}(0, \sigma^2)$. Functions of this form can be learned efficiently using Bayesian linear regression, which estimates the distribution of $\mathbf{w}$ and σ^2. See Section 4.3 for details. Now, in combination with $\mathbf{h}(\cdot)$ from (5.1), we have all the components necessary to do trajectory optimisation in SMPC.

Stochastic MPC Cost Function

In this chapter, we use a slightly different cost function to the previous chapter to account for the fact that changing the input feature changes the meaning of one of the decision variables in SMPC. If we used the same cost function as previous chapters and penalised the value of the input feature and its time derivative, each input feature would have a different optimal trajectory making it difficult to compare the impact of different input-feature model pairs on closed-loop performance. Instead, we use a penalty on $\dot{\xi}$ and $\ddot{\xi}$. In the example of the unicycle in Sec. 5.3, this is the forward acceleration and jerk. This has the same effect as penalising the input feature and its time derivative in that it gives us a means to smooth out the control inputs applied to the vehicle but with the advantage that the cost function encourages the same motions regardless of the input feature. This is especially useful if we consider a mixture of input-feature model pairs, which we show in Sec. 5.4.2.

Input-feature Constraints All robotic systems have constraints on the maximum input u that can be applied, e.g., the maximum voltage that can be applied to an electric motor. If the system is likely to operate close to these constraints, we must translate the input constraints on the original input u into constraints on $\dot{\xi}^f$ for SMPC. The challenge is that u and $\dot{\xi}^f$ may be related through a complex, nonlinear function so linear or box constraints on u may translate into nonlinear constraints on $\dot{\xi}^f$. This is a problem that appears when applying feedback linearisation on nonlinear systems in conjunction with an MPC outer-loop controller: constraints on the output of the inner-loop feedback linearisation controller must be translated into constraints on the output of the outer loop MPC controller. In our case, the inverse input-feature map takes the place of the inner-loop feedback linearisation controller. There have been several approaches proposed to deal with this in the literature. Most approaches rely on the previous MPC solution to get an initial guess for the state and then either compute linear [Margellos and Lygeros, 2010] or box constraints [Deng et al., 2009, Kurtz and Henson, 1998] on the transformed input (i.e. the input feature). Alternatively, Mueller and D'Andrea [2013] takes fixed, conservative box constraints on the transformed input.

If the function $f^{-1}(\cdot)$ is close to a linear function, it may be possible to compute reasonable constraints on $\dot{\xi}^f$ by leveraging an approximate, linear model for $f^{-1}(\cdot)$, computing bounds on the approximation error, and finding the range of $\dot{\xi}^f$ for which this approximate model generates u within the acceptable limits, tightening the limits by the approximation error. Since we did not drive the vehicle close to its input constraints in our experiments, this was beyond the scope of the work in this chapter. However, this

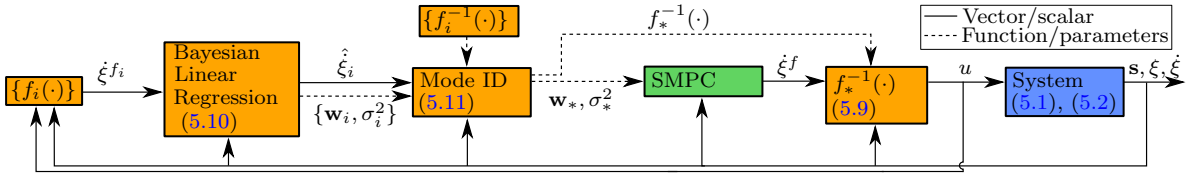


Figure 5.2: Block diagram of the proposed approach with multiple models. Curly braces indicate a discrete set of functions or values. Dashed lines indicate functions or parameters, and solid lines indicate scalar or vector quantities. The orange blocks are relevant for our model learning approach. The subscript i on $f_i(\cdot)$ and $f_i^{-1}(\cdot)$ indicates the mode. Equations relevant to each block are referenced in round braces.

would be a good next step for future work.

5.4.2 Multi-modal Learning

When a robot is deployed in a wide range of environments, there may be changes in the parameters of the linear model as well as the nonlinear term $f(\cdot)$. Changes in $f(\cdot)$ cannot be captured using Bayesian linear regression. A key advantage of our approach is that multiple models, each using a different input-feature model pair, can be simultaneously adapted to the robots current environment given a stream of $\mathbf{s}, \xi, \dot{\xi}$, and u. This allows us to train a set of input-feature model pairs ahead of time and choose the best pair for the current operating conditions.

Let c be a discrete index where each value corresponds to a different forward and inverse input-feature model pair and its associated BLR model and $med(\cdot)$ be the median function. At each sampling time k, the posterior probability of each model is calculated using a sliding window of n_w recent measurements $\mathcal{D}^- = \left\{\xi_i, \dot{\xi}_i, \mathbf{s}_i\right\}_{i=k-1-n_w}^{k-1}$ and a prior $p(c)$:

$$p(c=j|\mathcal{D}^-) \propto med\left(p(\dot{\xi}_i|u_i, \mathbf{s}_i, \xi_i, c=j)\right) p(c=j) \qquad i = k-1-n_w, ..., k-1 \quad (5.11)$$

where $p(\dot{\xi}_i|u_i, \mathbf{s}_i, \xi_i, c=j)$ is the probability of model j generating $\dot{\xi}_i$ given u_i, ξ_i, and $\mathbf{s}_i$. The input-feature is switched if $p(c=j|\mathcal{D}^-)$ for the most likely model is more likely than the previously most likely model by at least a pre-defined threshold. This prevents rapid switching between two or more models that have similar probability. Using the median instead of the product is a slight departure from the more common assumption that each recent measurement is independent but we found it produced more reliable results in experiment. We estimate the probability of each model using the model for the unknown dynamics only (i.e. (5.2) with a given input-feature model pair, set of weights, and noise

Figure 5.3: The Clearpath Grizzly with mast-mounted stereo camera used for the experiments in this paper.

variance) assuming that all models share a common $\mathbf{h}(\cdot)$. If this is not the case, the full dynamics model must be used.

5.5 Application to Grizzly

In this section, we describe how to apply our method to the Clearpath Grizzly shown in Fig. 5.3.

5.5.1 Dynamics Model

The dynamics model we use for the Clearpath Grizzly pictured in Fig. 5.3 is the same as in previous chapters, but with the input feature in place of the control input. Let x and y be the position of the vehicle, θ its heading, ω its turn rate, and v its forward speed. The model for robot dynamics is then:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{5.12}$$

$$\begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} [\dot{v}^f,\ v]\mathbf{w}^v + \eta^v \\ [\dot{\omega}^f,\ \omega]\mathbf{w}^\omega + \eta^\omega \end{bmatrix} \tag{5.13}$$

where $\dot{v}^f$ and $\dot{\omega}^f$ are the speed and turn-rate input feature, respectively, $\eta^v \sim \mathcal{N}(0, \sigma_v^2)$, and $\eta^\omega \sim \mathcal{N}(0, \sigma_\omega^2)$. Each row of (5.13) is analogous to (5.2), and (5.12) is analogous to

(5.1) with $\mathbf{s} = [x, y, \theta]^T$ and $\boldsymbol{\xi} = [v, \omega]^T$.

For the experiments in this paper, we considered two different input-feature model pairs for the speed and turn rate dynamics. First, as a baseline, we use the commanded speed v_{cmd} and turn rate ω_{cmd} directly. This the special case when the forward and inverse input-feature models are identity (used in our previous work [McKinnon and Schoellig, 2019a,b]). Second, we use a learned forward and inverse input-feature model pair where the forward input-feature models are:

$$\dot{v}^f = f_v(v, \omega, v_{cmd}), \tag{5.14}$$

$$\dot{\omega}^f = f_\omega(v, \omega, \omega_{cmd}), \tag{5.15}$$

and the inverse input-feature models are:

$$v_{cmd} = f_v^{-1}(v, \omega, \dot{v}^f), \tag{5.16}$$

$$\omega_{cmd} = f_\omega^{-1}(v, \omega, \dot{\omega}^f), \tag{5.17}$$

where each model $f_*(\cdot)$ and $f_*^{-1}(\cdot)$ is parametrised as a neural network. The network architecture we used in all cases was a fully connected network with three hidden layers, 20 hidden units in each layer, and ReLU activation functions. We trained all networks using 25,042 samples gathered in the UTIAS Mars Dome (sand and gravel terrain) with 20 percent of the samples used for validation and an L1 loss function.

5.5.2 Input-feature Model Accuracy

In the special case when the dynamics model is tested in the same environment as it was trained, the input feature should directly predict the associated motion in the robot (e.g., $\dot{v} \approx \dot{v}^f$). In Fig. 5.4a, we show that the forward input-feature model predicts a smoothed version of the acceleration of the vehicle. For eight runs of the vehicle driving in the UTIAS Mars Dome, the (50th, 75th, and 95th) percentiles of error between the input-feature (red in Fig. 5.4a) and the measured acceleration (blue in Fig. 5.4a) are $(0.074, 0.14, 0.29)$ m/s^2 for forward acceleration and $(0.064, 0.12, 0.23)$ rad/s^2 for turning acceleration.

Given the value of the input feature and the state at each sampling time, we can use the inverse input-feature model to predict the control input that generated the input-feature value at each sampling time. In Fig. 5.4b, we show that it accurately recovers the control input with error percentiles $(0.014, 0.017, 0.045)$ m/s for v_{cmd} and $(0.003, 0.005, 0.011)$ rad/s for ω_{cmd}. Furthermore, we can feed these control inputs back

into the forward input-feature model to see how much the input-feature changes when using the recovered control input (black in Fig. 5.4a) compared to the original control input. The percentiles of the change in the input-feature using the recovered vs. the actual control inputs are (0.007, 0.017, 0.036) m/s^2 for forward acceleration and (0.003, 0.006, 0.011) rad/s^2 for turning acceleration. Since these errors are approximately an order of magnitude smaller than the difference between the input-feature and the acceleration, we neglect error in the inverse input-feature models for our experiments.

The difference in accuracy between the forward and inverse input-feature models may result, in part, from the fact that the forward input-feature model is trained to model a physical process with an unknown dependence (the robot dynamics in the training environment). In contrast, the inverse input-feature model is trained to model a deterministic function with a known dependence (the inverse of the forward input-feature model). For example, in our case, the robot dynamics may be affected by local terrain properties, which may change from one run to the next as the vehicle displaces sand and gravel by driving over it. Since the forward input-feature model does not include local terrain properties as input, it cannot model the impact of these factors on the robot dynamics resulting in prediction errors. In addition, the output of the forward input-feature model is measured using sensors, which may introduce measurement noise. These problems do not exist for learning the inverse input-feature model since it is trained to invert the forward input-feature *model*, which is deterministic, has a known dependence, and is assumed to be invertible.

5.5.3 Cost Function

As mentioned in Sec. 5.4.1, the cost function should depend on the state and its derivatives and not the value of the input feature so that the optimal trajectory is consistent when the input-feature changes. For the Grizzly, we use a quadratic cost with penalties for position error, heading error, velocity error, acceleration, and jerk. Let $\mathbf{s}_k = [x_k, y_k, \theta_k]^T$ and $\boldsymbol{\xi}_k = [v_k, \omega_k]^T$. The penalty we used in MPC is:

$$\sum_{i=k+1}^{k+H} \ell_{lc}(\bar{\mathbf{s}}_i) + \sum_{i=k+1}^{k+H} \ell_{v\omega}(\bar{\boldsymbol{\xi}}_i) + \sum_{i=k}^{k+H-1} \frac{1}{2}\dot{\bar{\boldsymbol{\xi}}}_i^T \mathbf{R}_{\dot{\xi}} \dot{\bar{\boldsymbol{\xi}}}_i + \sum_{i=k}^{k+H-1} \frac{1}{2}\ddot{\bar{\boldsymbol{\xi}}}_i^T \mathbf{R}_{\ddot{\xi}} \ddot{\bar{\boldsymbol{\xi}}}_i \tag{5.18}$$

where $\ell_{lc}(\cdot)$ is a quadratic penalty on the lag, contouring, and heading error penalty, $\ell_{v\omega}(\cdot)$ is a quadratic penalty penalty for the speed and turn rate error, and $\mathbf{R}_{\dot{\xi}}$ and $\mathbf{R}_{\ddot{\xi}}$ are diagonal and positive semi-definite penalty matrices for acceleration and jerk, and $\bar{\cdot}$ is the mean of a random variable. We have omitted the terms of the cost function for

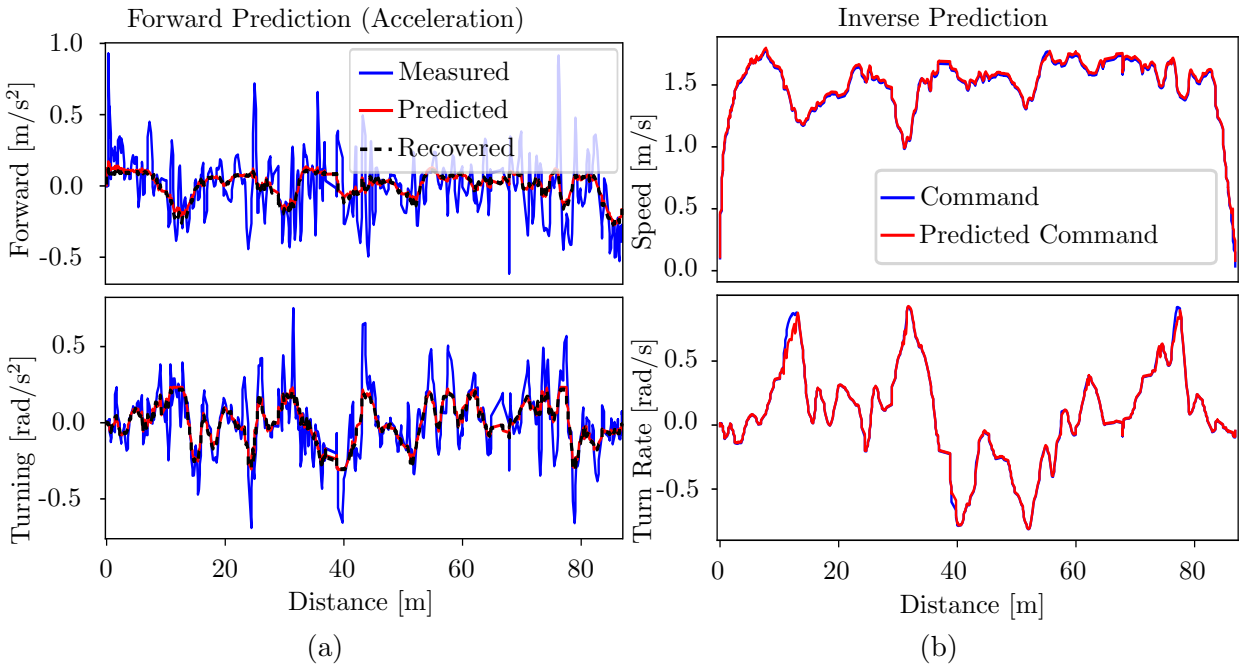


Figure 5.4: Measured values for acceleration and the control commands applied to the vehicle are shown in blue in 5.4a and 5.4b, respectively. Single step predictions from forward and inverse input-feature models are shown in red in 5.4a and 5.4b, respectively. In this special case since, the training and test environments were the same, the forward input-feature is analogous to acceleration. The dashed black line shows the value of the input feature recovered using the control commands predicted by the inverse input-feature model (e.g., $f_v(v, \omega, f_v^{-1}(v, \omega, \dot{v}^f))$). Since the difference between the input feature predicted using the original control inputs compared to the control inputs recovered by the inverse model is small relative to the difference between the forward input feature and the measured acceleration, we assumed that the inverse input-feature models were exact for our experiments.

contouring control (See Lam et al. [2010] for details) and the reference values for $\mathbf{s}$ and $\boldsymbol{\xi}$ for brevity. Values for $\dot{\bar{\boldsymbol{\xi}}}$ and $\ddot{\bar{\boldsymbol{\xi}}}$ are calculated by differentiating $\bar{\boldsymbol{\xi}}$ using finite difference.

