Learning-based Control for Autonomous Mobile Robots

by

Christopher John Ostafew

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Aerospace Science and Engineering
University of Toronto

Abstract

Learning-based Control for Autonomous Mobile Robots

Christopher John Ostafew
Doctor of Philosophy
Graduate Department of Aerospace Science and Engineering
University of Toronto
2016

As mobile robots leave structured indoor environments to operate in challenging outdoor environments, their motion controllers require advanced techniques to mitigate the effects of unmodelled surface materials (e.g., snow, sand, grass), terrain topography (e.g., side-slopes, inclines), and complex robot dynamics. This thesis investigates learning-based control within the context of path-tracking autonomous mobile robots. Learning-based algorithms alleviate the need for significant engineering work in identifying and modelling all disturbances that a controller may be required to mitigate. Furthermore, they are capable of predicting and acting in anticipation of repeatable effects and disturbances not modelled prior to deployment. The learning-based algorithms reduce tracking errors, increase operational speed, and increase localization reliability. Specifically, the thesis presents four approaches: 1) Iterative Learning Control (ILC), 2) Learning-based Nonlinear Model Predictive Control (LB-NMPC), 3) Robust Min-Max LB-NMPC (MM-LB-NMPC), and 4) Robust Constrained LB-NMPC (RC-LB-NMPC). ILC generates an acausal feedforward signal that reduces the path-tracking errors using information from any previous trial. While the approach is computationally appealing, ILC typically assumes that the robot is initialized with identical initial conditions for each trial and tracking the same desired path (i.e., generalization is non-trivial). On the other hand, LB-NMPC is a technique that uses a learned process model directly, enabling interpolation and extrapolation from experiences. In this case, the current control action is obtained by solving a finite-horizon optimal control problem using the current

state of the plant as the initial state at each time-step. Finally, this thesis investigates two recent results in Robust NMPC in order to guarantee controller stability throughout the learning process in spite of model uncertainty. For MM-LB-NMPC, the control problem is altered to optimize for plausible worst-case scenarios. For RC-LB-NMPC, tightened constraints are applied to nominal predictions such that all plausible predicted sequences satisfy the given constraints. The resulting RC-LB-NMPC algorithm is a robust, learning controller providing safe, conservative control during initial trials when model uncertainty is high and converging to high-performance, optimal control during later trials when model uncertainty is reduced with experience.

Acknowledgements

This thesis would not have been possible without the help of many people. First, I would like to thank my family for their support in this endeavour. To my parents, thank you for always being there. To my sister, I am proud of our adventures and look forward to what lay ahead. To Scott and Gerry, I have enjoyed our long discussions and am continually inspired by your examples. To the Grays, thank you for your generosity and acceptance. To my son, Ethan, thank you for the motivation and encouragement in finishing this thesis. Finally, to my wife, Megan, you have always encouraged the best of me and I will be forever grateful.

I would also like to thank my supervisors, Prof. Tim Barfoot and Prof. Angela Schoellig, for the much appreciated guidance and feedback over the years. I entered graduate studies with a goal of finding excellent role models and am thankful for having the opportunity to work with you. Many thanks to my colleagues at the Autonomous Space Robotics Lab (ASRL) as well for insightful conversations, the patience running the robot E-Stop during my experiments, and those Friday night barbecues enjoying the sunsets. I would also like to thank Paul Furgale, Chi Hay Tong, Colin McManus, and Mike Paton for the implementation of the vision-based localization system enabling the experimental results presented in this thesis.

Contents

List of Figures

Acronyms

C-LB-NMPC	:	Constrained LB-NMPC.
DMRV	:	Defence research and development Mule Research Vehicle.
GP	:	Gaussian Process.
GPS	:	Global Positioning System.
ILC	:	Iterative Learning Control.
LB-NMPC	:	Learning-based Nonlinear Model Predictive Control.
LGP	:	Local Gaussian Process.
MM-LB-NMPC	:	Min-Max LB-NMPC.
MPC	:	Model Predictive Control.
NMPC	:	Nonlinear Model Predictive Control.
PD-ILC	:	Proportional-Derivative Iterative Learning Control.
RC-LB-NMPC	:	Robust Constrained LB-NMPC.
RMS	:	Root-Mean-Square.
SURF	:	Speeded Up Robust Features.
UTIAS	:	University of Toronto Institute for Aerospace Studies.
VO	:	Visual Odometry.
VT&R	:	Visual Teach & Repeat.

Notation

a	:	Symbols in this font are real scalars.
$\mathbf{a}$	:	Symbols in this font are real column vectors.
$\mathbf{A}$	:	Symbols in this font are real matrices.
$\mathcal{N}(\bar{\mathbf{a}}, \mathbf{B})$	:	Normally distributed with mean $\bar{\mathbf{a}}$ and covariance $\mathbf{B}$.
$\tilde{\mathbf{a}}$	:	Nominal value of perturbed vector (i.e., $\mathbf{a} \approx \tilde{\mathbf{a}} + \delta\mathbf{a}$).
$\hat{\mathbf{a}}$	:	Estimate of vector $\mathbf{a}$.
$\mathbf{a}^{(j)}$	:	Column vector $\mathbf{a}$ occurring in the jth trial.
$\mathbf{B}^{(j)}$	:	Matrix $\mathbf{B}$ occurring in the jth trial.
$\mathbf{1}$	:	The identity matrix.
$\mathbf{0}$	:	The zero matrix.

Chapter 1

Introduction

Autonomous mobile robots can be incredibly useful in many fields, especially those dealing with dull, dirty, or dangerous tasks in challenging off-road terrain. For example, a few of the fields that could benefit most by autonomy include transportation (Thrun et al., 2006; Urmson et al., 2008), planetary exploration (Maimone et al., 2007; Johnson et al., 2008), forestry (Hellström et al., 2006; Rossmann et al., 2009), agriculture (Foglia and Reina, 2006), and mining (Marshall et al., 2008). Figure 1.1 shows several large machines that have been used in experimental work developing algorithms for autonomous mobile robots.

In such applications, mobile robots can provide several benefits. For example, they can provide increased productivity through consistent operation over long-term tasks. In addition to being unaffected by fatigue, they can be applied to long-term tasks facing inclemental weather provided they are equipped with appropriate sensors. For example, radar is capable of sensing through fog and lidar sensors are capable of sensing in darkness. Furthermore, they can provide improved safety and reduced operating costs for hazardous operations by removing the need for on-board human operators and the resulting life-support and protective equipment (i.e., mining, planetary exploration, and forestry). Without the need for life-support systems, autonomous mobile robots can also be applied to new applications since they can be designed to withstand conditions that would be harmful to humans such as extreme temperatures, pressures, and radiation levels. In other applications, they can improve safety by reducing collisions using high-rate, accurate omnidirectional sensing (i.e., transportation).

In many applications, mobile robot operation in off-road, outdoor environments can be characterized by the problem of autonomous traversal (i.e., the problem of getting from

Figure 1.1: Example machines that have been used as experimental autonomous mobile robots. (Top left) The Valmet 830 forest machine is used to shuttle timber from a felling area to a road (Hellström et al., 2006). (Top right) Stanley is the autonomous car created by Stanford University that won the 2005 DARPA Grand Challenge (Thrun et al., 2006). (Bottom) The Atlas Copco LH14 load-haul-dump machine is used to shuttle excavated, fragmented rock within underground mines (Marshall et al., 2008).

point A to point B) with the goal of transporting payloads. The problem is typically described by three components: Guidance, Navigation, and Control. First, a guidance algorithm must plan a safe and efficient path through the environment avoiding obstacles such as trees, boulders, or dynamic objects. In addition to planning, this component also relies on accurate terrain assessment and obstacle detection, both of which struggle in unstructured, outdoor environments. In many mobile robot applications, it is adequate if not necessary, to explore and navigate the environment by creating and maintaining a network of paths analogous to automotive roads. The use and reuse of paths reduces the need for repeated application of exploratory and terrain assessing software. Second, the mobile robot must localize relative to the desired path. Although the Global Positioning System (GPS) has become ubiquitous in everyday use, GPS can be unreliable due to signal blockage. Specifically, it is typically unreliable in exactly the environments where mobile robots need to localize: urban canyons, extraterrestrial bodies, forests, and underground. As a result, significant research has been focused on leveraging cameras and