The jerk at time k depends on the speed and turn rate of the vehicle at the previous timestep. We found that using the measured speed and turn rate at the previous timestep to calculate jerk introduced high-frequency noise in the control inputs which made it difficult to use high penalties on jerk to achieve smooth motions. Instead of relying on past measurements, we use the dynamics model, the previously applied input feature, and the current speed and turn rate to back-calculate the speed and turn rate at the previous timestep which mitigates this problem. While this introduces model error into the initial value for jerk, we found it enabled us to use higher penalties on jerk and achieve smoother closed-loop performance in our experiments.

For controller performance analysis in Figure 5.6b and 5.6c, we omitted terms in the cost function related to acceleration and jerk. This is because of the large magnitude of high frequency noise introduced into acceleration and jerk when these values were calculated using finite difference and the large impact filtering to remove this noise can have on the value of the cost function. In previous chapters, we did not encounter this issue because these terms were based on the control input, which is known exactly because it is output by the controller. While we do not have access to ground truth measurements of acceleration and jerk to examine the quality of our estimates of these values, we believe the remaining terms in the cost function still provide a meaningful assessment of controller performance. We also include lateral error as a complementary performance metric for tracking performance that is easy to interpret and independent of the cost function. The position estimate used to calculate lateral error is accurate to approximately 5 cm (see Paton [2018], Fig. 4.17).

5.5.4 Computational Advantage

A key advantage of our approach is that a simple model is used in SMPC and the complex forward and inverse models are evaluated only once per sampling time. This is important because the mean, uncertainty, and Jacobian of the model used in SMPC are queried 30 timestep prediction horizon x 3 re-linearizations x 10 Hz = 900 times per second whereas the scalar output of the forward and inverse models are evaluated at 10 Hz = 10 times per second. This means that the forward and inverse blocks can be much more computationally expensive (and therefore potentially more expressive) than any model used in SMPC. In addition, the computational cost associated with the forward and inverse input-feature models is does not grow with the length of the prediction horizon in

contrast to the model used in SMPC. In the experiments in this chapter, the BLR model has 14 independent parameters and the forward and inverse models have a combined 3764 parameters which reflects the required computational requirements of each component. In addition, the BLR model can provide a quantitative estimate of model uncertainty for SMPC. This allowed us to implement the forward and inverse model learning in Python on the CPU adding only 4.1 ms to 6.5 ms (50th to 95th percentile) to the run-time of the controller (small compared to the 100 ms sampling period of the controller).

5.6 Experiments

We compare four variants of the proposed method which are summarised in Table 5.1. In contrast to previous chapters, all of these methods only use data from the current run to adapt the model online. For this reason, we refer to the Bayesian Linear Regression component of the model as BLR and not wBLR, which is designed primarily to leverage a weighted combination of past data.

BLR: The method presented in Chapter 4. This is the special case when the forward and input-feature models are identity and only depend on the original control input.

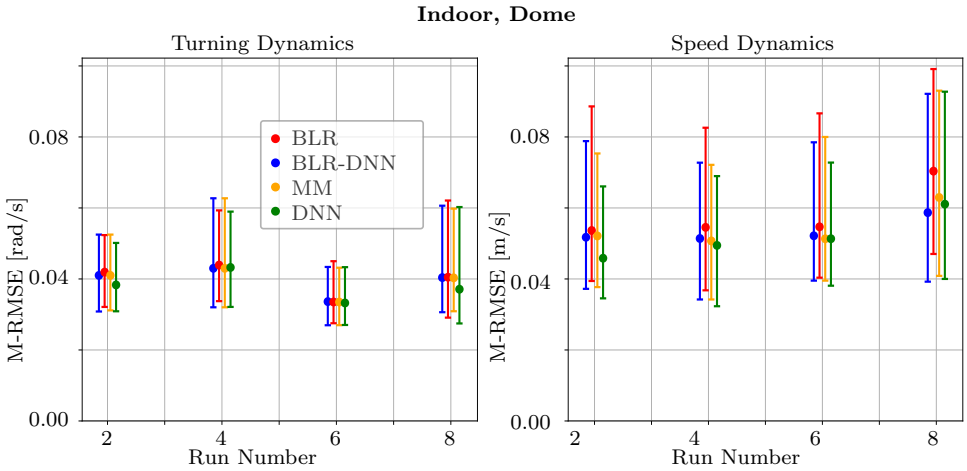
DNN: This method makes use of the input-feature but does not adapt the BLR model online.

BLR-DNN: This method makes use of both the learned input-feature and adapting the linear model online.

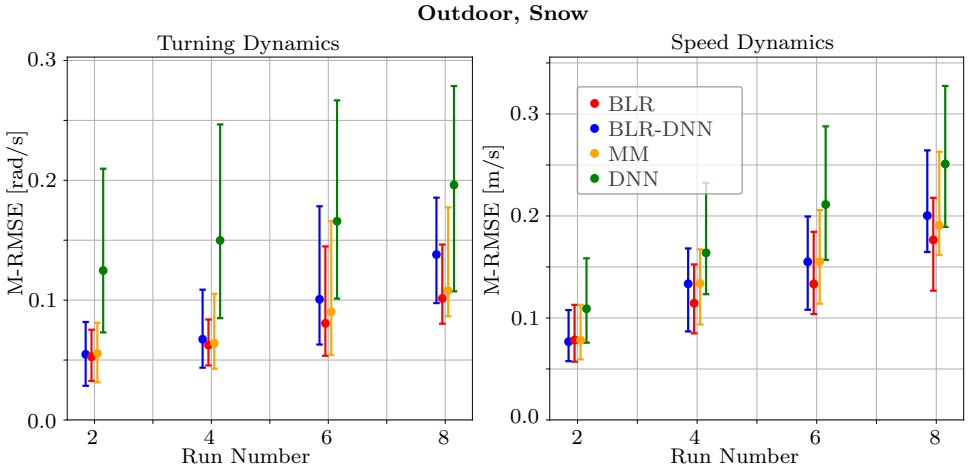
MM: This method includes BLR and BLR-DNN in a mixture model and automatically selects the best one online using (5.11) and a uniform prior.

These variants all use the unicycle model (5.12) but differ in whether or not a model is learned for the input features, whether the linear model parameters are updated online, and the number of input-feature model pairs considered at runtime. Initial values for the linear model parameters $\mathbf{w}$ are calculated using the dataset used to train the input-feature model pairs.

We use BLR with online adaptation as our base-line because improving on this method indicates an improvement over what can be achieved with the methods presented in the previous chapter. We include DNN, which does not have online adaptation, to illustrate the contribution of each component of the proposed algorithm. Further studies including a fixed model with conservative error bounds may result in valuable insights in the future.



(a) When the DNN is trained in the same conditions as the test environment, **BLR-DNN** performs similarly to **DNN** and both achieve lower M-RMSE than **BLR**, especially for the speed dynamics.



(b) When the DNN is trained in different conditions to the test environment, **BLR-DNN** significantly out-performs **DNN**. **BLR** sometimes achieves a lower M-RMSE indicating that the learned feature does not always generalize well.

Figure 5.5: This figures compares the M-RMSE of **DNN**, **BLR**, **BLR-DNN**, and **MM** in the UTIAS Mars Dome and outdoors on snow-covered terrain. The target speed for the Dome experiments is kept at 1.5 m/s for all runs whereas the target speed for the outdoor experiments is increased every two runs leading to the higher model error for later runs. In all cases, network used in **DNN** and **BLR-DNN** was trained using data from the UTIAS Mars Dome environment (Dome), which consists of loose sand and gravel.

Method	Learned Input Feature	Online Adaptation	Online Model Selection
BLR	x	✓	x
DNN	✓	x	x
BLR-DNN	✓	✓	x
MM	✓	✓	✓

Table 5.1: Summary of differences between the methods compared in this section. Learned input feature is when a learned input-feature model pair is used (Sec. 5.4.1 and 5.4.1). Online adaptation is when the linear model parameters are updated online (Sec. 5.4.1). Online model selection is when multiple input-feature model pairs are considered at runtime (Sec. 5.4.2).

In particular, such a model could be incorporated into the mixture model at relatively low computational cost.

5.6.1 Model Accuracy Comparison

We measure prediction accuracy of each model over the SMPC horizon using M-RMSE and M-RMSZ. For every fifth sampling time during each run, we predict the speed and turn rate over the next 3 seconds using the state measured at that sampling time as the initial condition and the control inputs actually applied to the vehicle over the prediction horizon. These predictions are compared to the measured speed and turn rate at the corresponding times.

Figure 5.5a shows that the DNN performs well when tested in the same environment that it was trained: all methods using a learned feature to model the robot dynamics have significantly lower M-RMSE for speed. This indicates that there are nonlinear components of the speed dynamics that cannot be modelled using the linear feature used in BLR. In addition, there is no significant difference between BLR-DNN and DNN, which indicates that local variations in robot dynamics that might be captured by the online adaptation in BLR-DNN do not contribute significantly to the prediction error when the input feature is used in the same environment it was trained.

Figure 5.5b shows that when the robot is deployed in different conditions to where it was trained, the online adaptation in BLR-DNN significantly reduces the prediction error compared to DNN for both speed and turn rate. In addition, while BLR does not perform as well in the Dome environment, it actually achieves lower M-RMSE in the snowy conditions, which are very different from the environment the DNNs used in DNN and BLR-DNN were trained in. This shows the value of including a simple model that may generalize better than one fit to specific conditions. For applications where the

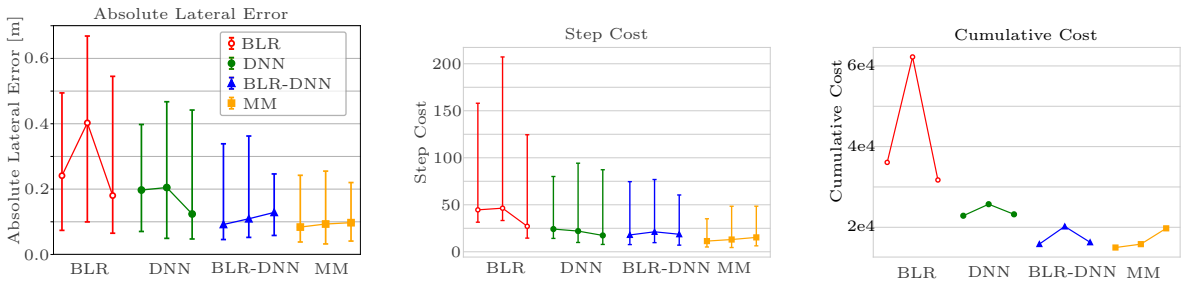
conditions a robot may encounter are initially unknown, this motivates using a mixture model to include either a reliable baseline model or, if data is available, a forward and inverse input-feature model pair trained on data from conditions more similar to the test conditions. In both Fig. 5.5a and Fig. 5.5b, MM performs similar or better than BLR-DNN most of the time showing the value of online model selection.

5.6.2 Closed-Loop Experiments

The experiments shown in the previous section established that using learned features instead a hand-crafted ones can improve model accuracy when the training and test conditions are similar. In addition, it showed that the ability of the proposed method to adapt the weights associated with the learned features online can improve performance when the robot is deployed in new conditions. In this section, we demonstrate the effectiveness of the proposed approach in closed loop. We use the same network used in the previous experiments, and conduct our tests over an 82 m course in a paved parking area. The high friction of the paved surface introduced the a larger difference between BLR and BLR-DNN than driving offroad on dirt or over snowy terrain.

Tracking Performance Comparison

Figure 5.6 shows the distribution of lateral error, step cost, and the cumulative cost when controlling the vehicle using BLR, BLR-DNN, MM, and DNN. All cases when the controller uses a model that incorporates a learned feature show a reduction in lateral error, step cost, and cumulative cost compared to BLR indicating that the learned feature captures more of the robot dynamics than the hand-crafted feature. Averaging over three runs, using BLR-DNN resulted in a 60% reduction in median lateral error, a 51% reduction in the median step cost, and a 60% reduction in cumulative cost compared to using BLR. Furthermore, BLR-DNN out-performs DNN in all metrics indicating that learning the feature weights online not only improves model accuracy when the robot is deployed in new conditions as we saw in Sec. 5.6.1, but that these improvements translate into better closed-loop performance. One interesting point is that MM achieved lower lateral error and step cost than BLR-DNN but achieved similar cumulative cost. This is likely due to our model switching strategy, which assumes that the acceleration and jerk are zero when the model switches which causes the vehicle to slow down. At this lower speed, the vehicle is able to track the path more closely which reduces the lateral error and step cost, but it takes longer to complete the path which increases the cumulative cost.



(a) (b) (c)

Figure 5.6: This figure shows performance metrics for a closed-loop experiment demonstrating the effect of each component of the proposed algorithm. The robot traversed an 82 m path over a paved parking area. Each point along the horizontal axis represents a traversal of the path. All traversals where the controller leveraged a model using the learned input-feature out-performed **BLR**, which does not us it. As we observed in the previous section, online adaptation (**BLR-DNN**) increased performance in all metrics compared to using a fixed model (**DNN**). The mixture model (**MM**) outperforms all models except for BLR-DNN, where it achieves a similar cumulative cost, for reasons explained in Sec. 5.6.2.

Closed-Loop Model Performance Comparison

In this section, we compare the prediction performance of BLR and BLR-DNN for the closed-loop experiment to gain insight into why their performance differed in closed loop. The results in this section are based on data from runs when the controller was using the corresponding method. We focus on the robots speed dynamics, but the results for turn rate show a similar trend.

Figure 5.7 shows that the M-RMSE and M-RMSZ for speed was substantially lower for BLR-DNN than for BLR. This discrepancy, which we did not observe in the offline model comparison of Sec. 5.6.1 highlights the importance of using a good feature for online learning in closed loop: large model errors induce tracking errors, which can result in larger control inputs, which can compound the model errors further as more complex dynamics are excited. While the controller using BLR does complete the course, the large M-RMSZ indicates that the uncertainty bounds are overconfident for several sections of the run, so the vehicle may not always be able to complete the run without violating the path tracking error bounds if the vehicle is close to the path tracking constraints at the same time that the model is overconfident.

Figure 5.8 shows the model parameters over the course of the run for BLR and BLR-DNN. For all three runs, the linear model parameters for BLR-DNN converge to similar values and the uncertainty converges to just under 0.15 m/s^2, which is much less than the model uncertainty for BLR. The parameters converging to a constant value for BLR-DNN is desirable since the online adaptation is retroactive; changes in the model parameters

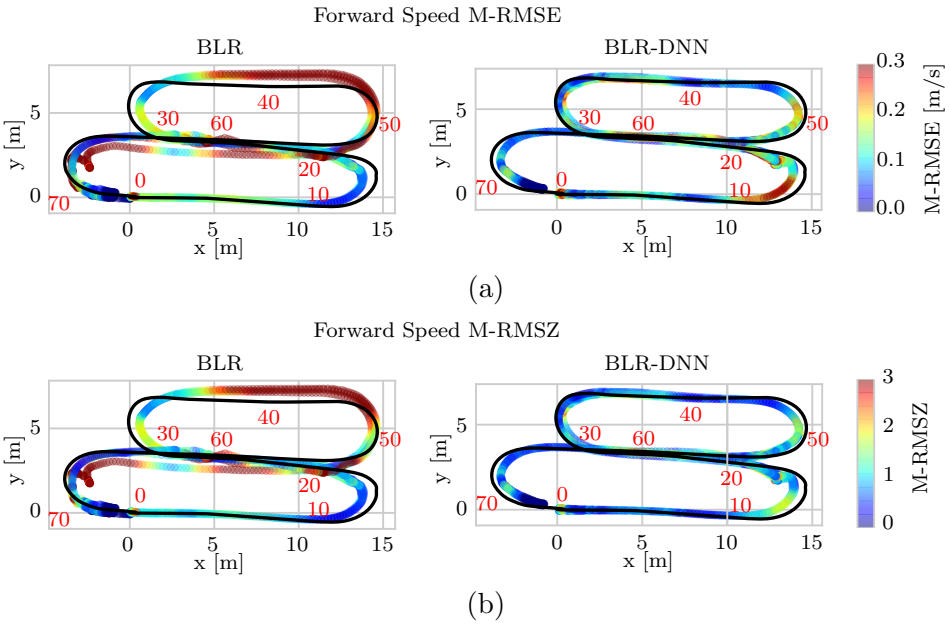
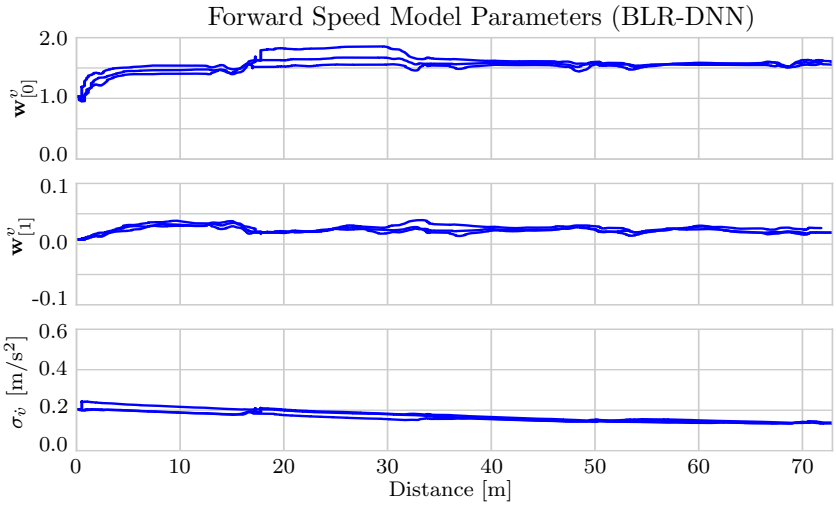


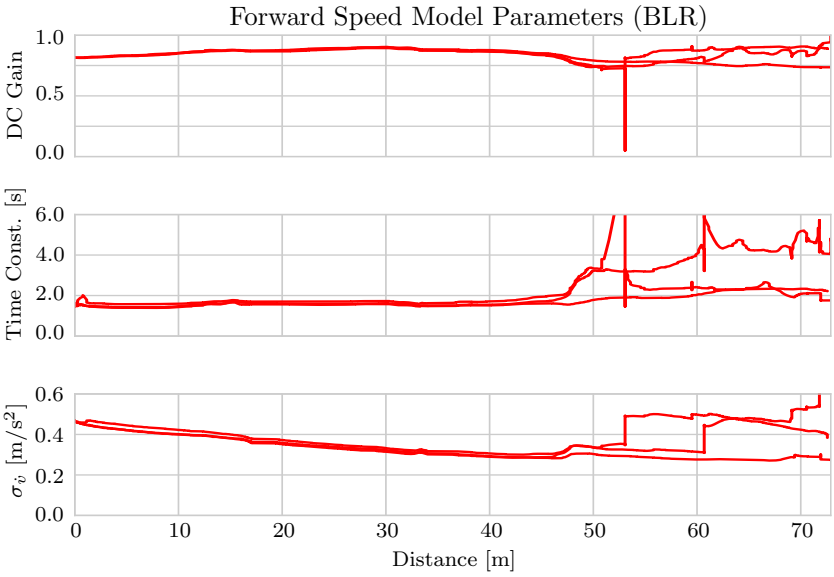
Figure 5.7: This figure shows the model prediction accuracy using the parameters estimated online when **BLR** and **BLR-DNN** are used in closed loop in a paved parking lot. Results for each method over three runs (the same runs shown in Fig. 5.6) are super-imposed. This shows that, in the parking lot environment, **BLR-DNN** clearly out-performs **BLR** in model prediction accuracy, uncertainty estimation, in addition to closed-loop performance as shown in Fig. 5.6.

indicate that the model was inaccurate for a short period of time leading up to the change. For BLR, the frequent changes in model parameters and large uncertainty indicate that the model was frequently inaccurate. The lower accuracy in Fig. 5.7 indicates that the online adaptation was not able to keep up with the changes in model weights required to model the dynamics using the BLR feature.

For each of BLR and BLR-DNN, the model weights have a physical interpretation. For BLR-DNN, referring to Fig. 5.8a, the fact that $\mathbf{w}^v_{[0]} \gg \mathbf{w}^v_{[1]}$, where $\mathbf{w}^v_{[0]}$ is associated with $\dot{\xi}^{f,v}$, indicates that the dynamics in the parking lot are largely a scaling of the dynamics in the dome and that terms associated with speed do not vary independently. $\mathbf{w}^v_{[0]} > 0$ indicates that the forward acceleration for the same input and turn rate is larger on pavement than in sand or gravel which is expected. For BLR, the model weights $\mathbf{w}$ can be related to the time constant and DC-gain of the system, which are shown in Fig. 5.8b for ease of interpretation.



(a) **BLR-DNN** model parameters during runs using the **BLR-DNN** model in Fig. 5.6.



(b) **BLR** model parameters and uncertainty during runs using the **BLR** model in Fig. 5.6.

Figure 5.8: The linear model parameters for **BLR** and **BLR-DNN** for the speed dynamics in closed loop.

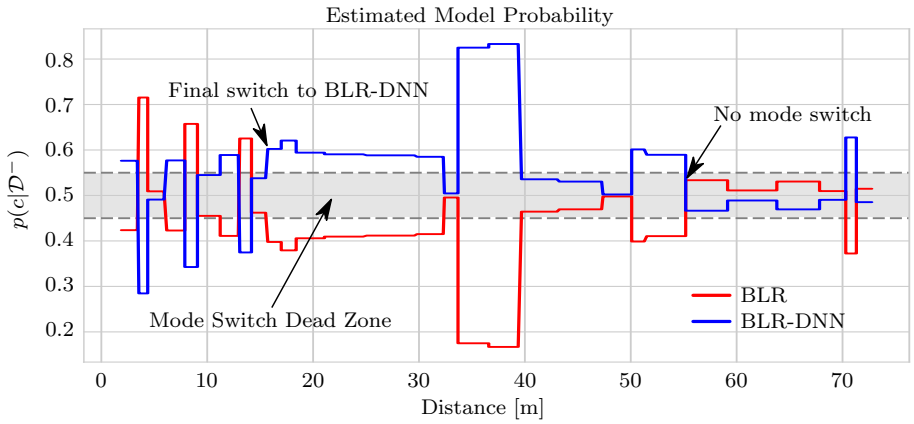


Figure 5.9: This figure shows the estimated probability of each model in the mixture model the first run using the MM of the closed-loop experiments shown in Fig. 5.6. The probability of each model is estimated every 2 s which is why the estimated probability has the appearance of a square wave with varying width. We avoid chatter by requiring that the model only switches when the most likely model is at least 10% more likely than the currently active model. For two models, this leads to the 'dead zone', shaded in gray, where no mode switches will occur even if the most likely model changes.

Online Model Selection

In this section, we illustrate how the model selection algorithm chooses the best model to use in the controller over the course of the run. Figure 5.9 shows the estimated probability of each model calculated using (5.11) during one run from the results shown in Fig. 5.6. BLR-DNN is estimated as the most likely model for most of the run, which is consistent with our the results in Fig. 5.6 where this model had better tracking performance, and Fig. 5.7, which showed that using BLR-DNN in closed loop resulted in higher model accuracy.

To prevent rapid switching between models when two models have similar probability, we require that the most likely model be at least 10% more likely than the one used in the controller before switching. This results in the dead-band shown in Fig. 5.9. When the estimated probability of both models is similar, each model explains the recent dynamics similarly well so using either should result in similar performance.

5.7 Discussion

An ongoing challenge in learning control is choosing the dependence of the unknown dynamics (i.e. the dependence of the input-feature model pair). Initially, it may be tempting to include many past states and rely on the function approximator to ignore irrelevant ones given enough data. We found that including past states as inputs to the

input-feature model pair model induced large oscillations in closed loop even when the optimal input-feature values from SMPC appeared to be smooth. The simpler model used in the experiments that only included states and inputs from one sampling time did not have this problem. A detailed study of feature selection for learning inverse models for closed-loop control is beyond the scope of this thesis. However, Zhou et al. [2020] also found improvements in performance using fewer inputs to their learning-based controller and derived a theoretical basis for choosing the inputs which may provide future researchers with further guidance in this area.

We found that the controller did not operate the vehicle near its input constraints so did not constrain the value of the input features during our experiments. Since this may not be the case for all systems, we have included a discussion of methods to adapt constraints on the original control input to constraints on the input features in Sec. 5.4.1. A follow-up study on efficient ways to map constraints on the control input to (approximate) constraints on the input-feature has potential to make the method presented in this chapter applicable to systems where the controller must use a large fraction of the control input available to it.

5.8 Summary and Contributions

In this chapter we presented a control approach specifically tailored for SMPC that leverages the combination of *(i)* expressive input features learned offline with *(ii)* Bayesian linear regression to adapt top changes online and *(iii)* online model selection to leverage multiple possible input feature models. Our formulation allows learned input features to be used as a drop-in replacement for the control input making it easy to apply to existing receding horizon controllers. We demonstrated our approach on a realistic robotic platform and provided a thorough analysis of the experimental results, showing a significant reduction in lateral tracking error, step cost, and cumulative cost to traverse a path. Compared to the previous chapter, we are able to use a more expressive model to learn the robot dynamics by learning a basis function that relates the control input to the robots motion rather than relying on a hand-specified one. Compared to existing meta-learning approaches, we leverage the combination of forward and inverse models to mitigate the cost of using an expressive model in SMPC. To our knowledge, we are also the first to demonstrate the combination of multiple forward and inverse models with online adaptation and SMPC.

Chapter 6

Cost Learning for Model Predictive Control

In previous chapters, we present methods to improve the dynamics model used in SMPC and evaluated them through experiments. In this chapter, we explore how learning a correction to the cost function can be used to improve tracking performance in cases where the accuracy of the dynamics model varies due to factors that are not captured by the dynamics model.

6.1 Introduction

In previous chapters, we studied ways to improve the dynamics model for SMPC in order to make more accurate predictions about the robots motion and to estimate the uncertainty in these predictions. These methods continually adapt to changes in dynamics, leverage data from previous runs to adapt to repetitive changes in dynamics during repetitive path-following tasks, and leverage data available before deploying a robot to learn more expressive dynamics models at a marginal increase in computational cost. Despite these improvements, we still observed variable model prediction accuracy over the course of a run depending on factors such as the local terrain or the particular motion of the robot which can impact tracking performance.

At this point, we could embark on a quest to further increase the models accuracy by exploring a more flexible parametrization for the model or adding additional variables as input to the model. However, the expressiveness of the dynamics model, and hence its accuracy, will always be limited by computational resources (especially for SMPC), prior knowledge about the robot dynamics, and the availability of data ahead of time.

As an alternative to improving the dynamics model, in this chapter, we explore the use of the cost function to discourage the controller from applying control inputs that result in higher cost (according to the SMPC cost function) than predicted using the dynamics model. The SMPC cost function is a natural measure of tracking performance because it already has to be specified by the user and specifies the relative importance of multiple error metrics such as speed and lateral error. Our approach is based on modelling the difference between the cost predicted using the dynamics model and the cost incurred in closed loop. By adding a model of this difference to the cost function optimised by the controller, we enable the controller to account for errors in the cost predicted using the dynamics model and reduce the cost incurred in closed loop. This improves tracking performance in cases where the dynamics model is not accurate.

In addition, even with a perfect dynamics model, control performance can be affected by the localisation system, which provides the initial condition for predicting future states in SMPC; if the vehicles motion affects the performance of the localisation system, then it may be useful to for the controller to adjust the vehicles motion to reduce its impact on localisation performance. Through simulation, we show that the method developed in this chapter can improve closed-loop performance when the accuracy of the localisation system varies as a function of the vehicles speed even if the dynamics model is perfect.

6.2 Related Work

The most common way to improve the performance of SMPC is to improve the dynamics model, which has been the focus of the previous chapters of this thesis. Researchers in this area continue to demonstrate new and improved methods for learning robot dynamics. Many of these methods are based on Gaussian process regression [McKinnon and Schoellig, 2017, Hewing et al., 2018b, McKinnon and Schoellig, 2018, F. Meier and S. Schaal, 2016, Niekerk et al., 2017, Kabzan et al., 2019], local linear regression [McKinnon and Schoellig, 2019a, Jamone et al., 2014, Desaraju et al., 2017, Ting et al., 2008, McKinnon and Schoellig, 2019b], and neural networks [Williams et al., 2017, Mohajerin and Waslander, 2019]. All of these methods, however, limit the expressive power of the model in some way to meet the computational requirements of MPC, which requires solving a (nonlinear) optimisation problem at each sampling time. For GP-based methods, the number of basis points is limited and kernel hyperparameters are often fixed. Both neural networks and local linear regression assume a fixed form for the system dynamics in terms of a fixed feature or neural network architecture. This can limit the performance of these algorithms because the accuracy of the chosen function approximation

may vary–especially if the robot is deployed in changing conditions. In addition, such models are usually trained to predict for one timestep in contrast to the long horizons used in MPC. Making accurate multi-step predictions remains an active area of study [Venkatraman et al., 2014, Doerr et al., 2017].

Alternatively to improving the model for robot dynamics, the methods in [Rosolia and Borrelli, 2017, Rosolia et al., 2017a, Lowrey et al., 2018] learn the value function–the discounted sum of rewards from a given state to the completion of the task–from data and leverage MPC for efficient short-term trajectory optimisation. In Rosolia and Borrelli [2017], Rosolia et al. [2017a], the authors iteratively expand a set of sampled safe states–and their associated value–so that an MPC-based controller can minimize the total cost to complete the task by using a model-based prediction over the MPC horizon and the value of the sampled safe states for the cost of the final state in the horizon. In Lowrey et al. [2018], the authors learn the value function using function approximation and showed how combining it with receding horizon control could accelerate learning. While estimating the value function using data does improve performance, these methods still rely on a dynamics model for making predictions over the MPC horizon. In contrast, we correct the cost over the MPC horizon to account for varied model accuracy. We also account for changes in operating conditions by inferring which previous runs resulted in similar cost prediction error to the current run.

In addition to learning just the value function, Tamar et al. [2017] proposed a method to modify the cost function of MPC over the prediction horizon and demonstrated it in simulation. The goal was to enable a short-horizon controller to mimic the behaviour of a longer-horizon controller computed offline using a better model for robot dynamics. In this case, the long-horizon controller was still model-based. In contrast, we correct the cost over the MPC horizon to account for cost prediction errors based on past experience directly.

Finally, for the specific case of speed scheduling for repetitive path following, Ostafew et al. [2014b] designed a speed scheduling heuristic to increase or decrease the target speed of a path-following controller to take into account tracking error and aspects of the performance of a vision-based localisation system. While this was effective for the purpose it was designed for, it involved adding many manual tuning parameters which were not linked to the control cost. Choosing these parameters required detailed knowledge about the performance of the localisation system. In contrast, we propose a method that directly uses the existing control cost as an objective to minimise.

Our method is inspired by ideas from reward shaping, whereby the reward function (which provides rewards for incremental actions) is modified to encourage and discourage

behaviours that lead to high and low reward respectively [Ng et al., 1999]. Reward shaping has been shown to be effective for model-based learning in simple scenarios where safety is not a major consideration [Asmuth et al., 2008] as well as for episodic tasks [Grześ, 2017] like repetitive path following. Our method builds on these ideas to improve performance of a physical robot in challenging outdoor conditions in addition to leveraging SMPC to account for model uncertainty. We use the term cost learning to be consistent with the MPC literature which considers a cost function rather than a reward function.