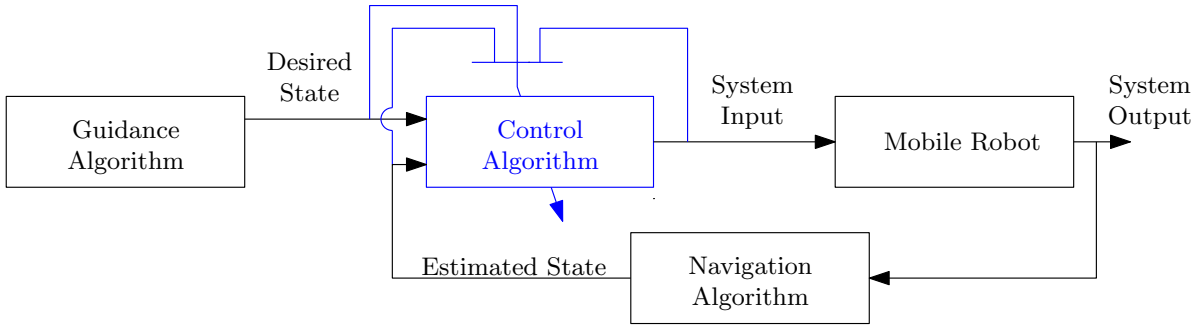


Figure 1.2: Block diagram showing the general organization of guidance, navigation, and control algorithms for autonomous mobile robots. In this thesis, experiences based on desired state, estimated state, and system input are gathered over time and used to update the controller at the end of each path traversal or trial through learning (dashed line). At any given time, the improved controller uses real-time desired and estimated states to compute the system input.

lidar units for localization. Such approaches typically generate relative state estimates based on local landmarks and scenery. Finally, given a desired path and localization relative to the path, the mobile robot must compute commands to be sent to the vehicle actuators that keep the machine on the desired path. This thesis explores the problem of computing high-performance control inputs through learning (Figure 1.2).

1.1 Mobile Robot Control

Many mobile robots, including those shown in Figure 1.1, achieve locomotion using wheels and can be modelled as either unicycles or bicycles (Oriolo, 2014), both examples of non-holonomic systems. Such systems require fundamentally nonlinear approaches for path-tracking control (Brockett, 1983; Kolmanovsky and McClamroch, 1995). As a result, initial work on path-tracking control for wheeled mobile robots focused on finding solutions considering operation in simple, small-scale, indoor environments. Two common approaches include Feedback Linearization (Samson and Ait-Abderrahim, 1991; De Luca and Di Benedetto, 1993) and Lyapunov techniques (Kanayama et al., 1990; Aicardi et al., 1995). Feedback Linearization proposes a time-varying transformation of the nonlinear system into an equivalent linear system. The feedback-linearized system is then stabilized with a suitable control input. On the other hand, Lyapunov techniques propose a non-linear, scalar, positive-definite function of the process states and solve for time-varying control laws such that the time derivative of the Lyapunov function is negative-definite.

Figure 1.3: Practical mobile-robot controllers face modelling challenges due to surface materials, terrain topography, and complex robot dynamics. In this work, controllers are tested on robots ranging from the skid-steered, 50 kg Husky and 900 kg Grizzly robots (left and center), to the Ackermann-steered, 600 kg MATS vehicle (right) in off-road terrain using vision-based localization.

Between these two elegant approaches, Oriolo et al. (2002) later declared the problem of controlling nonholonomic systems virtually solved from a theoretical point of view. However, the stability and performance of these approaches depend heavily upon the accuracy of the unicycle and bicycle models.

Nonholonomic constraints are not the only modelling challenge facing practical mobile robot control. Ultimately, operation in off-road terrain requires the controller to mitigate the effects of surface materials (e.g., snow, sand, grass), terrain topography (e.g., side-slopes, inclines), and complex robot dynamics (Figure 1.3). As a result, research shifted towards compensating for modelling errors. Initially, backstepping control and adaptive control were widely investigated to mitigate uncertain dynamics (Fierro and Lewis, 1998; Fukao et al., 2000) and uncertain kinematics such as caused by wheel slip (Dong and Kuhnert, 2005; Cariou et al., 2009; Guillet et al., 2013). However, these examples are founded on Feedback Linearization or Lyapunov techniques with parameterized process models, and are thus limited by the requirements to find either an appropriate nonlinear transformation or an applicable scalar Lyapunov function. As opposed to such approaches that restrict model fidelity, Model Predictive Control (MPC) is an optimal control framework that uses a process model directly (Rawlings and Mayne, 2009; Mayne, 2014). MPC in general has received a significant amount of attention in recent years due to its ability

to handle complex linear and nonlinear systems with state and input constraints. For example, Kühne et al. (2005) and Klančar and Škrjanc (2007) present MPC-based mobile robot controllers based on kinematic models and show results for robots travelling on smooth, flat surfaces. Peters and Iagnemma (2008) demonstrate MPC for a mobile robot where the process model includes effects such as tire deformation, wheel-terrain interaction, and suspension compliance. Notably, MPC has enabled algorithms to use richer models in computing control inputs.

In practice, finding representative *a priori* models for off-road effects is challenging since (i) the terrain is often not known ahead of time, (ii) robot-terrain interaction models often do not exist, and (iii) even if such models did exist, finding model parameters is cumbersome. In the last decade, there has been significant work on learning-based controllers for robotics in general (Schaal and Atkeson, 2010; Nguyen-Tuong and Peters, 2011). Learning-based algorithms collect data over sequential trials and use it to improve operation. For mobile robots, improvements may take the form of reduced path-tracking errors, increased operational speed, or increased localization reliability. As stated by Arimoto et al. (1984), credited with the development of a technique called Iterative Learning Control for manipulator robots, "It is human to make mistakes, but it is also human to learn much from experience."

Learning-based controllers produce the aforementioned improvements for mobile, path-repeating robots in several ways. First, learning-based algorithms address complex effects using data collected *in situ*. This alleviates the need for significant engineering work in identifying and modelling all disturbances that a controller may be required to mitigate prior to operation. Second, since learning-based algorithms collect data in operation, they are capable of compensating for gradually changing effects, such as robot wear, wheel ruts, or terrain changes due to weather or seasons. For example, wheel ruts cannot be reliably predicted prior to operation since they are the result of terrain properties and actual robot behavior. However, they can be readily characterized by data acquired during operation. Finally, learning-based algorithms are capable of predicting and acting in anticipation of repeatable disturbances along a desired path using disturbance observations that were collected and stored in memory during previous trials. This distinguishes learning-based algorithms from adaptive algorithms that only react to tracking errors by briefly using disturbance observations to adjust controller parameters before discarding the information.

1.2 Thesis Overview

Motivated by modelling challenges stemming from outdoor environments, this thesis focuses on learning-based control enabling high-performance, off-road, mobile robot operation. The work progressively investigates algorithms addressing increasingly demanding capabilities largely inspired by insights gained through experimental work:

Computationally Tractable: Practical learning-based controllers for mobile robots must be capable of producing real-time inputs if they are to improve safety and productivity with respect to human operators. In this work, we consider 10 Hz to be a minimum update frequency for real-time operation of mobile robots. Furthermore, we aim for algorithms that can be implemented by a single CPU core, leaving other cores to be used by the necessary guidance and navigation algorithms.

Large-scale Operation: In addition to real-time operation, practical learning-based controllers must be capable of operating on multi-km networks of paths. This suggests that any proposed learning-based controller should be capable of providing real-time control independent of the size of the dataset. In order to handle large datasets in real-time, we build upon state-of-the-art machine learning techniques and optimization techniques.

Efficient Training: Training is time consuming and expensive in practice! Prior to learning, a robot is likely travelling at slower speeds resulting in extended battery use and exposure to path-tracking errors. In an effort to make the most of training time, practical learning-based controllers should enable safe path completion during the first trial and rapid improvements over following trials. Furthermore, we show that anytime learning and the ability to generalize from collected data further maximize the value extracted from previous experience.

Robust Control: In general, our proposed algorithms begin with a nominal controller based on *a priori* knowledge of the system process model and improve it with machine learning techniques. Controller robustness, i.e. the capability of a controller to maintain stability of a system in spite of model uncertainty, is an open question for learning controllers in general. However, practical learning-based controllers need to offer controller robustness as robots tackle increasingly challenging and unknown environments at higher speeds.