In this chapter, we present a practical algorithm to improve tracking performance for repetitive path following that leverages the combination of model learning (to efficiently make predictions of the cost associated with a sequence of control inputs) with cost learning (to account for systematic changes in the accuracy of these predictions). Intuitively, this is like how humans practice driving to understand how a car handles, but drive at a reasonable speed where we understand the cost/risk of our actions; our approach increases the cost of taking actions where the model under-estimates the cost of taking that action. Second, we show how this algorithm can be integrated seamlessly with a state-of-the-art path-following controller based on stochastic MPC which can account for model uncertainty. Finally, we demonstrate the proposed algorithm in both simulation and experiment and provide a thorough analysis of the effect of the proposed algorithm on several aspects of controller performance.

6.3 Problem Statement

We consider a ground robot performing a repetitive path following task using SMPC (see (2.8)–(2.11)) with a cost function of the form (2.8):

$$\ell_f(\bar{\mathbf{s}}_H) + \sum_{i=0}^{H-1} \ell(\bar{\mathbf{s}}_{k+i}, \bar{\mathbf{u}}_{k+i}), \tag{6.1}$$

which is included here for the convenience of the reader. We recall that $\ell(\bar{\mathbf{s}}, \bar{\mathbf{u}})$ is the non-negative scalar cost associated with applying control with mean $\bar{\mathbf{u}}$ in a state with mean $\bar{\mathbf{s}}$, $\ell_f(\bar{\mathbf{s}}_H)$ is the non-negative scalar cost of the mean of the final state in the SMPC prediction horizon of length H. In addition, we assume that the dynamics model used in SMPC has limited expressive power (and therefore predictive accuracy) due to limited prior knowledge or data about the operating conditions as well as computational constraints. Since the accuracy of the dynamics model varies over the length of the path,

the accuracy of predictions about the cost associated with a sequence of control inputs will also vary because the cost depends on future states, which are predicted using the dynamics model.

The goal of the method presented in this chapter is to enable the controller to automatically adapt the robot's behaviour to reduce the cost of traversing the path (calculated using the SMPC cost function) by accounting for errors in the cost predicted using the dynamics model.

6.4 Methodology

In this section, we present our approach and how it fits into the SMPC formulation presented in Chapter 4. Our approach is based on modelling the difference between the cost (2.8) over the prediction horizon predicted using the approximate model for robot dynamics (2.9) and the cost actually incurred over the horizon. By parametrizing this difference in terms of a variable that can be controlled, we enable the controller to automatically adjust vehicle behaviour to reduce the impact of model mismatch on minimizing the cost function (2.8) in closed loop. We also use a measure of similarity to previous traversals of the path to construct a local cost correction model using the most relevant data. In this thesis, we calculate cost using the mean states and inputs. Our approach would also apply if we used the expected value of the cost because uncertainty predictions are also model-based.

6.4.1 Cost Prediction Error

Let the sampling time k refer to each time a control is computed and time step i refer to a point along the prediction horizon at a given sampling time. At each sampling time k, we compute the optimal sequence $\{\mathbf{s}_{k+i+1}, \mathbf{u}_{k+i}\}_{i=0}^{H-1}$ and apply $\mathbf{u}_k$ to the robot. After repeating this for H sampling times, we have the actual states the robot visited and the controls that were applied over the initial horizon at time k. This gives us both the cost predicted at sampling time k for the i^{th} step in the horizon, $\ell_{k,i}$, and the actual cost incurred at sampling time $k+i$, $\ell_{k+i,0}$. We can then compute the cost prediction error $\delta\ell_{k,i}$ for sampling time k at timestep i:

$$\underbrace{\ell_{k+i,0}}_{\text{actual}} = \underbrace{\ell_{k,i}}_{\text{predicted}} + \underbrace{\delta\ell_{k,i}}_{\substack{\text{cost}\\ \text{prediction}\\ \text{error}}}. \tag{6.2}$$

We assume that the distribution of $\delta\ell$ depends on a quantity $\mathbf{a}$ which can be predicted using the model for robot dynamics and how far the timestep is along the horizon, i. This results in a dataset $\{\delta\ell_{k,i}, \mathbf{a}_{k,i}\}$ for each timestep $i = 1...H-1$ in the horizon at each sampling time $k = 0...N-H-1$.

In general, $\mathbf{a}$ should be related to a state of the vehicle that affects model accuracy and can be altered using the control inputs. For ground robots, factors that are hard to model often become more significant as speed increases which makes speed a natural candidate. This choice will be platform-dependent and require some expert knowledge to choose effectively. More variables can be included at the expense of data efficiency.

6.4.2 Data Management

Let a run be defined as a complete traversal of the path. Since we are considering a repetitive path-following task, we store data $\{\delta\ell_{k,i}, \mathbf{a}_{k,i}\}$ indexed by location along the path and run number. This automatically encodes factors such as local terrain properties (for ground vehicles) and local path geometry that influence the cost prediction error.

6.4.3 Model for Cost Prediction Error

The main properties of $\delta\ell_{k,i}(\cdot)$ that we wish to capture are: *(i)* the dependence on $\mathbf{a}$ which may be nonlinear, *(ii)* heteroscedasticity since the predictions of cost may be more or less precise depending on the value of $\mathbf{a}$, and *(iii)* a principled mechanism to weight data from previous runs according to a measure of similarity to the current run. One model that has these properties is the mixture of experts [Murphy, 2012]. Specifically, we will consider a mixture of experts based on Bayesian linear regression where each expert is fit to data from one run c. Predictions for $\delta\ell_{k,i}$ (dropping subscripts for compactness) are made using:

$$p(\delta\ell|\mathbf{a}) = \sum_c p(\delta\ell|\mathbf{a}, c)p(c|\mathbf{a}), \tag{6.3}$$

where each expert is:

$$p(\delta\ell|\mathbf{a}, c) = \mathbf{w}_c^T\mathbf{a} + \eta_c, \qquad \eta_c \sim \mathcal{N}(0, \sigma_c^2), \tag{6.4}$$

and the gating function is a mixture of Gaussians:

$$p(c|\mathbf{a}) \propto p(\mathbf{a}|c)p(c), \tag{6.5}$$

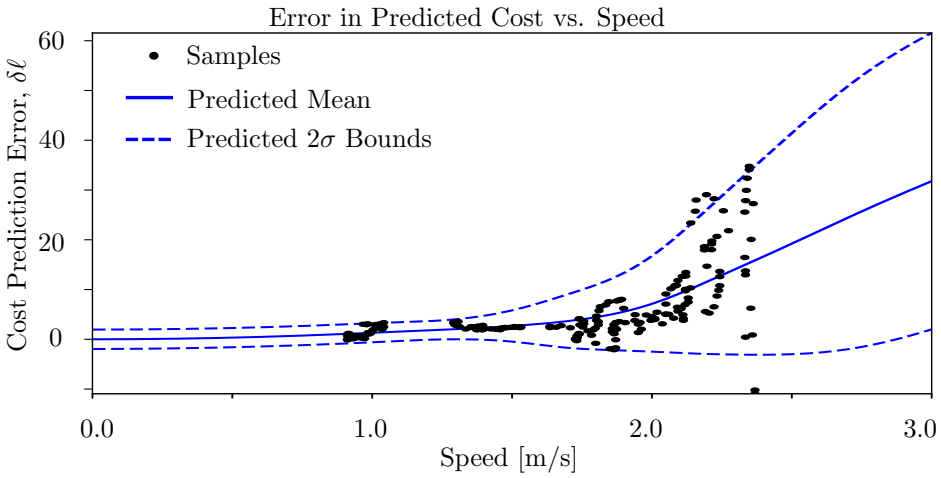


Figure 6.1: Example of $\delta\ell$ plotted as a function of forward speed taken from an experiment in snowy conditions. The black dots are samples, the solid blue line is the estimated mean from a model $\delta\ell(\cdot)$, and the dotted lines are upper and lower 2σ bounds. In this case, the variance of $\delta\ell$ changes substantially between 1.5 and 2.5 m/s which can be incoroporated into the cost function for MPC by augmenting the cost with an upper percentile (e.g., 2σ) of the cost rather than the mean to discourage actions that produce a high variance in cost prediction error.

where $p(\mathbf{a}|c)$ is a Gaussian describing the density of $\mathbf{a}$ for run c and $p(c)$ is the prior probability of observing data from the same distribution as run c. Parameters $\mathbf{w}_c$ and σ_c^2 are updated continuously and so depend on k. Property *(i)* can be achieved through the feature $\mathbf{a}$ (which can be a nonlinear transformation of the state) and the mixing weights, *(ii)* is achieved since each expert c can have a different variance, σ_c^2, and *(iii)* can be achieved through setting the prior $p(c)$ based on recent observations, which will be discussed in the next section. To account for the fact that cost prediction error varies over the prediction horizon (generally increasing further into the future as prediction error accumulates), we partition the horizon into multiple sections and fit a separate mixture model to each grouping of i.

6.4.4 Cost-based Mode Inference

We assume that $\delta\ell$ is dependent on $\mathbf{a}$ and i, which can be measured directly, and additional factors cannot be measured directly which we will call the mode. Let $\mathcal{D}_{\delta\ell}^- = \left\{\delta\ell_j^-, \mathbf{a}_j^-\right\}_{j=0}^n$ be the n most recent measurements of predicted cost error from the current run for a particular section of the horizon. We will follow McKinnon and Schoellig [2019a] and make the run prior $p(c)$ dependent on recent data so it becomes:

$$p(c|\mathcal{D}_{\delta\ell}^-) \propto med\left(p(\delta\ell_j^-|\mathbf{a}_j^-, c)\right) \qquad j = 0...n \tag{6.6}$$

Figure 6.2: The Clearpath Grizzly in **nominal** (left) and **loaded** configurations (right). The mass of the bags of gravel was 133 kg.

where $med(\cdot)$ is the median. Here, we have departed from the conventional assumption that data is i.i.d. (which would lead to $med(\cdot)$ being replaced by a product). Our main motivation was that this produced smoother estimates of $p(c|\mathcal{D}^{-}_{\delta\ell})$ in experiment.

6.4.5 Augmented MPC Cost Function

Now that we have a model for $\delta\ell(\cdot)$ we can augment the cost function used to compute the optimal control sequence in (2.8). This results in the following cost function for MPC:

$$\ell_f(\bar{\mathbf{s}}_H) + \sum_{i=0}^{H-1} \ell(\bar{\mathbf{s}}_{k+i}, \bar{\mathbf{u}}_{k+i}) + \delta\ell^u(\bar{\mathbf{a}}_{k+i}) \tag{6.7}$$

where $\delta\ell^u$ is an upper percentile of $p(\delta\ell|\mathbf{a})$. See Fig. 6.1 for a visualization of what the cost function may look like and how using different percentiles change the additional cost. Our main contribution in this paper is to learn $\delta\ell(\cdot)$ and demonstrate the effectiveness of our approach on a robot in changing conditions.

6.4.6 Model Learning

The model learning method we used in this chapter is based on using Bayesian linear regression to learn the unknown dynamics. Please see Chapter 4 for details.

6.5 Application to a Ground Robot

This section outlines how we apply our method to the Clearpath Grizzly shown in Fig. 6.2. See `http://tiny.cc/cost-model-learning` for a video of our experiments.

6.5.1 Robot Model

The model for robot dynamics is the one used in Chapter 4 given by (4.25) and (4.26).

6.5.2 Implementation

Our algorithm was implemented in C++ and the controller runs at 10 Hz with a three second look-ahead discretized by 30 points. The optimisation problem (2.8)-(2.11) is solved as a sequential quadratic program and re-linearized three times, taking an average of 45 ms. We use a quadratic penalty for $\ell(\cdot)$ where the weights on tangential, lateral, heading, speed, and turn rate error are 100, 200, 100, 2, and 2 respectively. The weights on deviation of commanded speed, turn rate, and reference speed from their references are 1, 1, and 40 respectively. The weights on rate of change of commands in the same order are 10, 15, and 5.

The proposed cost error model is updated at 5 Hz in a separate thread from the controller. This keeps the most computationally expensive step of the proposed approach, fitting the model, separate from the control loop. For all experiments, we use $\mathbf{a} = v^2$ because we expect similar cost prediction error driving forwards and backwards. We partition the horizon into two segments for all of our experiments.

6.6 Experiments

To demonstrate the effectiveness of the proposed algorithm in experiment, we tested our algorithm on a Clearpath Grizzly driving in various outdoor environments. For all figures showing a distribution of a quantity, we show the 25th, 50th, and 75th percentiles.

6.6.1 Cost Learning vs. Model Learning

In this experiment, we compare the effectiveness of model learning and cost learning separately to the combination of both. Cost learning alone means that the vehicle uses a fixed model for the robot dynamics that is fit using previous data but learns $\delta\ell(\cdot)$. Model learning alone means that the controller uses the online model learning algorithm presented in McKinnon and Schoellig [2019a] and $\delta\ell(\cdot) = 0$. The combination means that both the model and cost are learned online over the course of the run. For these experiments, we use the mean value of $\delta\ell(\cdot)$ to augment the MPC cost function (6.7). We drove the vehicle 6 times around a 73 m course (shown in Fig. 6.3). The two main sources of model error in this case are a bumpy section of the path, which causes significant but

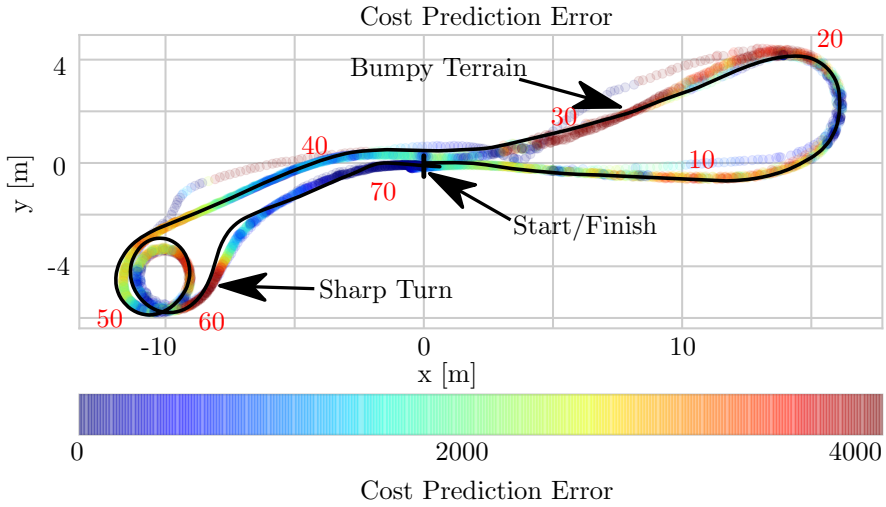
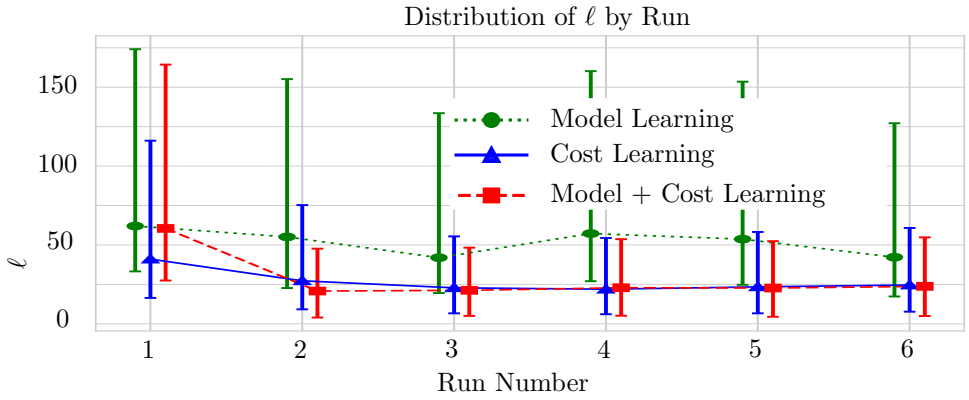


Figure 6.3: Sum of cost prediction error over the horizon at each sampling time superimposed for 6 runs. Distance along the path in meters is marked in red. For reference, the actual cost over the horizon ranges from 300 to 8000 depending on the section of the path. The cost prediction error is consistently large over a bumpy section of the path and around a sharp turn near the end of the path. The vehicle has limited suspension so driving over the bumpy section of the path produces significant disturbances that are hard to model.