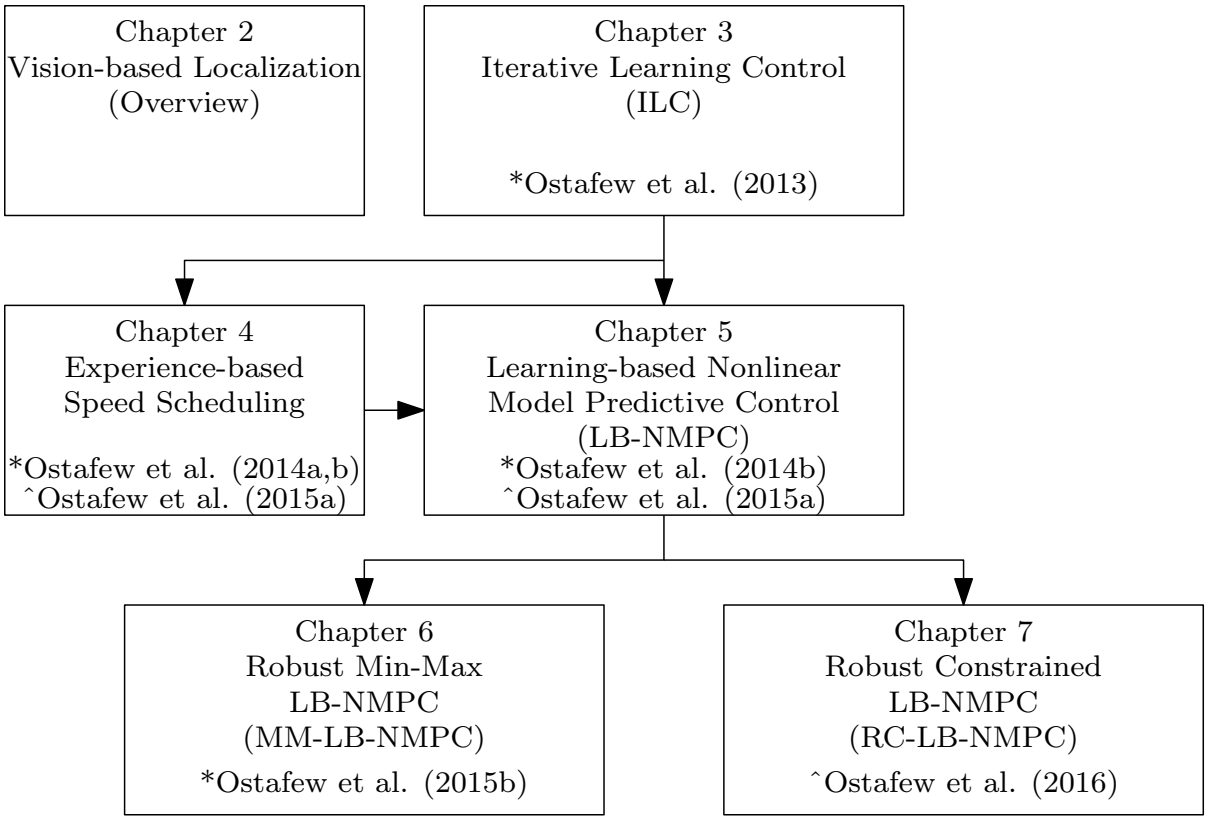


Figure 1.4: A diagram depicting the logical flow of this thesis with the associated publications (* full-paper refereed conference papers, ˆ journal papers).

The structure of this thesis is shown in Figure 1.4 where we identify the logical flow of the work. We begin with Chapter 2 where we present an overview of the vision-based path localizer that provides state estimates for all learning-based algorithms presented in this thesis.

In Chapter 3, we introduce Iterative Learning Control (ILC). ILC was initially proposed by Arimoto et al. (1984) to improve the operation of fixed-base robots. ILC constructs an acausal feedforward signal over sequential trials using error information from any previous trial. The contribution of this chapter is a learning-based algorithm for mobile robots composed of ILC in parallel with a feedback controller. This enables path completion during the first trial, and improved performance thereafter. Moreover, we propose a novel formulation enabling the robot to track the desired path at a safe speed during the first trial and increased speed during later trials. Finally, we provide the first experimental results for ILC on challenging, off-road paths including over 700 m of travel by two significantly different skid-steered robots[1].

[1]Associated video at http://tiny.cc/RobotLearnsIteratively

Chapter 4 presents an automated speed scheduler to provide safe desired speeds considering previous path-tracking errors, control inputs, and localization reliability. In our experiments with ILC, angular speeds were computed by the controller while a constant linear speed was manually selected prior to each trial based on operator experience. This revealed an opportunity to use experience to compute time-optimal speed schedules for any given trial. A time-optimal speed schedule results in a mobile robot driving along a planned path at or near the limits of the robot's capability. However, deriving models to predict the effect of increased speed can be very difficult. The novelty here is the incorporation of experience when computing a speed schedule in order to guarantee low path-tracking errors and reliable localization. The proposed speed scheduler was tested in over 4 km of travel on outdoor terrain using a large Ackermann-steered robot travelling between 0.5 m/s and 2.0 m/s and is later used to specify training speeds.

In Chapter 5, we present Learning-based Nonlinear Model Predictive Control (LB-NMPC), a learning-based algorithm capable of interpolating and extrapolating from previous experience. In practice, ILC is computationally appealing for real-time applications or applications with low computing power since the ILC feedforward signal is computed offline. However, we found in our initial work with ILC that anytime learning and the ability to generalize from learned experiences are key requirements for flexible (and useful) learning-based mobile robot operation in large-scale, outdoor environments. For example, a learning-based controller should be capable of maintaining and accessing learned experiences for many paths, speeds, and combinations thereof in order to operate in a changing environment. In NMPC, the current control action is obtained by solving a finite-horizon optimal control problem at each time-step based on the current state and a model of the system (Rawlings and Mayne, 2009). Furthermore, the process model is typically specified *a priori* and remains unchanged during operation. The contribution in this chapter is an NMPC algorithm where the *a priori* process model is augmented with an experience-based disturbance model. The resulting controller compensates for effects not captured by the fixed process model, such as environmental disturbances and unknown dynamics. Inspired by local learning, we present a novel approach where disturbances are modelled as a Gaussian Process (GP), and predictions are computed based on a sliding window of training data. This allows for real-time operation over long paths, anytime learning, and the ability to generalize from experience. Finally, we present extensive real-world testing on three significantly different robots in over 3 km of travel[2].

[2]Associated video at http://tiny.cc/RobotLearnsDisturbances

In general, our LB-NMPC algorithm is initialized with an *a priori* model and learns the discrepancies between the known model and actual robot behavior. Therefore, the augmented process model used by our LB-NMPC algorithm has varying levels of uncertainty depending on disturbance observations collected during previous trials. In Chapters 6 and 7, we investigate robust LB-NMPC algorithms to explicitly deal with the uncertainty in the disturbance model and guarantee safe, collision-free path-tracking throughout the learning process. Ideally, the robust extensions to the LB-NMPC algorithm automatically choose between conservative inputs when the learned model is relatively uncertain, and optimal, high performance inputs when model uncertainty is reduced through learning. Both chapters present experimental results showing improvement of the controllers in the face of modelling errors that occur naturally throughout the learning process.

Specifically, we investigate robust Min-Max LB-NMPC (MM-LB-NMPC) in Chapter 6. When optimizing based on the nominal predicted behavior, i.e., not taking model uncertainty into account, worst-case tracking errors due to modelling errors and an overconfident controller can be large. In our LB-NMPC algorithm, these tracking errors were mitigated by low scheduled speeds. However, in this work, we aim to reduce worst-case errors by altering the NMPC performance function to solve for optimal controls based on the worst-case scenario. This represents the first example of MM-LB-NMPC to our knowledge. Furthermore, the novelty in this work also includes the efficient prediction of the mean and uncertainty of state sequences considering the learned model using a Sigma-Point Transform. We present results from testing on a demonstrative path and show the reductions in worst-case tracking errors due to the robust controller.