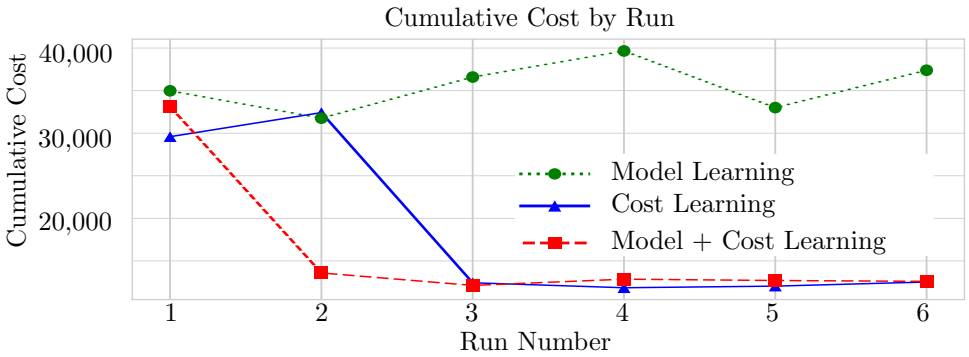
non-repeatable disturbances at high speed, and a sharp turn, which causes significant side-slip (not in the model) and skidding. The vehicle was in the nominal configuration (see Fig. 6.2) for these experiments.

To measure the effect of each component of the proposed algorithm, we first look at differences in the control cost along the path. In this experiment, the main effect of adding cost learning is for the vehicle to slow down over the section of the path with bumps and sharp turns. Fig. 6.4a shows that the distribution of ℓ over each run is significantly lower when using cost learning compared to model learning. Similarly, Fig. 6.4b shows a lower cumulative cost to traverse the path with cost learning compared to model learning. In both cases, the combination of cost learning and model learning convergences to low cost in the fewest runs. In this experiment, we used a good initial guess for the dynamics model–(4.25) and (4.26) with parameters identified using data from similar conditions–so cost learning alone achieved good results. For a different initial guess, the best performance could be arbitrarily bad.

To gain insight into why, we investigate the effect of cost learning on model accuracy. Intuitively, adding $\delta\ell(\cdot)$ should discourage the system from taking actions that result in high model error since this leads to high cost prediction error. We measure prediction accuracy of the mean states over the MPC horizon using M-RMSE and M-RMSZ. Ideally, M-RMSE would be low and M-RMSZ would be around one.



(a) The distribution of ℓ for each run. The combination of cost and model learning converges to the lowest ℓ in the fewest runs.



(b) Cumulative cost to traverse the path by run. The combination of cost learning and model achieves the lowest cost in the fewest run.

Figure 6.4: A comparison of the cumulative cost and ℓ with **cost learning**, **model learning** and **cost learning + model learning** on the course shown in Fig. 6.3. Cost learning reduces the cost to traverse the path, both in the average cost and cumulative cost, after just three runs. When combined with model learning, it converges after just one run. Model learning on its own is not sufficient to achieve the same performance making little improvement in the average cost and no improvement in the cumulative cost.

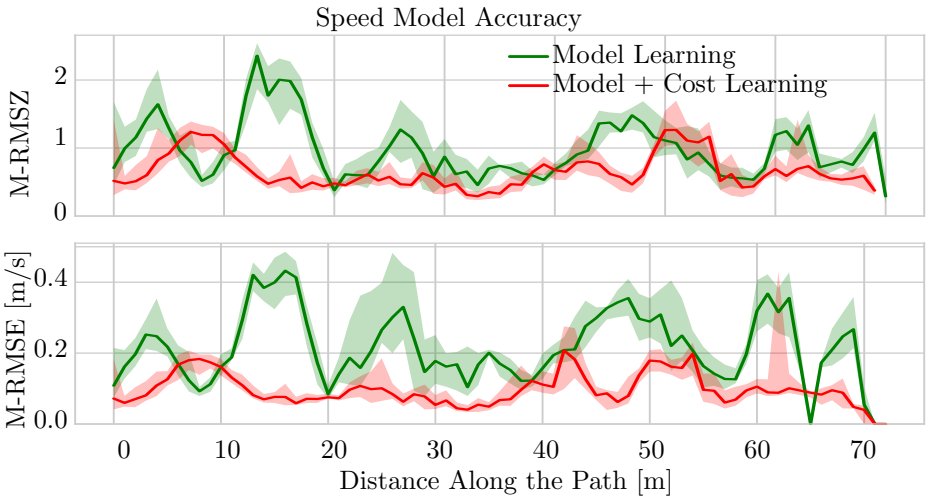


Figure 6.5: A comparison of the multi-step RMS Error (M-RMSE) and Z-score (M-RMSZ) when using only **model learning** and **cost learning + model learning** on the course shown in Fig. 6.3. The addition of cost learning improves prediction accuracy (lower M-RMSE) and uncertainty estimates (M-RMSZ closer to 1.0).

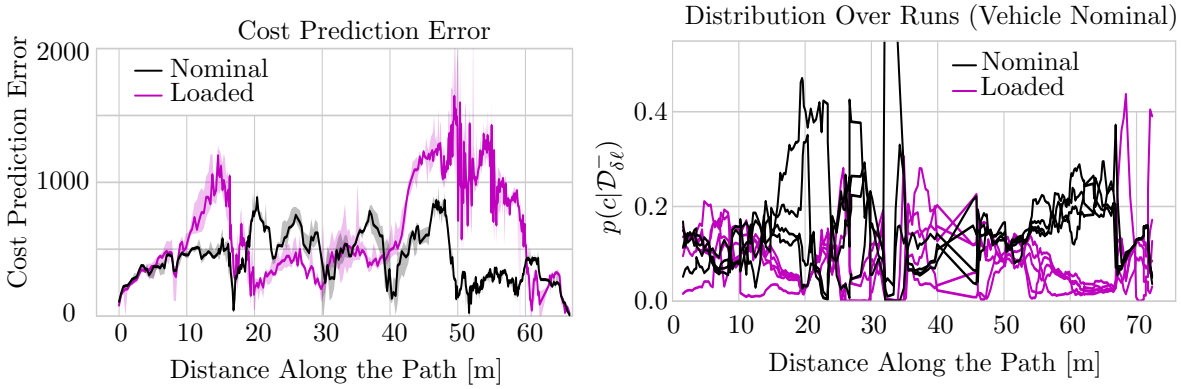
Figure 6.5 shows the M-RMSE and RMSZ for speed over the course of 6 runs with model learning vs. model and cost learning. The combination of cost and model learning reduces the average M-RMSE for speed and turn rate by 50% and 46% respectively and reduces the number of times that the M-RMSZ jumps significantly above one.

While using a richer model may improve prediction accuracy, cost correction provides a means of adapting the vehicle's behaviour in cases where the dynamics model accuracy varies along the path which is useful for driving in challenging off-road conditions.

6.6.2 Cost-based Mode Inference

In the previous section, we showed that cost learning reduces the cost of traversing a path when the dynamics were the same between runs. For the experiments in this section, we changed the robot dynamics between runs by adding or removing a payload of gravel bags (see Fig. 6.2). The path was similar to the one shown in Fig. 6.3. For these experiments, we used a fixed dynamics model and only vary whether or not mode inference (Sec 6.4.4) is enabled to isolate the effect of this component of the algorithm. When mode inference is not enabled, $p(c|\mathcal{D}^{-}_{\delta\ell})$ is uniform.

First, we investigate whether the mode inference recommends runs where the vehicle was in a similar configuration. Figure 6.6a shows the predicted cost error for 15 traversals of the path coloured by vehicle configuration. Here, we see that adding payload increased $\delta\ell$ between 40 and 60 m when the vehicle was turning sharply. The proposed algorithm



(a) Distribution of the sum of $\delta\ell$ over the prediction horizon at each sampling time along the path for 15 runs coloured by vehicle configuration.

(b) Estimated $p(c|\mathcal{D}^-_{\delta\ell})$ for each previous run when the vehicle was in the **nominal** configuration, during run 10. Each line is coloured by the actual vehicle configuration during that previous run.

Figure 6.6: The vehicle was driven for 15 runs alternating between the **loaded** and **nominal** configuration. The sum of cost prediction error along the horizon at each sampling time, depicted in *(a)*, shows that the cost prediction error differs significantly between 40 and 60 m along the path, which corresponds to a sharp turn. Over this section of the path, *(b)* shows that during run 10, when the vehicle was in the nominal configuration, mode inference correctly identifies that runs in the nominal configuration are more similar since those runs are assigned a higher $p(c|\mathcal{D}^-_{\delta\ell})$.

correctly infers that $\delta\ell$ is the most similar to runs when the vehicle is in the same configuration as the live run, especially between 45 and 65 m (e.g., see Fig. 6.6b for the estimated probability of each run during run 10 when the vehicle was in the nominal configuration). There is a slight delay because the algorithm uses a sliding window of previous experiences to estimate $p(c|\mathcal{D}^-_{\delta\ell})$. Finally, Fig. 6.7 shows that the algorithm consistently selects the majority of experiences from a run where the vehicle was in a similar configuration with the exception of run 6, which is the first run in the nominal configuration. Runs in a different configuration from the live run receive high $p(c|\mathcal{D}^-_{\delta\ell})$ some fraction of the time, however Fig. 6.6a and 6.6b show that this is over sections of the path when $\delta\ell$ is similar between configurations so this would have little impact on the model $\delta\ell(\cdot)$.

Second, we show that including mode inference makes a difference to the control performance over the section of the path where the cost prediction error is different between the two configurations (between 40 and 60 m along the path). Figure 6.8a shows that the distribution of ℓ incurred over the path is lower when mode inference is enabled after the algorithm has one run in each configuration. Figure 6.8b shows that the cumulative cost to traverse the path from 40-60 m is also reduced when mode inference

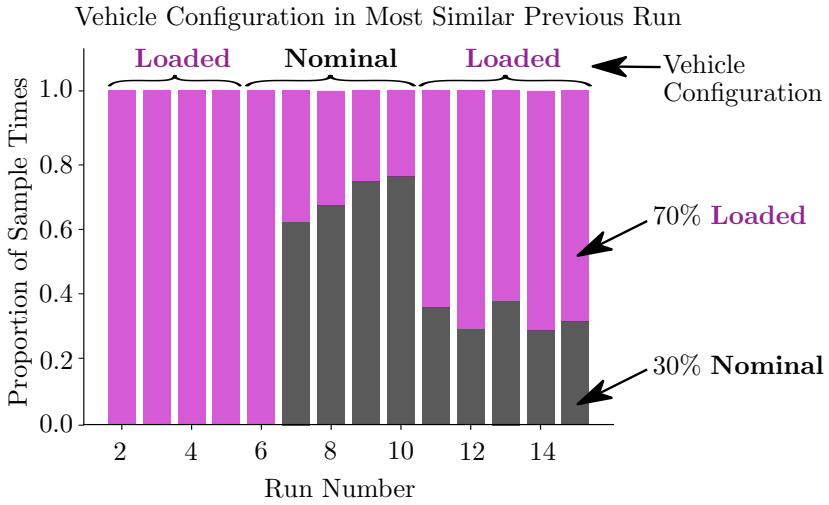
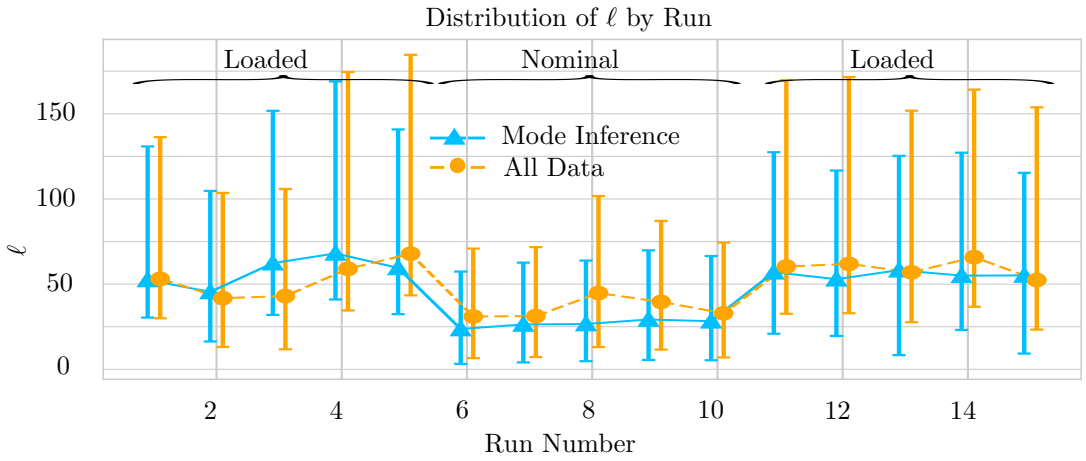


Figure 6.7: This figure shows the fraction of the time the previous run with the maximum $p(c|\mathcal{D}^-_{\delta\ell})$ was from each configuration for each run. This figure shows that the run with the highest $p(c|\mathcal{D}^-_{\delta\ell})$ consistently came from a run with the vehicle in the same configuration as the live run with the exception of run 6 when the vehicle has no previous experience in the nominal configuration. Mixing experience from different configurations is acceptable when the cost prediction error is similar for both configurations since the resulting model $\delta\ell(\cdot)$ will therefore also be similar.

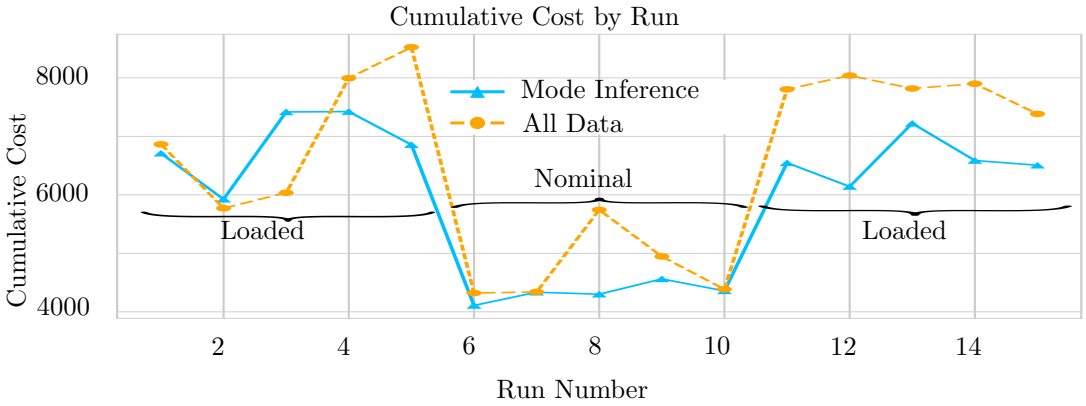
is enabled after the vehicle has completed one run in each configuration.

6.6.3 State Dependent Measurement Noise in Simulation

In addition to model error, measurement noise can also contribute to cost prediction error since the measured state provides the initial condition for rolling out the sequence of control inputs over the prediction horizon. Figure 6.9 shows how the proposed algorithm reduces the cost to traverse the path when we added zero-mean Gaussian noise with variance proportional to v^2 over a straight section of the path. The model for robot dynamics was otherwise perfect. We used the upper 2σ bound on the cost prediction error to augment the cost function to account for the random nature of the cost induced by the noise. The algorithm learns that there is a large additional cost associated with driving quickly over this section of the path so reduces the speed (Fig. 6.9b) and, consequently, the cost (Fig. 6.9a).

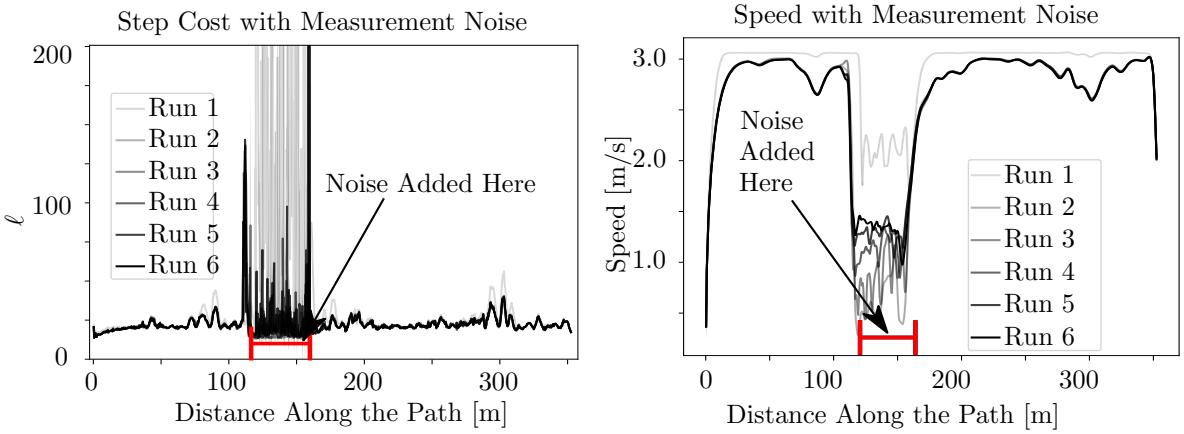


(a) The distribution of ℓ incurred over each traverse of the path with and without mode inference enabled. After run 6, when the vehicle has experience from both configurations, adding mode inference consistently lowers the step cost throughout the run.



(b) The cumulative cost for the vehicle to go from 40-60 m. Including mode inference for learning the cost prediction error consistently improves performance after the algorithm has experience in each configuration.

Figure 6.8: A comparison of the distribution of ℓ and the cumulative cost to traverse the path shown with **mode inference** compared to using **all previous data** to construct $\delta\ell(\cdot)$. Adding experience recommendation consistently reduces the cost after run 6, when the vehicle has experience from both configurations.



(a) The cost ℓ along the path for each run when cost learning is enabled.

(b) The vehicle's forward speed with each run when cost learning is enabled.

Figure 6.9: Results of enabling the cost learning in simulation with speed dependent noise. The system learns to drive slowly over the section of path with speed dependent noise to reduce the associated cost.

6.7 Discussion

In the previous section, we demonstrated that the proposed algorithm is able to reduce the cumulative cost to traverse the path. Throughout this paper, we have only considered the cost over the *prediction horizon* at each sampling time which does not guarantee consistent reduction in the cumulative cost to traverse the path. Approaches such as Rosolia and Borrelli [2017] and Rosolia et al. [2017a] rely on a model to make predictions over the MPC horizon, however they do consider the optimal cost-to-go from the final state in the prediction horizon to the end of the path. Combining ideas from both may help to further improve the performance of MPC-based algorithms in challenging environments.

With regards to parametrising the cost prediction error, the representation we chose for $\delta\ell(\cdot)$ could be non-convex for more than one expert in the mixture model (6.3). While it is difficult to discern whether or not the behaviour observed in experiment is the result of SMPC finding a sub-optimal, local minima, we found that the proposed algorithm converged to consistent behaviour in our experiments. In a further iteration of the work in this chapter, it may be worth exploring the implications of non-convexity in the cost prediction error and how best to deal with it. In addition, the computational complexity of our approach is linear in the number of runs for both fitting the model and evaluating it in the control loop. On average (and 99th percentile), it takes 0.4 ms (1.0 ms) per previous run in memory to fit the model and 0.32 ms (0.44 ms) per previous run in

memory to evaluate (including the time to calculate all relevant quantities in the three iterations of the SQP solved in SMPC). If the number of runs becomes large, it may become economical to fix the number of experts and use a procedure like expectation maximization to limit the time required to evaluate the cost correction term.

It is reasonable to ask whether it is appropriate to add complexity in terms of cost learning as we have proposed or to concentrate on improving the dynamics model. Model learning is effective when the form of the dynamics is known or there is sufficient data to train an expressive model. There is a large body of work demonstrating various ways to do so. Cost learning is effective when the dependence of model accuracy is known but the form of the dynamics is not, when data is scarce, or when other factors separate from the dynamics model such as the localisation system affect closed-loop performance. By combining both, we can improve performance when form of the dynamics is known (model learning) and avoid states that induce higher cost than predicted (cost learning).

Another point to consider in this trade-off is whether or not the factor that causes varied model prediction accuracy is localised to a relatively short section of the path. If so, improving the dynamics model to accurately capture these dynamics may only have a small impact on the overall task performance. In this case, using cost learning may be a simple solution to reduce the impact of this factor while maintaining similar performance everywhere else. If the dynamics model has poor predictive performance over large sections of the path, improving it may have a larger impact on tracking performance and be a good investment of time. What constitutes good and poor predictive performance will be task specific.

6.8 Summary and Contributions

The novel contribution of this chapter was to present an algorithm for correcting the cost predicted using a dynamics model *over the prediction horizon* in SMPC for a repetitive path-following task. We demonstrated our algorithm in simulation and experiment in combination with model learning and showed that the combination of cost and model learning out-performed either component separately in terms of the control cost to traverse the path. In addition, using cost learning improved the effective accuracy of the dynamics model by encouraging the controller to execute motions where the dynamics model accurately predicted the cost. We encourage the reader to watch our video at http://tiny.cc/cost-model-learning showing the experiments conducted in this chapter.

Chapter 7

Summary and Future Work

In this chapter, we provide a summary of the novel contributions presented in this thesis and the publications that arose from them. We also discuss future work to further improve model learning approaches for SMPC and experiments that might give us further insight into the performance of the proposed algorithms.

7.1 Summary of Contributions and Publications

The motivation behind this thesis stemmed from the desire to enable mobile robots to navigate safely in changing operating conditions without requiring extensive prior knowledge about the robot's dynamics in each operating condition. Path following control has the potential to enable many exciting industrial applications by automating the movement of vehicles to improve their efficiency and predictability or by keeping human operators out of harm's way. The objective of this thesis was to extend existing model learning methods for SMPC beyond the case when the dynamics were unknown but could be modelled by a single underlying function. Throughout this process, we have gained insights into the challenges in autonomous path following control in changing conditions which have influenced our approach to this problem. This has resulted in several publications to communicate our findings to the scientific community which we hope others find useful in their continued pursuit in this area.

In our early work, presented in Chapter 3 we presented two methods to extend GP-based methods to model multi-modal dynamics without requiring that the dynamics in each mode or the number of modes were known ahead of time. These methods have appeared in the proceedings of two full-paper refereed conferences:

- C. McKinnon and A. P. Schoellig. Learning Multi-Modal Models for Robot Dy-

namics with a Mixture of Gaussian Process Experts. *In Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 322–328, 2017.

- C. McKinnon and A.P. Schoellig. Experience-Based Model Selection to Enable Long-Term, Safe Control for Repetitive Tasks Under Changing Conditions. *In Proc. of the Intl. Conference on Intelligent Robots and Systems (IROS)*, pages 2977–2984, 2018.

In summary, the novel contributions of Chapter 3 are:

- A multi-modal learning framework capable of learning an apriori unspecified number of nonlinear dynamics models from a continuous, live stream of data.
- A thorough analysis of the predictive accuracy of this method and results demonstrating that it is capable of recovering the predictive performance of GP models trained using known data association.
- An experience recommendation technique for fitting local GPs to model robot dynamics in an apriori unspecified number of modes that is computationally feasible in closed loop with SMPC.
- Closed-loop experiments demonstrating that the proposed experience-recommendation method improves over an existing GP-based model learning method in the case of changing dynamics.

After this initial work, we focused on addressing the one-size-fits-all assumption that GP hyper-parameters could be identified ahead of time and used to model the robot dynamics in all conditions. Rather than increasing the complexity of the model and trying to capture all the subtleties in the robot dynamics, we took the approach that any model will perform better in some conditions than in others, and that it was therefore important to be able to adjust the model's aleatoric uncertainty–the uncertainty that cannot be reduced by gathering more data–using data from the robot's current operating conditions online. To do so, we developed a novel approach based on Bayesian Linear Regression. Our approach can leverage data from multiple traversals of a path, in keeping with the results from the previous chapter, in addition to leveraging data from the current traversal of the path to adapt to new conditions continuously. This is in contrast to the previous chapter, where only data from previous traversals was used to model the robot dynamics. We were able to achieve comparable or better prediction accuracy to GPs using basis functions that required minimal prior knowledge about the robot dynamics.

In addition, we showed through experiments that this translated into better closed-loop performance.

These results appeared in the proceedings of a full-paper refereed conference and journal:

- C. McKinnon and A. P. Schoellig. Learn Fast, Forget Slow: Safe Predictive Learning Control for Systems with Unknown and Changing Dynamics Performing Repetitive Tasks. *Robotics and Automation Letters*, 4(2):2180–2187, 2019a.
  - See http://tiny.cc/fast-slow-learn for the associated video.
- C. McKinnon and A. P. Schoellig. Learning Probabilistic Models for Safe Predictive Control in Unknown Environments. In *Proc. of the European. Conf. Control (ECC)*, pages 2472–2479, 2019b.

In summary, the novel contributions of Chapter 4 are:

- A novel model learning method based on Bayesian linear regression capable of adapting to novel dynamics continuously while simultaneously leveraging data from previous traversals in a repetitive path-following task.
- A thorough experimental analysis of the predictive performance of this model on a ground robot with changing dynamics in comparison to a GP-based method.
- An experimental analysis of the closed-loop performance of the proposed algorithm to highlight the advantage of continuous adaptation and leveraging past data.
- A closed-loop comparison of the proposed method to a GP-based method to demonstrate improved tracking performance.

In Chapter 5, we focused on enabling the BLR-based method from the previous chapter to leverage prior datasets to learn basis functions from data. Specifically, we presented a novel approach to learn the forward and inverse map for a function relating the control input to the time derivative of a particular state of the robot. The output of this function was called the input feature. Our approach adapted to changes in the robot dynamics by using BLR to model the dynamics of the robot in new conditions as a linear combination of the input feature with other manually specified basis functions. These manually specified basis functions can be used to include prior knowledge about possible changes in the robot dynamics.

The results of this work have been accepted for publication in a full-paper refereed journal:

- C. McKinnon and A. P. Schoellig. Meta Learning with Paired Forward and Inverse Models for Efficient Receding Horizon Control. *Robotics and Automation Letters*, 6(2):3240-3247, 2021.

In summary, the novel contributions of Chapter 5 are:

- The first approach using the combination of forward and inverse models with online BLR to enable computationally efficient meta-learning with SMPC.
- A method to automatically select the best input feature from from a set of input features provided apriori.
- An approach for designing the cost function in SMPC to maintain consistent performance when changing the input feature without re-tuning the cost function parameters.

Finally, in Chapter 6, we explored alternative methods to learning the dynamics model to improve the closed-loop performance of SMPC. In this chapter, our assumption is that all methods used to model the robot dynamics are limited in their representational power due to the computational constraints imposed by SMPC. In addition, the dynamics model is not the only thing that determines closed-loop performance; the localisation system also plays a critical role. To address this, we proposed a novel method for correcting the cost predicted using the dynamics model *over the prediction horizon* to account for factors not included in the dynamics model.

This method was presented in the proceedings of a full-paper refereed conference:

- C. McKinnon and A. P. Schoellig. Context-aware Cost Shaping to Reduce the Impact of Model Error in Receding Horizon Control. In *Proc. of the Intl. Conf. on Intelligent Robotics and Automation (ICRA)*, pages 2386–2392, 2020.
  - See `http://tiny.cc/cost-model-learning` for the associated video.

In summary, the novel contributions of 6 are:

- A novel method for using past experience to correct cost-prediction errors over the SMPC prediction horizon to account for factors not included in the dynamics model.

- The extension of this method to the case when un-modelled factors affecting cost prediction error depend on a discrete, latent variable.

- Experimental results demonstrating the effectiveness of this approach on the Clearpath Grizzly in closed loop for changes in its dynamics.

- Simulation results demonstrating the potential of this approach to improve performance in the presence of speed dependent measurement noise in the localisation system even when the dynamics model is perfect.

7.2 Future Work

In this section, we discuss potential work to further improve the performance of model learning methods for SMPC in changing conditions.

All of our methods were developed in the context of the receding horizon controller outlined in the introduction and detailed in the remaining chapters. This formulation was chosen because it was the basis for the initial control implementation we built upon Ostafew et al. [2016] and proved sufficient for demonstrating improvements in the dynamics model, which was the focus of this thesis. While it was sufficient for this purpose, we neglected important factors such as recursive feasibility and the value function. While not necessarily novel contributions (both have been studied in Koller et al. [2018], Rosolia et al. [2017a], these would improve the theoretical soundness of the approach and could lead to performance improvements.

In addition, the controller implementation used a single dynamics model with additive, Gaussian uncertainty. This relies on our model selection algorithms to choose the appropriate dynamics model from many. A key assumption that we made was that the dynamics over the previous section of the path could be used to infer the dynamics over the upcoming section of the path. This was reasonable for our applications because the dynamics changed in a repeatable way over the course of each run. Scenario MPC is a branch of MPC that is capable of leveraging multiple dynamics models simultaneously. This would allow us explore cases when the dynamics model could switch between a limited number of modes *during* a traversal of the path. As an alternative to relying solely on the dynamics over the previous section of the path, it would also be interesting to explore the use of a sensor such as a camera to provide additional information about the robot dynamics over the upcoming section of the path. Recent results from computer vision including semantic segmentation could be investigated for this purpose.

In Chapter 3, we investigated methods using GP regression to model the robot dynamics. A key assumption that we made was the dynamics could be represented with a fixed set of hyper-parameters. We investigated the possibility of optimising the hyper-parameters online using a short window of data, however this resulted in inconsistent performance that was often worse than what could be achieved using our BLR-based approaches. Since this could be a symptom of overfitting, it would be interesting to investigate the use of a prior for the GP hyper-parameters [Murphy, 2012], or consider optimising fewer hyper-parameters, such as only optimising the noise variance and leaving the length-scales fixed.

In Chapter 4, we investigated model learning using Bayesian linear regression. This allowed us to leverage prior knowledge about the robot dynamics to reliably improve on our GP-based methods in terms of both computational efficiency and model accuracy. An important factor for the BLR-based methods was the strength of the prior, which smooths out the estimated model parameters and thus the controls applied to the vehicle. However, it also limited the rate at which the model parameters could change to reflect changes in the robot dynamics. The need to smooth the model parameters was driven, in part, by the need to be robust to noise in the data used to fit the model. In the case of our models for the speed dynamics in Chapter 4, which predict acceleration, the acceleration was calculated by numerically differentiating speed which introduced significant noise that had to be removed using a sliding window median filter. As suggested by Ioinnis Paschadilis at the European Controls Conference following our presentation, it would be interesting to explore robust regression models to avoid ad-hoc filtering and perhaps allow faster adaptation to changes in the robot dynamics.

In Chapter 5, we introduced a meta-learning approach for SMPC to model the robot dynamics that leveraged inverse models to maintain computational efficiency. One major limitation of our approach is that it only extracted a single basis function from prior data and required that this was the basis function that related the control input to the robot dynamics. Other meta-learning methods such as Harrison et al. [2018] learn multiple bases functions from data and train the model specifically so that changes in the robot dynamics can be captured by a method such as Bayesian linear regression. The main challenge in taking an approach like this would be converting constraints on the state and control input into constraints on the values of these basis functions, if they were used as decision variables in SMPC. However, if possible, such a method could significantly increase the representative power of models that are computationally feasible in SMPC

Finally, in Chapter 6, we presented a method for learning the difference between the cost predicted using the dynamics model and the closed-loop cost incurred by the robot

driving over that section of the path. In simulation, we were able to show that this method could reduce the cumulative and step cost incurred when the vehicle traversed a path in the presence of speed-dependent noise. During later experiments conducted in the UTIAS Mars Dome, we noticed that the localisation system would momentarily fail when the robot went over a bump at speeds approaching 2 m/s. Bumps are particularly prevalent in the sandy environment when the vehicle Grizzly crossed its own tracks, since the heavy vehicle quickly produced ruts in the soft sand. This did not happen outdoors and is likely due to the dim lighting in the Mars Dome environment since we found that adding artificial light reduced the frequency of this problem in our experiments. This is a real-life example of speed-dependent noise in localisation, so it would be interesting to try the algorithms developed in Chapter 7 on this problem. The main requirement, however, is that the failures are momentary, since the approach presented in Chapter 7 will not help in the case of total localisation failure.