In Chapter 7, we investigate Robust Constrained LB-NMPC. Robust Constrained MPC (RC-MPC) is an active area of research and endeavours to guarantee constraint satisfaction when considering uncertain systems (Mayne, 2014). State constraints provide a direct method of incorporating real-time obstacle information from an on-board guidance algorithm. At each time-step, RC-MPC aims to identify control inputs that result in constraint satisfaction by all plausible predicted trajectories considering a fixed estimate of model uncertainty. However, such approaches are generally conservative since the models are not updated online. The major contribution in this chapter is a RC-LB-NMPC algorithm combining several of the components developed throughout the thesis. Specifically, the algorithm combines the GP-based learned model (Chapter 5), efficient prediction of state sequences (Chapter 6), and state and input constraints. The result

is a robust, learning controller that provides safe, conservative control during initial trials when model uncertainty is high and high-performance, optimal control during later trials when model uncertainty is reduced with experience. We present extensive experimental results including over 5 km of travel by a 900 kg skid-steered robot at speeds up to 2.0 m/s showing constraint satisfaction and performance improvements over time[3]. Finally, Chapter 8 summarizes the contributions of this thesis and discusses future work.

[3]Associated video at http://tiny.cc/RobotLearnsRobustly

Chapter 2

Vision-based Localization

2.1 Introduction

One of the key requirements for the automation of off-road mobile robots is accurate and timely state estimation, a task that has been prohibitively challenging for many years. In this chapter, we present an overview of the vision-based localization algorithm (Furgale and Barfoot, 2010) that provides accurate state estimates for our learning-based controllers over long-distance, off-road paths. While visual localization is not a novel claim of this thesis, the resulting state estimates are a critical prerequisite for the learning algorithms described subsequently, and thus this chapter is included for completeness. Specifically, the longitudinal position along a path and the lateral and heading information with respect to a path are used as inputs to our learned models (i.e., they are a function of robot position relative to the desired path). It is important to note that GPS is not accurate enough to provide the required localization information, and thus this vision-based system is a key enabler of the learning algorithms in this thesis.

Stereo Visual Odometry (VO) is a technique for dead-reckoning a robot pose using a stereo camera as the primary sensor. Cameras are generally viewed as accurate, cost effective, real-time ($>10\,\text{Hz}$) sensors enabling mobile robot operation. VO is a sparse method, representing the world using a sparse set of key points tracked over a sequence of image pairs. Work on estimating robot motion with stereo VO can initially be traced back to work by Moravec (1980). His work was further developed by Matthies (Matthies and Shafer, 1987; Matthies, 1989) and later demonstrated on NASA's Mars Exploration Rovers (Maimone et al., 2007) and Mars Science Laboratory (Johnson et al., 2008).

Visual Teach & Repeat (VT&R) (Furgale and Barfoot, 2010; Furgale, 2011) is an

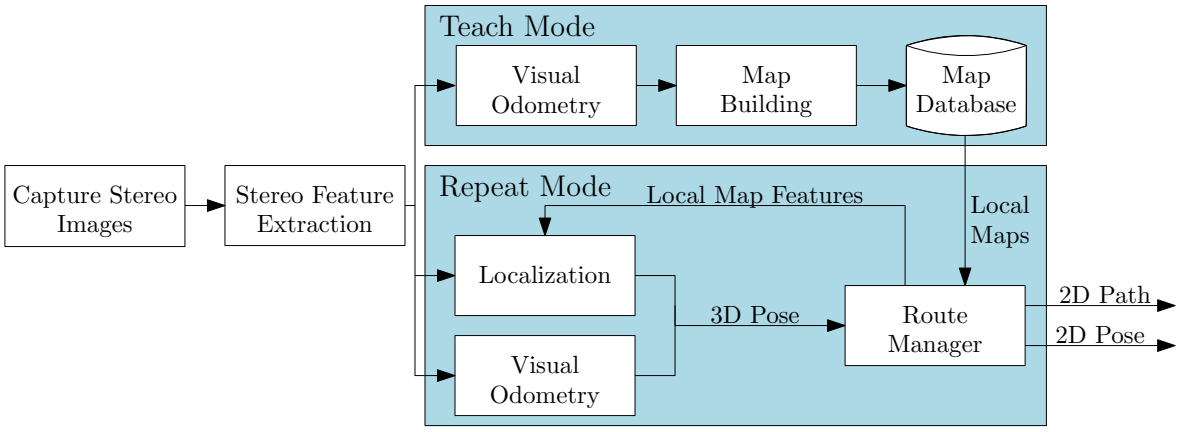


Figure 2.1: Overview of the vision-based localization algorithm.

extension to the VO pipeline that enables accurate and timely relocalization relative to a previously taught desired path (Figure 2.1). The ability of the VT&R algorithm to produce accurate, real-time state estimates in off-road environments is a key technology enabling this research. The localization algorithm operates in two phases. In the first operational phase, the visual teach phase, the robot is manually piloted along the desired path. Localization in this initial phase is obtained relative to the robot's starting position by VO. In addition to the VO pipeline, a Map Building component defines path vertices at short and regular intervals along the path, composed of a key frame and the associated local key points (feature descriptors and their 3D positions). During the repeat phase, the VT&R algorithm estimates the pose of the robot relative to the nearest key frame by relocalizing against the stored key points. Relocalization is achieved by matching feature descriptors to generate feature tracks between the current robot view and the teach-pass robot view. As long as sufficient correct feature matches are made, the system generates consistent localization over trials and is able to support a learning-based controller.

2.2 Mathematical Formulation

2.2.1 Feature Extraction

The first step in both teach and repeat phases is to extract relevant features from the most recent stereo image pair. The current implementation uses the Speeded Up Robust Features (SURF) algorithm (Bay et al., 2008) for feature detection and description. At time k, the algorithm generates a set of stereo key points from the latest stereo pair by

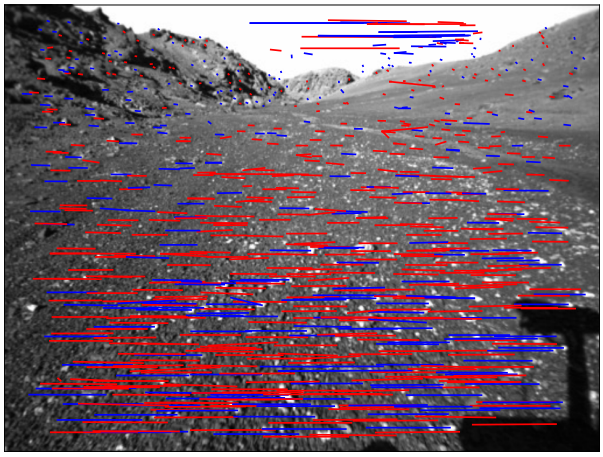


Figure 2.2: Example of key points extracted from a stereo image pair. Lines shown connect the left and right image locations associated with a single key point. Blue lines are from features based on light blobs on dark backgrounds, while red lines are from features based on dark blobs on light backgrounds. Image credit: Furgale (2011).

collecting SURF features from both the left and right images, and making descriptor-based associations between the pair (Figure 2.2). Key point b has image coordinates, $\mathbf{y}_{k,b} = (u, v, d)$, and an associated 64-dimensional description vector, $\mathbf{d}_{k,b}$. The image coordinates, u and v, are the horizontal and vertical pixel coordinates in the left image, respectively, while d is the disparity, the difference between the left and right horizontal pixel locations. Each key point also has a 3D feature location, $\mathbf{p}_k^{b,k} = (x, y, z)$, representing the position of feature b with respect to and expressed in the left camera frame, $\underrightarrow{\mathcal{F}}_{c_k}$ (Figure 2.3). The image and measurement coordinates are related by an observation model, $\mathbf{h}(\cdot)$,

$$\mathbf{y}_{k,b} = \mathbf{h}(\mathbf{p}_k^{b,k}) = \frac{1}{z} \begin{bmatrix} f_u & 0 & c_u & 0 \\ 0 & f_v & c_v & 0 \\ 0 & 0 & 0 & c_b f_u \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \tag{2.1}$$

and an inverse observation model, $\mathbf{g}(\cdot)$, that triangulates points seen in a stereo pair,

$$\mathbf{p}_k^{b,k} = \mathbf{g}(\mathbf{y}_{k,b}) = \frac{c_b}{d} \begin{bmatrix} u - c_u \\ \frac{f_u}{f_v}(v - c_v) \\ f_u \end{bmatrix}, \tag{2.2}$$

where c_u and c_v are horizontal and vertical optical centers in pixels, f_u and f_v are horizontal and vertical focal lengths in pixels, and c_b is the camera baseline in meters.

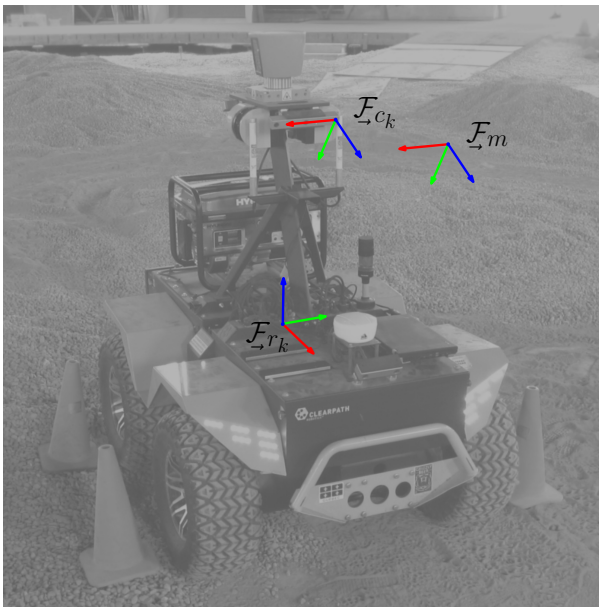


Figure 2.3: The two major frames used in this chapter are the camera frame, $\underrightarrow{\mathcal{F}}_{c_k}$, and robot control frame, $\underrightarrow{\mathcal{F}}_{r_k}$, that are related by a known and often fixed transform, $\mathbf{T}_{c_k,r_k}$. The camera frame is defined by the left camera while the control frame is the frame about which process models will be defined for controllers presented in this thesis. The submap frame, $\underrightarrow{\mathcal{F}}_{m}$, is only used for derivation purposes but can be generally thought of as a camera frame from a previous time-step.

2.2.2 Feature Tracking

In general, the VO and localization blocks in Figure 2.1 compute a transformation between the camera frame at time k, $\underrightarrow{\mathcal{F}}_{c_k}$, and a submap frame, $\underrightarrow{\mathcal{F}}_{m}$, based on a set of tracked features (Figures 2.3 and 2.4). A submap is a collection of key points where each key point contains: (i) $\mathbf{q}_m^{a,m} = (x, y, z)$, the position of feature a with respect to and expressed in $\underrightarrow{\mathcal{F}}_{m}$, and (ii) $\mathbf{v}_a$, the SURF descriptor associated with feature a. Descriptor-based tracking between the live view and submap is done by identifying nearest neighbours in descriptor space. Outlier candidate tracks are rejected based on an implementation of Random Sample and Consensus (Nistér, 2005). Finally, the pose of the camera relative to $\underrightarrow{\mathcal{F}}_{m}$ is computed by minimizing a cost function weighting the reprojection error of inlier features. Considering the nth inlying feature track representing a match between key point a (submap) and key point b (latest stereo pair), the reprojection error, $\mathbf{e}_n$, is defined as

$$\mathbf{e}_n = \mathbf{y}_{k,b} - \mathbf{h}\big(\mathbf{C}_{k,m}(\mathbf{q}_m^{a,m} - \boldsymbol{\rho}_m^{k,m})\big), \tag{2.3}$$

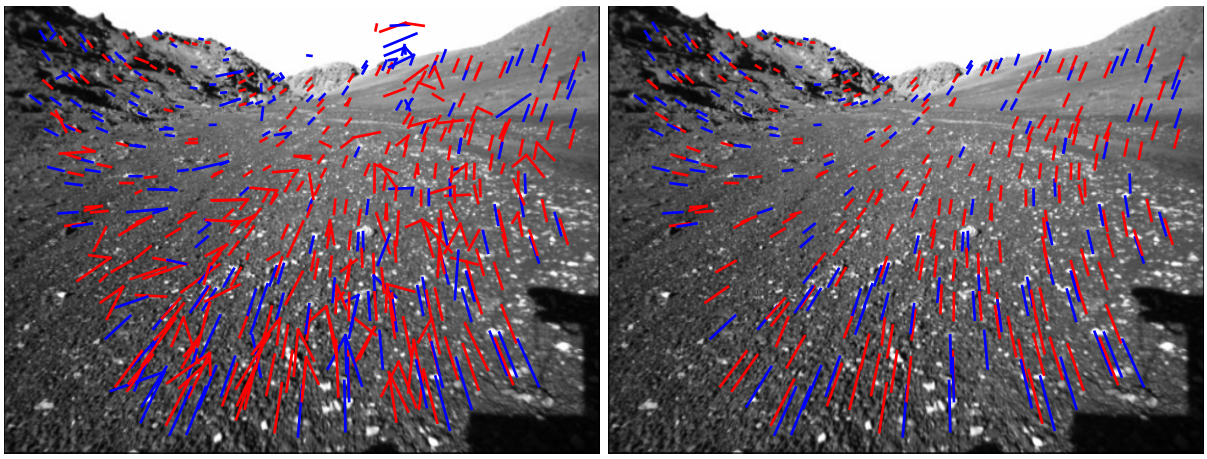


Figure 2.4: Example of key points tracked from a previous stereo image before (left) and after (right) outlier rejection. For VO, tracked features are used to estimate camera motion relative to the previous camera frame. For localization, tracked features are used to estimate the camera motion relative to the nearest key frame. Image credit: Furgale (2011).

where $\mathbf{C}_{k,m} \in \mathbb{R}^{3\times3}$ and $\boldsymbol{\rho}_m^{k,m} = (x, y, z)$ represent the camera rotation and translation, respectively, with respect to $\underrightarrow{\mathcal{F}}_m$. The cost function is defined as

$$J = \sum_{n=1}^{M} \mathbf{e}_n^T \mathbf{W}_n \, \mathbf{e}_n, \tag{2.4}$$

where M is the number of inlier feature tracks, and $\mathbf{W}_n$ is a weighting matrix based on the inverse of the estimated measurement covariance of $\mathbf{y}_{k,b}$. Specifically, (2.4) is minimized with respect to $\mathbf{C}_{k,m}$ and $\boldsymbol{\rho}_m^{k,m}$ using the Gauss-Newton method. Finally, the algorithm produces an estimate of the camera's pose in the submap frame,

$$\mathbf{T}_{k,m} = \begin{bmatrix} \mathbf{C}_{k,m} & -\mathbf{C}_{k,m}\,\boldsymbol{\rho}_m^{k,m} \\ \mathbf{0}^T & 1 \end{bmatrix}. \tag{2.5}$$

For VO, the submap is based on the previous camera frame, $\underrightarrow{\mathcal{F}}_{c_{k-1}}$, and associated stereo features, thus the computed transformation is the camera's pose relative to the camera frame at the previous time-step, $\mathbf{T}_{c_k,c_{k-1}} = \mathbf{T}_{c_k,m}$. For localization, the submap is based on key frame i, $\underrightarrow{\mathcal{F}}_{c_i}$, and the associated submap features, thus the computed transformation is the camera's pose relative to the ith key frame, $\mathbf{T}_{c_k,c_i} = \mathbf{T}_{c_k,m}$.

2.2.3 Map Building

Considering Sections 2.2.1 and 2.2.2, we now have the two main tools to generate a topometric pose graph composed of a set of vertices related by relative transformations, each containing a feature-based submap. Specifically, an initial key frame, $\underrightarrow{\mathcal{F}}_{c_i}$, where $i = 0$, is defined as the initial camera pose and the corresponding submap is initialized with the key point list at time $k = 0$, $\{\mathbf{y}_{0,j}, \mathbf{d}_{0,j}\}$. The robot is then manually piloted along the desired path. In subsequent time-steps, the camera's pose is computed by VO relative to the most recent key frame by a series of transformations, $\mathbf{T}_{c_k,c_i} = \mathbf{T}_{c_k,c_{k-1}}\mathbf{T}_{c_{k-1},c_i}$. New tracked features are added to the ith key frame submap until the camera motion exceeds a threshold on either translation or rotation. At this point, a new key frame is defined, $\underrightarrow{\mathcal{F}}_{c_{i+1}}$, where $\mathbf{T}_{c_{i+1},c_i} = \mathbf{T}_{c_k,c_i}$, and the submap for vertex $i+1$ is initialized with the key point list generated at time k. It is also assumed that there exists a robot control frame, $\underrightarrow{\mathcal{F}}_{r_k}$, where the transformation between the camera and robot frame is given by the known transform, $\mathbf{T}_{c_k,r_k}$. As a result, the algorithm also defines robot key frames, $\underrightarrow{\mathcal{F}}_{r_i}$, where $\mathbf{T}_{r_{i+1},r_i} = \mathbf{T}^{-1}_{c_k,r_k}\,\mathbf{T}_{c_{i+1},c_i}\,\mathbf{T}_{c_k,r_k}$.