Appendix A

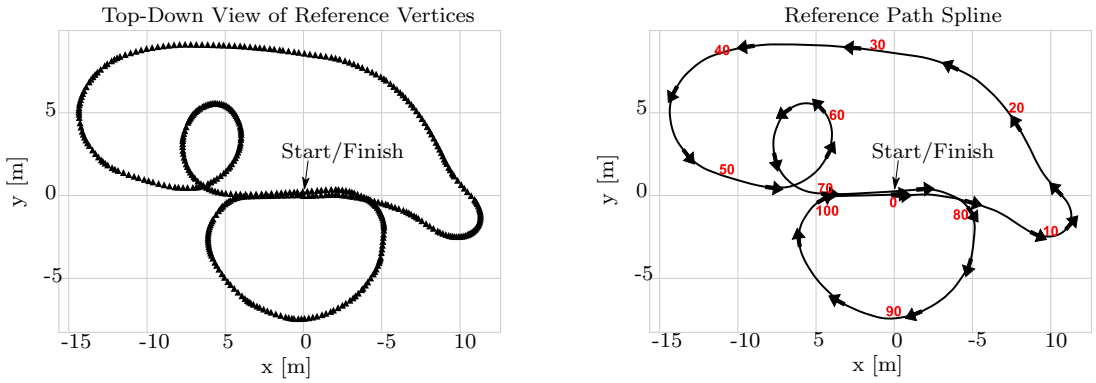
Path Parametrisation

All of the closed-loop experiments in this thesis were conducted using Visual Teach and Repeat 2.0 (VTR2) for localisation Paton et al. [2017]. VTR2 defines the goal for the controller as a sequence of waypoints (also known as vertices) to be traversed in order. The path is considered complete when the vehicle is within a certain distance of the end of the path. The purpose of this section is to describe the path parametrization used for converting this sequence of waypoints into a continuous path that can be used by a controller.

A.1 Path Parametrization

The desired properties for the path are that it: *(i)* have continuous first derivatives, *(ii)* go through all vertices in the path, and *(iii)* work for paths that contain turns-on-the-spot and direction-switches without modification. The path should have continuous first derivatives because the reference for the vehicles speed and turn rate will be derived from its first derivatives and these should be smooth to encourage smooth controller performance.

The parametrization used for the path was a cubic spline. Figure A.1 shows an example of the path fit to a sequence of vertices from Visual Teach and Repeat 2.0. Algorithm 2 details the procedure for fitting this path. This algorithm was used to produce a continuous reference that can be queried for x, y, and θ and is parametrised by an arc length parameter. This algorithm takes as input the sequence of reference vertices $\mathcal{V}^r$ and their relative transforms.



(a) Example of the 2D projection of the vertices specifying a 105 m path. Each vertex is indicated by a black triangle.

(b) The resulting reference path with heading indicated by the black arrows. The value of d shown in red corresponds roughly to path length because there are no turns-on-the-spot in this example.

Figure A.1: The reference path is specified by vtr 2.0 Paton et al. [2017] as a sequence of vertices which represent a pose in SE(3). The projection of these poses into SE(2) is used to construct a smooth reference path for the controller.

Algorithm 2 Procedure for Fitting the Path.

$\mathcal{V}^r \leftarrow$ Vertex IDs that define the reference path
$\left\{T_k^{k-1}\right\}_{k\in\mathcal{V}^r} \leftarrow$ Vertex-to-vertex transforms for the reference path.
$\left\{d_k^{xy}, d_k^{\theta}\right\}_{k\in\mathcal{V}^r} \leftarrow$ Euclidean distance, change in heading to the previous vertex.
$\{T_k^0\}_{k\in\mathcal{V}^r} \leftarrow$ Compound transforms to get transforms from each vertex to the origin
$\{x_k^r, y_k^r, \theta_k^r\}_{k\in\mathcal{V}^r} \leftarrow$ Project reference path to x, y, heading for the unicycle. Do not wrap angles!
$d_0^{ref} = 0$
$\hat{d}^{xy} \leftarrow$ unit Eucliden distance (e.g., 1 m)
$\hat{d}^{\theta} \leftarrow$ equivalent unit angle (e.g., 1 rad)
for $k \in \mathcal{V}^r / \mathcal{V}_0^r$ **do**
$\quad d_k^{ref} \leftarrow d_{k-1}^{ref} + \max\left(\frac{d_k^{xy}}{\hat{d}^{xy}}, \frac{d_k^{\theta}}{\hat{d}^{\theta}}\right)$
end for
$s_x^r(d), s_y^r(d), s_\theta^r(d) \leftarrow$ fit a spline to $\{x_k^r, y_k^r, \theta_k^r\}_{k\in\mathcal{V}^r}$ with $\{d_k^r\}_{k\in\mathcal{V}^r}$ as the independent variable.
$s_x^{r,\prime}(d), s_y^{r,\prime}(d), s_\theta^{r,\prime}(d) \leftarrow$ derivatives of $s_*^r(d)$ with respect to d

Appendix B

Exponential Family Distributions

This section borrows heavily from section 9.2.5 of Murphy [2012] and is intended to aid the reader in understanding the data weighting approach presented in section 4.3. For this purpose, it is helpful to review the exponential family of probability distributions and their canonical form.

As an example, consider the pdf of a univariate Gaussian in its most common form:

$$p(x|\mu,\sigma^2) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp(-\frac{1}{2\sigma^2}(x-\mu)^2). \tag{B.1}$$

This can be re-written in exponential family form as:

$$p(x|\mu,\sigma^2) = \underbrace{(2\pi\sigma^2)^{-\frac{1}{2}} \exp(-\frac{\mu^2}{2\sigma^2})}_{a(\boldsymbol{\lambda})} \exp\Big(\underbrace{\begin{bmatrix} \frac{\mu}{\sigma^2} & -\frac{1}{2\sigma^2} \end{bmatrix}}_{\boldsymbol{\lambda}^T} \underbrace{\begin{bmatrix} x \\ x^2 \end{bmatrix}}_{\mathbf{q}} \Big). \tag{B.2}$$

from which we can identify the canonical parameters as $\boldsymbol{\lambda} = \begin{bmatrix} \frac{\mu}{\sigma^2} & -\frac{1}{2\sigma^2} \end{bmatrix}$ and the sufficient statistics as $\mathbf{q} = [x, x^2]^T$, where $a(\boldsymbol{\lambda})$ is the reciprocal of the partition function. This can be re-written in canonical form as:

$$p(x|\mu,\sigma^2) = \exp\Bigg(\underbrace{\begin{bmatrix} \frac{\mu}{\sigma^2} & -\frac{1}{2\sigma^2} \end{bmatrix}}_{\lambda} \underbrace{\begin{bmatrix} x \\ x^2 \end{bmatrix}}_{\mathbf{q}} - \underbrace{\Big(\frac{\mu}{2\sigma^2} + \frac{1}{2}\log(2\pi) + \frac{1}{2}\log(\sigma^2) \Big)}_{A(\boldsymbol{\lambda})} \Bigg), \tag{B.3}$$

where $A(\boldsymbol{\lambda})$ is called the log-partition function. In what follows, we will review Bayes for the exponential family in its canonical form, which applies to any distribution in the exponential family (e.g., the multi-variate Gaussian and Inverse Gamma distributions used in Sec. 4.3).

B.1 Likelihood

If we consider a dataset $\mathcal{D}$ of N independent samples of a random variable $\mathbf{x}$, the likelihood of the exponential family is given by:

$$p(\mathcal{D}|\boldsymbol{\lambda}) \propto a(\boldsymbol{\lambda})^N \exp(\boldsymbol{\lambda}^T \mathbf{q}_N), \tag{B.4}$$

where $\mathbf{q}_N = \sum_{i=1}^N \mathbf{q}(\mathbf{x}_i)$ and $\mathbf{q}(\mathbf{x}_i)$ is the function that returns a vector of sufficient statistics for a sample $\mathbf{x}_i$. In terms of the canonical parameters, this becomes

$$p(\mathcal{D}|\boldsymbol{\lambda}) \propto \exp\left(N\boldsymbol{\lambda}^T \bar{\mathbf{q}} - NA(\boldsymbol{\lambda})\right), \tag{B.5}$$

where $\bar{\mathbf{q}} = \frac{1}{N}\mathbf{q}_N$.

B.2 Prior

The natural conjugate prior has the form:

$$p(\boldsymbol{\lambda}|\nu_0, \boldsymbol{\tau}_0) \propto a(\boldsymbol{\lambda})^{\nu_0} \exp(\boldsymbol{\lambda}^T \boldsymbol{\tau}_0), \tag{B.6}$$

where ν_0 is the size of the prior pseudo-data and $\boldsymbol{\tau}_0$ is the sum of the sufficient statistics for the prior pseudo-data. As before, this can be written in canonical form:

$$p(\boldsymbol{\eta}|\nu_0, \bar{\boldsymbol{\tau}}_0) \propto \exp\left(\nu_0 \boldsymbol{\lambda}^T \bar{\boldsymbol{\tau}}_0 - \nu_0 A(\boldsymbol{\lambda})\right). \tag{B.7}$$

B.3 Posterior

The posterior is the product of the likelihood and prior:

$$p(\boldsymbol{\lambda}|\mathcal{D}) \propto a(\boldsymbol{\lambda})^{N+\nu_0} \exp\left(\boldsymbol{\lambda}^T(\mathbf{q}_N + \boldsymbol{\tau}_0)\right). \tag{B.8}$$

In canonical form, this becomes:

$$p(\boldsymbol{\lambda}|\mathcal{D}) \propto \exp\left(\boldsymbol{\lambda}^T(\nu_0\bar{\boldsymbol{\tau}}_0 + N\bar{\mathbf{q}}) - (\nu + N)A(\boldsymbol{\lambda})\right), \tag{B.9}$$

where we can see that the sufficient statistics of the posterior are the sum of the average sufficient statistics for the prior and likelihood weighted by the number of samples in each dataset (or pseudo-dataset in the case of the prior) respectively.

B.4 Relation to Data Weighting

The data weighting presented in section 4.3 is to raise each term in the likelihood to the power of a weight $l \in [0, 1]$. If we consider a dataset of N points weighted by a constant weight l this is:

$$p(\mathcal{D}|\boldsymbol{\lambda}, l) \propto \exp\left(N\boldsymbol{\lambda}^T\bar{\mathbf{q}} - NA(\boldsymbol{\lambda})\right)^l, \tag{B.10}$$

$$\propto \exp\left(lN\boldsymbol{\lambda}^T\bar{\mathbf{q}} - lNA(\boldsymbol{\lambda})\right) \tag{B.11}$$

$$\propto \exp\left(\tilde{N}\boldsymbol{\lambda}^T\bar{\mathbf{q}} - \tilde{N}A(\boldsymbol{\lambda})\right), \tag{B.12}$$

which can be viewed as modifying the effective number of samples in a dataset from N to $\tilde{N} = lN$.

In the context of repetitive path-following, this provides a means to adjust the contribution of data from previous runs depending on how similar the dynamics were in those runs to the current run. The data weight should be $\in [0, 1]$ because each data point can be, at most, either completely omitted or fully included. A weighting below zero would be introducing a negative number of data points and going above one would be inflating the effective size of the dataset.

Re-weighting is important to keep the model flexible and able to continually adapt to new changes. If we did not continually re-weight the prior to keep ν_0 constant, then the effective number of data points in the prior ν_0 would grow by N at each sampling time. In a streaming setting, N is usually close to one. Eventually, $\nu_0 \gg N$ and new data would influence the parameters of the distribution less slowing adaptation to change. Re-weighting the prior keeps ν_0 constant, resolving this issue.

Bibliography

K. Mohta, K. Sun, S. Liu, M. Watterson, B. Pfrommer, J. Svacha, Y. Mulgaonkar, C. Taylor, and V. Kumar. Experiments in Fast, Autonomous, GPS-Denied Quadrotor Flight. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 7832–7839, 2018.

M. Paton, F. Pomerleau, K. MacTavish, C. Ostafew, and T. Barfoot. Expanding the Limits of Vision-based Localization for Long-term Route-Following Autonomy. *Journal of Field Robotics (JFR)*, 34(1):98–122, 2017.

J. Liang, N. Homayounfar, W. Ma, S. Wang, and R. Urtasun. Convolutional Recurrent Network for Road Boundary Extraction. In *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 9512–9521, 2019.

R. Hoover, T. Van Buskirk, and W. Garrott. Class 8 Mack Straight Truck Emulating a Refuse Hauler–Braking Improvement Study. Technical report, Washington, DC: National Highway Traffic Safety Administration, 2013. Report No. DOT HS 811 747.

J. Kober, J. Bagnell, and J. Peters. Reinforcement Learning in Robotics: A Survey. *Intl. Journal of Robotics Research (IJRR)*, 32(11):1238–1274, 2013a.

F. Berkenkamp and A. P. Schoellig. Safe and Robust Learning Control with Gaussian Processes. In *Proc. of the European Control Conf. (ECC)*, pages 2501–2506, 2015.

T. Moldovan, S. Levine, M. Jordan, and P. Abbeel. Optimism-Driven Exploration for Nonlinear Systems. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 3239–3246, 2015.

C. Xie, S. Patil, T. Moldovan, S. Levine, and P. Abbeel. Model-based Reinforcement Learning with Parametrized Physical Models and Optimism-Driven Exploration. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 504–511, 2016.

F. Berkenkamp, A. P. Schoellig, and A. Krause. Safe Controller Optimization for Quadrotors with Gaussian Processes. In *Proc. of the Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 491–496, 2016.

C. Ostafew, A. P. Schoellig, and T. Barfoot. Conservative to Confident: Treating Uncertainty Robustly Within Learning-Based Control. In *Proc of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 421–427, 2015.

C. Ostafew, A. P. Schoellig, and T. Barfoot. Robust Constrained Learning-Based NMPC Enabling Reliable Mobile Robot Path Tracking. *Intl. Journal of Robotics Research (IJRR)*, 35(13):1547–1563, 2016.

A. Aswani, H. Gonzalez, S. Sastry, and C. Tomlin. Provably Safe and Robust Learning-based Model Predictive Control. *Automatica*, 49(5):1216–1226, 2013.

G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information Theoretic MPC for Model-Based Reinforcement Learning. In *Proc. of the International on Robotics and Automation (ICRA)*, pages 1714–1721, 2017.

B. Luders, I. Sugel, and J. How. Robust Trajectory Planning for Autonomous Parafoils Under Wind Uncertainty. In *Proc. of the AIAA Conf. on Guidance, Navigation and Control*, 2013.

Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig. Deep Neural Networks for Improved, Impromptu Trajectory Tracking of Quadrotors. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 5183–5189, 2017.

G. Aoude, B. Luders, J. Joseph, N. Roy, and J. How. Probabilistically Safe Motion Planning to Avoid Dynamic Obstacles with Uncertain Motion Patterns. *Autonomous Robots*, 35(1):51–76, 2013.

C. McKinnon and A. P. Schoellig. Learning Multi-Modal Models for Robot Dynamics with a Mixture of Gaussian Process Experts. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 322–328, 2017.

C. McKinnon and A.P. Schoellig. Experience-Based Model Selection to Enable Long-Term, Safe Control for Repetitive Tasks Under Changing Conditions. In *Proc. of the Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2977–2984, 2018.

C. McKinnon and A. P. Schoellig. Learn Fast, Forget Slow: Safe Predictive Learning Control for Systems with Unknown and Changing Dynamics Performing Repetitive Tasks. *Robotics and Automation Letters*, 4(2):2180–2187, 2019a.

C. McKinnon and A. P. Schoellig. Learning Probabilistic Models for Safe Predictive Control in Unknown Environments. In *Proc. of the European. Conf. Control (ECC)*, pages 2472–2479, 2019b.