2.2.4 Repeat Phase Localization

State estimation at every time-step in the repeat phase is accomplished using a combination of VO and localization. The algorithm first uses VO to propagate the pose estimate relative to the previous time-step and previous closest path vertex, $\mathbf{T}_{c_k,c_i} = \mathbf{T}_{c_k,c_{k-1}}\mathbf{T}_{c_{k-1},c_i}$. The algorithm then searches for the closest key frame by euclidean distance and refines the state estimate considering the associated submap. In the case that refinement fails (i.e., due to too few feature matches), the algorithm discards the 'refined' estimate and proceeds to the next time-step using the VO-based state estimate in a dead-reckoning fashion. At the start of a repeat, the algorithm performs a localization search to find its position relative to the closest vertex in the pose graph.

The final output at time k is a 2D estimate of the current robot pose (Figure 2.5), $\hat{\mathbf{x}}_k = (x, y, \theta)$, and a sequence of 2D desired poses, $\mathbf{x}_d = (\ldots, \mathbf{x}_{d,i-1}, \mathbf{x}_{d,i}, \mathbf{x}_{d,i+1}, \ldots)$, all relative to the nearest robot key frame, $\underrightarrow{\mathcal{F}}_{r_i}$. Specifically, 2D desired poses, $\mathbf{x}_{d,i+j}$, are formed based on $\mathbf{T}_{r_{i+j},r_i}$ by extracting the (x, y) coordinates from $\boldsymbol{\rho}^{i+j,i}_{r_i}$, and the yaw coordinate, θ, from the yaw-pitch-roll Euler-angle sequence extracted from $\mathbf{C}_{r_{i+j},r_i}$. The estimated robot pose is similarly extracted from $\mathbf{T}_{r_k,r_i}$, where $\mathbf{T}_{r_k,r_i} = \mathbf{T}^{-1}_{c_k,r_k}\,\mathbf{T}_{c_k,c_i}\,\mathbf{T}_{c_k,r_k}$.

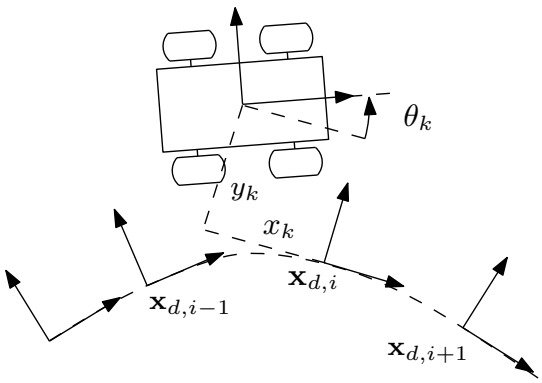


Figure 2.5: At every time-step, the vision-based localization algorithm provides the 2D pose of the robot, $\hat{\mathbf{x}}_k = (x, y, \theta)$, relative to the nearest path vertex, $\mathbf{x}_{d,i}$.

2.3 Conclusion

In summary, we have presented a brief overview of the real-time, vision-based localization algorithm that provides state estimates for all path-tracking controllers presented in this thesis. The algorithm operates in two phases: (i) the visual teach phase, where localization relative to the start of the path is obtained using VO, and (ii) the repeat phase, where localization relative to the nearest path vertex is obtained by relocalizing against stored key points. During an experiment to quantify relocalization accuracy, the authors compared the lateral path-tracking error estimated by the algorithm to that measured by Differential-GPS (Thales DG-16), revealing a lateral estimation error with mean -0.3 cm and standard deviation 2.9 cm. However, the ability of the VT&R algorithm to match features depends on the perspective of the camera during repeat passes. As a result, one motivation for the learning-based algorithms presented in this thesis is the ability to improve localization reliability through reductions in path-tracking errors and thus camera perspective errors.

Chapter 3

Iterative Learning Control

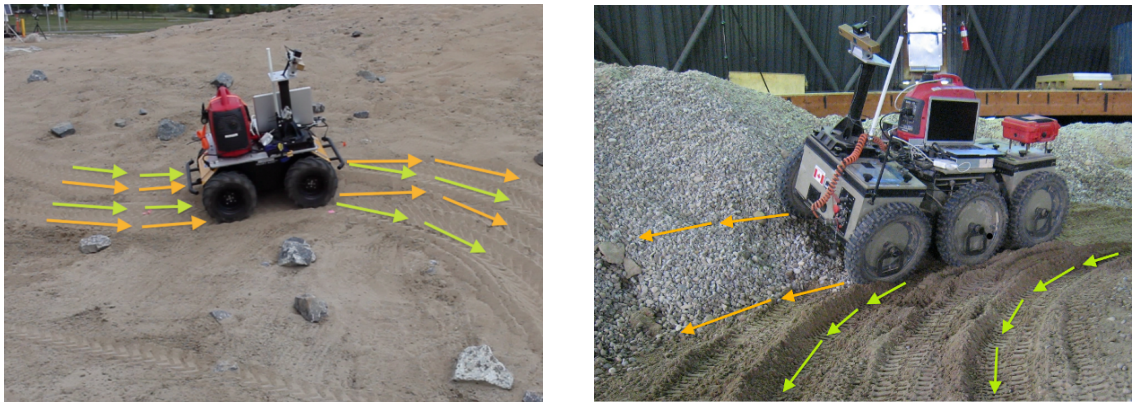
3.1 Introduction

In this chapter, we present ILC as an added-benefit, feedforward control for a practical autonomous mobile robot[1]. ILC was first introduced in the literature by Arimoto et al. (1984) to give systems that operate repetitively the ability to take advantage of knowledge gained during previous iterations. Here, we seek to apply ILC to a path-repeating mobile robot. There exist few references in the literature of applications of ILC on autonomous mobile robots and none, to our knowledge, demonstrating operation in challenging off-road paths with unknown terrain and vehicle dynamics (Figure 3.1).

ILC constructs an acausal feedforward signal that reduces control error over sequential iterations using error information from any previous trial. The algorithm generally requires little online computation since the feedforward signal is computed offline. As a result, ILC has been commonly applied to mass-production, industrial systems with low computational power (Bristow et al., 2006). In a few cases, it has also been demonstrated on quadrotors (Schoellig et al., 2012; Hehn and D'Andrea, 2014). In our work, the low computation time during operation is attractive since it enables real-time control and operation over long-distance paths involving potentially large datasets.

Recent sources for ILC literature include survey papers by Bristow et al. (2006) and Ahn et al. (2007). In the review by Bristow et al., ILC algorithms can be separated into four main groups: (i) Proportional-Derivative (PD), (ii) model inversion, (iii) H_∞, and (iv) optimal (quadratic) algorithms. PD-type algorithms are a tunable design and construct the feedforward signal using proportional and derivative gains on

[1] Associated video at http://tiny.cc/RobotLearnsIteratively



(a) Husky A200 robot tracking a sandy path including a 5-15° side slope.

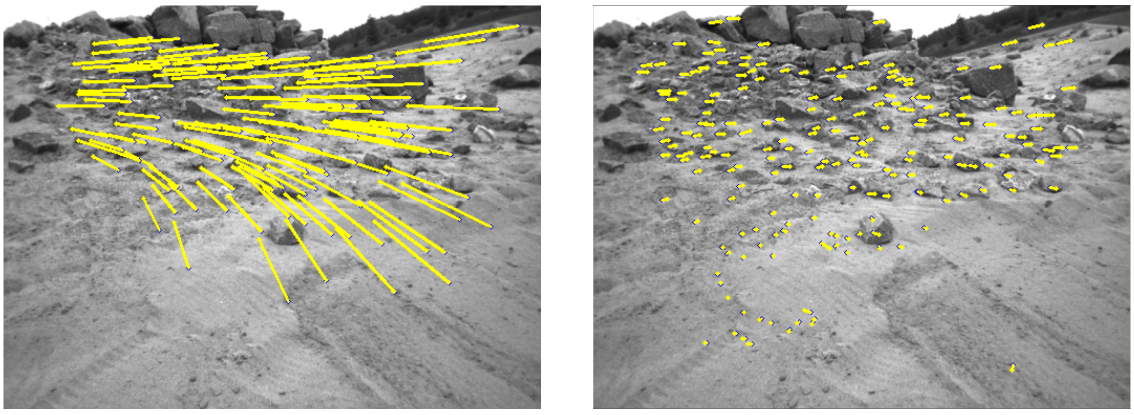
(b) ROC6 robot tracking a path at three times the nominal safe speed of 0.35 m/s.

Figure 3.1: Experimenting with ILC for mobile robots operating in outdoor environments. Here we show resulting paths with ILC enabled and disabled. The orange arrows show the current trajectory of the robots as they drive with ILC disabled. The green arrows show the path with reduced errors from when the robots were travelling with ILC enabled.

error signals collected in previous trials (Arimoto et al., 1984; Chen and Moore, 2002b). Model inversion algorithms use inverted plant dynamics as the learning function and thus converge rapidly in few trials at the cost of extensive modelling (Ghosh and Paden, 2002). H_∞ algorithms incorporate model uncertainty to ensure robust convergence but can be conservative (De Roover and Bosgra, 2000). Finally, quadratic algorithms compute optimal feedforward signals, balancing control inputs and tracking errors (Gunnarsson and Norrlöf, 2001; Schoellig et al., 2012).

In previous work on mobile robots, Oriolo et al. (1998) and Han and Lee (2011) demonstrated PD-ILC on short, indoor paths less than 4 m in length using wheel odometry for localization. Chen and Moore (2002a) demonstrated PD-ILC for mobile robots in simulation. In this work, we also experiment with PD-ILC. However, our implementation is founded on VT&R, a vision-based, on-board localization system that enables us to test on long paths in challenging outdoor terrain (Figure 3.2). As a result, our PD-ILC algorithm is required to compensate for unmodelled effects such as wheel slip, terrain topography, and robot dynamics. First, we show results from the repetition of a 40-m-long, outdoor path with sand and side-slopes by a 50 kg four-wheeled robot. Then we show the results from the repetition of a 50-m-long path by a 150 kg six-wheeled robot learning to drive quickly.

Our PD-ILC algorithm is implemented in parallel to a feedback-linearized controller, as in the work of Kang et al. (2005). The addition of the feedback signal allows for path



(a) Image taken during the first trial of experiment 1 with relatively high path-tracking error.

(b) Image taken during the sixth trial of experiment 1 with low path-tracking error.

Figure 3.2: Visual representations of relocalization in our VT&R framework. Each feature track represents the translation between a feature identified during the teach phase and re-identified during a repeat phase. In general, high path-tracking errors reduce the number of matched features and the resulting localization reliability.

completion during the first trial (i.e., prior to learning). This is important in reducing the training time required for a learning-based algorithm. Furthermore, the feedback signal also compensates for nonrepeatable disturbances in later trials. For example, wheel ruts and gravel paths evolve over multiple trials and thus cannot in general be completely compensated for by the feedforward signal. The feedback-linearized controller is selected due to its low computational complexity and its ability to handle nonholonomic constraints.

Finally, we propose a version of ILC where the feedforward control signal is parameterized by path length as opposed to time. In previous work on mobile robots, altering the scheduled speed on successive trials is non-trivial as the feedforward ILC signal is parameterized by time. In our work, we are able to schedule increasing speeds as tracking errors are reduced over sequential trials without reindexing feedforward signals since we assume the feedforward commands are place-specific. As a result, our robots begin experiments at a safe low speed and are capable of iteratively learning to drive faster.

3.2 Mathematical Formulation

The path-tracking controller is the sum of a feedback-linearized control signal and a feedforward ILC signal (Figure 3.3). In the following, we present the two controllers.

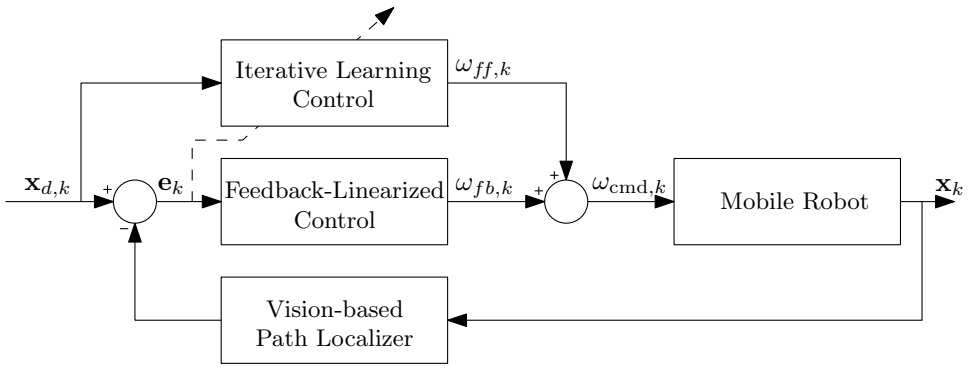


Figure 3.3: The ILC algorithm is in parallel to the FL controller and learns only from path-tracking errors. The dashed line indicates that the tracking errors are used to update the feedforward signal.

3.2.1 Feedback-Linearized Control

We use Feedback Linearization as described by Samson and Ait-Abderrahim (1991) as our base path-tracking controller. We model our robots as unicycles with state, $\mathbf{x}_k = (x_k, y_k, \theta_k)$, and inputs, $v_{\text{cmd},k}$ and $\omega_{\text{cmd},k} = \omega_{ff,k} + \omega_{fb,k}$, representing the linear and angular velocities, respectively, at time k (Figure 3.4). At every time-step, the VT&R localization algorithm provides the position of the robot, $\mathbf{x}_k$, relative to the nearest desired pose by Euclidean distance. We also define the lateral and heading path-tracking errors, $e_{L,k} = y_{d,k} - y_k$ and $e_{H,k} = \theta_{d,k} - \theta_k$. Assuming for now that the feedforward command, $\omega_{ff,k}$, is zero and defining the time between control signals as Δt, we find

$$\begin{bmatrix} e_{L,k+1} \\ e_{H,k+1} \end{bmatrix} = \begin{bmatrix} e_{L,k} \\ e_{H,k} \end{bmatrix} + \Delta t \begin{bmatrix} v_k \sin e_{H,k} \\ \omega_{fb,k} \end{bmatrix}. \tag{3.1}$$

Assuming the robot forward velocity is constant and letting $\mathbf{p}_k = \big(e_{L,k}, v_k \sin e_{H,k}\big)$, a new system of equations is given by

$$\mathbf{p}_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \mathbf{p}_k + \Delta t \begin{bmatrix} 0 \\ v_k \cos e_{H,k} \end{bmatrix} \omega_{fb,k}, \tag{3.2}$$

$$= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \mathbf{p}_k + \Delta t \begin{bmatrix} 0 \\ 1 \end{bmatrix} \eta_k, \tag{3.3}$$

where $\eta_k = v_k \cos(e_{H,k})\, \omega_{fb,k}$. Equation 3.3 represents a linear system with state $\mathbf{p}_k$ and input η_k. Now by setting $\eta_k = -\boldsymbol{\gamma}^T \mathbf{p}$, where $\boldsymbol{\gamma} = (\gamma_1, \gamma_2)$ with $\gamma_1, \gamma_2 > 0$, we find the

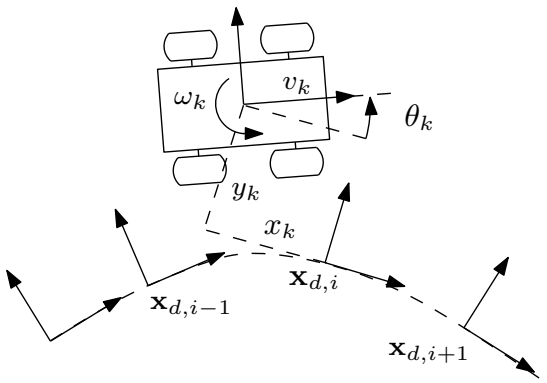


Figure 3.4: Definition of robot velocities, v_k and ω_k, and three pose variables, x_k, y_k, and θ_k. At each time-step, the VT&R algorithm provides an estimate of the robot position relative to the nearest desired pose by Euclidean distance. (ref. pages 21, 33, 54, 79).

stable, closed-loop system

$$\mathbf{p}_{k+1} = \begin{bmatrix} 1 & \Delta t \\ -\Delta t\gamma_1 & 1 - \Delta t\gamma_2 \end{bmatrix} \mathbf{p}_k. \tag{3.4}$$

Solving for the feedback control input, $\omega_{fb,k}$, given the two definitions of η_k, we find

$$\omega_{fb,k} = \frac{-\gamma_1 e_{L,k} - \gamma_2 v_k \sin e_{H,k}}{v_k \cos e_{H,k}}. \tag{3.5}$$

With suitable tuning values and paths, this control input without feedforward in combination with VT&R vision-based localization has been demonstrated to follow paths up to 3.2 km in length autonomously (Furgale and Barfoot, 2010) at speeds up to 0.35 m/s.