C. McKinnon and A. P. Schoellig. Meta Learning with Paired Forward and Inverse Models for Efficient Receding Horizon Control. *Robotics and Automation Letters*, 6 (2):3240–3247, 2021.

C. McKinnon and A. P. Schoellig. Context-Aware Cost Shaping to Reduce the Impact of Model Error in Receding Horizon Control. In *Proc. of the Intl. Conf. on Intelligent Robotics and Automation (ICRA)*, pages 2386–2392, 2020.

A. Mesbah. Stochastic Model Predictive Control: An Overview and Perspectives for Future Research. *Control Systems*, 36(6):30–44, 2016.

M. Morari and J. Lee. Model Predictive Control: Past, Present and Future. *Computers & Chemical Engineering*, 23(4-5):667–682, 1999.

W. Sun, S. Patil, and R. Alterovitz. High-Frequency Replanning Under Uncertainty Using Parallel Sampling-based Motion Planning. *Robotics*, 31(1):104–116, 2015.

J. Kabzan, L. Hewing, A. Liniger, and M. Zeilinger. Learning-Based Model Predictive Control for Autonomous Racing. *Robotics and Automation Letters*, 4(4):3363–3370, 2019.

T. Koller, F. Berkenkamp, M. Turchetta, and A. Krause. Learning-based Model Predictive Control for Safe Exploration. In *Conf. on Decision and Control (CDC)*, pages 6059–6066, 2018.

S. Liu, M. Watterson, S. Tang, and V. Kumar. High Speed Navigation for Quadrotors with Limited Onboard Sensing. In *Intl. Conf. on Robotics and Automation (ICRA)*, pages 1484–1491, 2016.

J. Fisac, A. Akametalu, M. Zeilinger, S. Kaynama, J. Gillula, and C. Tomlin. A General Safety Framework for Learning-based Control in Uncertain Robotic Systems. *Transactions on Automatic Control*, 64(7):2737–2752, 2018.

D. Fridovich-Keil, J. Fisac, and C. Tomlin. Safely Probabilistically Complete Real-Time Planning and Exploration in Unknown Environments. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7470–7476, 2019.

U. Rosolia, X. Zhang, and F. Borrelli. A Stochastic MPC Approach with Application to Iterative Learning. In *Conf. on Decision and Control (CDC)*, pages 5152–5157, 2018.

U. Rosolia, X. Zhang, and F. Borrelli. Robust Learning Model Predictive Control for Iterative Tasks: Learning from Experience. In *Proc. of the Conf. on Decision and Control (CDC)*, pages 1157–1162, 2017a.

J Kober, D. Bagnell, and J. Peters. Reinforcement Learning in Robotics: A Survey. *Intl. Journal of Robotics Research (IJRR)*, (11):1238–1274, 2013b.

K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch. Plan Online, Learn Offline: Efficient Learning and Exploration via Model-based Control. *arXiv preprint arXiv:1811.01848*, 2018.

IBM. IBM ILOG CPLEX Optimization Studio 12.7.1. URL www.ibm.com.

A. Der Kiureghian and O. Ditlevsen. Aleatory or Epistemic? Does it Matter? *Structural Safety*, 31(2):105–112, 2009.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. ISBN 0-262-18253-X. URL http://www.gaussianprocess.org/gpml/.

P. Bouffard, A. Aswani, and C. Tomlin. Learning-based Model Predictive Control on a Quadrotor: Onboard Implementation and Experimental Results. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 279–284, 2012.

K. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

C. Ostafew, A. P. Schoellig, and T. Barfoot. Learning-based Nonlinear Model Predictive Control to Improve Vision-Based Mobile Robot Path-tracking in Challenging Outdoor Environments. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 4029–4036, 2014a.

J. Gillula and C. Tomlin. Reducing Conservativeness in Safety Guarantees by Learning Disturbances Online: Iterated Guaranteed Safe Online Learning. In *Proc. of Robotics: Science and Systems (RSS)*, pages 81–88, 2012.

J. Mahler, S. Krishnan, M. Laskey, S. Sen, A. Murali, B. Kehoe, S. Patil, J. Wang, M. Franklin, P. Abbeel, et al. Learning Accurate Kinematic Control of Cable-driven Surgical Robots using Data Cleaning and Gaussian Process Regression. In *Proc. of the Intl. Conf. on Automation Science and Engineering (CASE)*, pages 532–539, 2014.

J. Kocijan, R. Murray-Smith, C. Rasmussen, and A. Girard. Gaussian Process Model-Based Predictive Control. In *Proc of the American Control Conference (ACC)*, volume 3, pages 2214–2219, 2004.

L. Hewing, A. Liniger, and M. N. Zeilinger. Cautious NMPC with Gaussian Process Dynamics for Miniature Race Cars. In *Proc. of the European Control Conf.*, pages 1341–1348, 2018a.

J. Ting, A. D'Souza, S. Vijayakumar, and S. Schaal. A Bayesian Approach to Empirical Local Linearization for Robotics. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 2860–2865, 2008.

F. Meier, D. Kappler, N. Ratliff, and S. Schaal. Towards Robust Online Inverse Dynamics Learning. In *Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4034–4039, 2016.

F. Meier and S. Schaal. Drifting Gaussian Processes with Varying Neighborhood Sizes for Online Model Learning. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 264–269, 2016.

A. Lederer, Alejandro J. C., K. Maier, W. Xiao, and S. Hirche. Real-Time Regression with Dividing Local Gaussian Processes. *arXiv preprint arXiv:2006.09446*, 2020.

D. Nguyen-Tuong, J. Peters, and M. Seeger. Local Gaussian Process Regression for Real Time Online Model Learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1193–1200, 2009a.

V. Desaraju, A. Spitzer, and N. Michael. Experience-Driven Predictive Control with Robust Constraint Satisfaction Under Time-Varying State Uncertainty. In *Proc. of the Robotics: Science and Systems Conf. (RSS)*, 2017.

D. Nguyen-Tuong, M. Seeger, and J. Peters. Model Learning with Local Gaussian Process Regression. *Advanced Robotics*, 23(15):2015–2034, 2009b.

K. Jo, K. Chu, and M. Sunwoo. Interacting Multiple Model Filter-Based Sensor Fusion of GPS with In-vehicle Sensors for Real-time Vehicle Positioning. *Transactions on Intelligent Transportation Systems*, 13(1):329–343, 2012.

R. Calandra, S. Ivaldi, M. Deisenroth, E. Rueckert, and J. Peters. Learning Inverse Dynamics Models with Contacts. In *Intl. Conf. on Robotics and Automation (ICRA)*, pages 3186–3191, 2015a.

E. Fox, E. Sudderth, M. Jordan, and A. Willsky. Nonparametric Bayesian Learning of Switching Linear Dynamical Systems. In *Proc. of Advances in Neural Information Processing Systems (NIPS)*, pages 457–464, 2009.

C. Rasmussen and Z. Ghahramani. Infinite Mixtures of Gaussian Process Experts. In *Proc. of Advances in Neural Information Processing Systems (NIPS)*, volume 2, pages 881–888, 2002.

C. Linegar, W. Churchill, and P. Newman. Work Smart, Not Hard: Recalling Relevant Experiences for Vast-Scale but Time-Constrained Localisation. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 4137–4143, 2015.

GPy. GPy: A Gaussian Process Framework in Python. http://github.com/SheffieldML/GPy, since 2012.

C. Xie, S. Patil, T. Moldovan, S. Levine, and P. Abbeel. Model-Based Reinforcement Learning with Parametrized Physical Models and Optimism-Driven Exploration. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 504–511, 2016.

F. Berkenkamp. *Safe Exploration in Reinforcement Learning: Theory and Applications in Robotics*. PhD thesis, ETH Zurich, 2019.

A. Angelova. *Visual Prediction of Rover Slip: Learning Algorithms and Field Experiments*. Phd thesis, California Institute of Technology, 2008.

L. Hewing, A. Liniger, and M. Zeilinger. Cautious NMPC with Gaussian Process Dynamics for Autonomous Miniature Race Cars. In *Proc. of the European Control Conf. (ECC)*, pages 1341–1348, 2018b.

A. Akametalu, J. Fisac, J. Gillula, S. Kaynama, M. Zeilinger, and C. Tomlin. Reachability-based Safe Learning with Gaussian Processes. In *Proc. of the Conf. on Decision and Control (CDC)*, pages 1424–1431, 2014.

L. Jamone, B. Damas, and J. Santos-Victor. Incremental Learning of Context-dependent Dynamic Internal Models for Robot Control. In *Intl. Symp. on Intelligent Control (ISIC)*, pages 1336–1341, 2014.

U. Rosolia, A. Carvalho, and F. Borrelli. Autonomous Racing Using Learning Model Predictive Control. In *American Control Conf. (ACC)*, pages 5115–5120, 2017b.

M. Lázaro-Gredilla, J. Quiñonero-Candela, C. Rasmussen, and A. Figueiras-Vidal. Sparse Spectrum Gaussian Process Regression. *The Journal of Machine Learning Research*, 11:1865–1881, 2010.

A. Gijsberts and G. Metta. Real-Time Model Learning Using Incremental Sparse Spectrum Gaussian Process Regression. *Neural Networks*, 41:59–69, 2013.

D. Lam, C. Manzie, and M. Good. Model Predictive Contouring Control. In *Conf. on Decision and Control (CDC)*, pages 6137–6142, 2010.

C. Van Loan. *Introduction to Scientific Computing: A Matrix Vector Approach Using MATLAB*. Prentice Hall, 1997.

L. Hewing and M. N. Zeilinger. Cautious Model Predictive Control Using Gaussian Process Regression. In *arXiv:1705.10702*, 2017.

Y. Gao, A. Gray, H. Tseng, and F. Borrelli. A Tube-based Robust Nonlinear Predictive Control Approach to Semiautonomous Ground Vehicles. *Vehicle System Dynamics*, 52 (6):802–823, 2014.

M. Vitus and C. Tomlin. On Feedback Design and Risk Allocation in Chance Constrained Control. In *Proc. of the Decision and Control and European Control Conf. (CDC-ECC), 2011 50th IEEE Conf. on*, pages 734–739, 2011.

M. Paton. *Expanding the Limits of Vision-Based Autonomous Path Following*. PhD thesis, University of Toronto, 2018.

BV Niekerk, A Damianou, and B Rosman. Online Constrained Model-Based Reinforcement Learning. In *Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2017.

N. Mohajerin and S. Waslander. Multi-Step Prediction of Dynamic Systems with Recurrent Neural Networks. *Transactions on Neural Networks and Learning Systems*, pages 1–14, 2019.

J. Harrison, A. Sharma, and M. Pavone. Meta-Learning Priors for Efficient Online Bayesian Regression. In *Intl. Workshop on the Algorithmic Foundations of Robotics*, pages 318–337, 2018.

Michael O'Connell, Guanya Shi, Xichen Shi, and Soon-Jo Chung. Meta-Learning-Based Robust Adaptive Flight Control under Uncertain Wind Conditions. 2020.

C. Finn, P. Abbeel, and S. Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, volume 70, pages 1126–1135. JMLR. org, 2017.

T. Lew, A. Sharma, J. Harrison, and M. Pavone. Safe Learning and Control using Meta-Learning. *openreview*, 2019.

M. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, 2020.

S. Zhou, M. Helwa, and A. Schoellig. Design of Deep Neural Networks as Add-on Blocks for Improving Impromptu Trajectory Tracking. In *Proc. of the Conf. on Decision and Control (CDC)*, pages 5201–5207, 2017.

R. Calandra, S. Ivaldi, M. Deisenroth, E. Rueckert, and J. Peters. Learning Inverse Dynamics Models with Contacts. In *Intl. Conf. on Robotics and Automation (ICRA)*, pages 3186–3191, 2015b.

Athanasios S Polydoros, Lazaros Nalpantidis, and Volker Krüger. Real-Time Deep Learning of Robotic Manipulator Inverse Dynamics. In *Proc. of the Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3442–3448, 2015.

Duy Nguyen-Tuong and Jan Peters. Using Model Knowledge for Learning Inverse Dynamics. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 2677–2682, 2010.

J. Sun de la Cruz. Learning Inverse Dynamics for Robot Manipulator Control. Master's thesis, University of Waterloo, 2011.

M. Helwa, A. Heins, and A. P. Schoellig. Provably Robust Learning-Based Approach for High-Accuracy Tracking Control of Lagrangian Systems. *Robotics and Automation Letters*, 4(2):1587–1594, 2019.

R. Pautrat, K. Chatzilygeroudis, and J. Mouret. Bayesian Optimization with Automatic Prior Selection for Data-Efficient Direct Policy Search. In *arXiv:1709.06919*, 2017.

M. Sorocky, S. Zhou, and A. P. Schoellig. Experience Selection Using Dynamics Similarity for Efficient Multi-Source Transfer Learning Between Robots. In *Proc. of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 2739–2745, 2020.

D. Wolpert and M. Kawato. Multiple Paired Forward and Inverse Models for Motor Control. *Neural Networks*, 11(7-8):1317–1329, 1998.

G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin. Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment. In *Proc of the Guidance, Navigation and Control Conf.*, page 6461, 2007.

F. Augugliaro and R. D'Andrea. Admittance control for physical human-quadrocopter interaction. In *Proc. of the European Control Conference (ECC)*, pages 1805–1810, 2013.

V. Hagenmeyer and E. Delaleau. Exact Feedforward Linearization Based on Differential Flatness. *Intl. Journal of Control*, 76(6):537–556, 2003.

Michael I Jordan and David E Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16(3):307–354, 1992.

K. Margellos and J. Lygeros. A Simulation Based MPC Technique for Feedback Linearizable Systems with Input Constraints. In *Conf. on Decision and Control (CDC)*, pages 7539–7544, 2010.

J. Deng, V. Becerra, and R. Stobart. Input Constraints Handling in an MPC/Feedback Linearization Scheme. *Intl. Journal of Applied Mathematics and Computer Science*, 19(2):219–232, 2009.

M. Kurtz and M. Henson. Feedback Linearizing Control of Discrete-time Nonlinear Systems with Input Constraints. *Intl. Journal of Control*, 70(4):603–616, 1998.

M. Mueller and R. D'Andrea. A Model Predictive Controller for Quadrocopter State Interception. In *European Control Conf. (ECC)*, pages 1383–1389, 2013.

S. Zhou, M. Helwa, and A. Schoellig. Deep Neural Networks as Add-on Modules for Enhancing Robot Performance in Impromptu Trajectory Tracking. *The Intl. Journal of Robotics Research (IJRR)*, 0(0):1–22, 2020.

A. Venkatraman, B. Boots, M. Hebert, and J. Bagnell. Data as Demonstrator with Applications to System Identification. In *ALR Workshop, NIPS*, 2014.

A. Doerr, C. Daniel, D. Nguyen-Tuong, A. Marco, S. Schaal, T. Marc, and S. Trimpe. Optimizing Long-term Predictions for Model-based Policy Search. In *Proc. of the Conf. on Robot Learning (CoRL)*, pages 227–238, 2017.

U. Rosolia and F. Borrelli. Learning Model Predictive Control for Iterative Tasks. a Data-driven Control Framework. *Automatic Control*, 63(7):1883–1896, 2017.

A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel. Learning from the Hindsight Plan—Episodic MPC Improvement. In *Proc of the Intl. Conf. on Robotics and Automation (ICRA)*, pages 336–343, 2017.

C. Ostafew, A. P Schoellig, T. D. Barfoot, and J. Collier. Speed daemon: experience-based mobile robot speed scheduling. In *Proc. of the Canadian Conference on Computer and Robot Vision (CRV)*, pages 56–62, 2014b.

A. Ng, D. Harada, and S. Russell. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proc of the Intl. Conf. on Machine Learning (ICML)*, volume 99, pages 278–287, 1999.

J. Asmuth, M. Littman, and R. Zinkov. Potential-Based Shaping in Model-based Reinforcement Learning. In *AAAI*, pages 604–609, 2008.

M. Grześ. Reward Shaping in Episodic Reinforcement Learning. In *Proc of the Conf. on Autonomous Agents and MultiAgent Systems*, pages 565–573, 2017.