3.2.2 Added-benefit Iterative Learning Control

In practice, increasing the forward velocity or travelling on paths with sand or side-slopes can result in increased path-tracking errors that may put the robot in danger and challenge the ability of VT&R to relocalize. As shown in Figure 3.3, ILC learns a feedforward control input, $\omega_{ff,k}$, from these path-tracking errors. The ILC algorithm is able to generate feedforward commands that anticipate and preemptively respond to repeated disturbances and unmodelled robot dynamics since it has access to the entire sequence of errors along the path from previous trials. Intuitively, the learned, feedforward signal amounts to steering corrections along the path.

Unlike traditional ILC algorithms where feedforward commands are a function of time, our feedforward commands are a function of vertex indices, representing the distance along the path. This allows the algorithm to track the desired path at a safe speed during the first trial, and travel at increased speed during later trials using place-specific feedforward commands. We introduce i as an index for variables occurring at the ith path vertex and denote the time between vertex i and vertex $i{+}1$ as Δt_i. Now assuming a non-zero feedforward command, ω_{ff,k_i}, we approximate (3.4) as

$$\mathbf{p}_{i+1} = \begin{bmatrix} 1 & \Delta t_i \\ -\Delta t_i \gamma_1 & 1 - \Delta t_i \gamma_2 \end{bmatrix} \mathbf{p}_i + \Delta t_i \begin{bmatrix} 0 \\ v_i \cos e_{H,i} \end{bmatrix} \omega_{ff,i} \tag{3.6}$$

$$= \mathbf{A}_i \mathbf{p}_i + \mathbf{B}_i \omega_{ff,i}. \tag{3.7}$$

Considering there are N vertices in our path, we can produce N relationships,

$$\begin{aligned} \mathbf{p}_1 &= \mathbf{A}_0 \mathbf{p}_0 + \mathbf{B}_0 \, \omega_{ff,0}, \\ \mathbf{p}_2 &= \mathbf{A}_1 \mathbf{p}_1 + \mathbf{B}_1 \, \omega_{ff,1} \\ &= \mathbf{A}_1 \mathbf{A}_0 \mathbf{p}_0 + \mathbf{A}_1 \mathbf{B}_0 \, \omega_{ff,0} + \mathbf{B}_1 \omega_{ff,1}, \\ &\vdots \end{aligned}$$

that can be organized to produce the 'lifted form' (Bristow et al., 2006) for trial j,

$$\mathbf{p}^{(j)} = \mathbf{p}_{\text{init}} + \mathbf{B}^{(j)} \boldsymbol{\omega}_{ff}^{(j)}, \tag{3.8}$$

where

$$\mathbf{p}^{(j)} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_N \end{bmatrix}, \quad \mathbf{p}_{\text{init}} = \begin{bmatrix} \mathbf{A}_0 \mathbf{p}_0 \\ \mathbf{A}_{1:0} \mathbf{p}_0 \\ \vdots \\ \mathbf{A}_{N-1:0} \mathbf{p}_0 \end{bmatrix}, \quad \boldsymbol{\omega}_{ff}^{(j)} = \begin{bmatrix} \omega_{ff,0} \\ \omega_{ff,1} \\ \vdots \\ \omega_{ff,N-1} \end{bmatrix},$$

$$\mathbf{B}^{(j)} = \begin{bmatrix} \mathbf{B}_0 & 0 & 0 & 0 & \dots & 0 \\ \mathbf{A}_1 \mathbf{B}_0 & \mathbf{B}_1 & 0 & 0 & \dots & 0 \\ \mathbf{A}_{2:1} \mathbf{B}_0 & \mathbf{A}_2 \mathbf{B}_1 & \mathbf{B}_2 & 0 & \dots & 0 \\ \vdots & \vdots & & & \ddots & \\ \mathbf{A}_{N-1:1} \mathbf{B}_0 & \mathbf{A}_{N-1:2} \mathbf{B}_1 & & & \dots & \mathbf{B}_{N-1} \end{bmatrix},$$

and $\mathbf{A}_{a:b} = \mathbf{A}_a \, \mathbf{A}_{a-1} \ldots \mathbf{A}_b$. Once in lifted form, we can introduce the error for trial j,

$$\mathbf{e}^{(j)} = \mathbf{p}_d - \mathbf{p}^{(j)}, \tag{3.9}$$

where $\mathbf{p}_d$ is the desired state and is identically zero. Then

$$\mathbf{e}^{(j)} = -\mathbf{p}_{\text{init}} - \mathbf{B}^{(j)} \boldsymbol{\omega}_{ff}^{(j)}, \tag{3.10}$$

and

$$\begin{aligned} \mathbf{e}^{(j+1)} - \mathbf{e}^{(j)} &= -\mathbf{B}^{(j+1)} \boldsymbol{\omega}_{ff}^{(j+1)} + \mathbf{B}^{(j)} \boldsymbol{\omega}_{ff}^{(j)} \\ &\approx -\mathbf{B}^{(j)} \boldsymbol{\omega}_{ff}^{(j+1)} + \mathbf{B}^{(j)} \boldsymbol{\omega}_{ff}^{(j)}. \end{aligned} \tag{3.11}$$

To compute the feedforward control input for the next trial we use a proportional-type ILC controller of the form

$$\boldsymbol{\omega}_{ff}^{(j+1)} = \gamma_u \boldsymbol{\omega}_{ff}^{(j)} + \boldsymbol{\Gamma}_e \mathbf{e}^{(j)}, \tag{3.12}$$

with forgetting factor, γ_u, and learning gain, $\boldsymbol{\Gamma}_e$. As a result, we have that

$$\begin{aligned} \mathbf{e}^{(j+1)} &= \mathbf{e}^{(j)} - \gamma_u \mathbf{B}^{(j)} \boldsymbol{\omega}_{ff}^{(j)} - \mathbf{B}^{(j)} \boldsymbol{\Gamma}_e \mathbf{e}^{(j)} + \mathbf{B}^{(j)} \boldsymbol{\omega}_{ff}^{(j)} \\ &= \mathbf{e}^{(j)} + \gamma_u (\mathbf{e}^{(j)} + \mathbf{p}_{\text{init}}) - \mathbf{B}^{(j)} \boldsymbol{\Gamma}_e \mathbf{e}^{(j)} - (\mathbf{e}^{(j)} + \mathbf{p}_{\text{init}}) \\ &= (\gamma_u - \mathbf{B}^{(j)} \boldsymbol{\Gamma}_e) \mathbf{e}^{(j)} - (1 - \gamma_u) \mathbf{p}_{\text{init}}. \end{aligned} \tag{3.13}$$

Stabilization in the trial domain amounts to selecting γ_u and $\boldsymbol{\Gamma}_e$ such that the eigenvalues of $(\gamma_u - \mathbf{B}^{(j)} \boldsymbol{\Gamma}_e)$ are stable. Considering $\mathbf{B}^{(j)}$, we used the update law

$$\omega_{ff,i}^{(j+1)} = \gamma_u \omega_{ff,i}^{(j)} + \frac{-\gamma_L \epsilon_{L,i} - \gamma_H \sin \epsilon_{H,i}}{\cos \epsilon_{H,k_i}}, \tag{3.14}$$

where $\epsilon_{L,i}$ and $\epsilon_{H,i}$ are lookahead errors,

$$\epsilon_{L,i} = \frac{1}{\kappa} \sum_{m=i+1}^{i+\kappa} e_{L,m}, \quad \epsilon_{H,i} = \frac{1}{\kappa} \sum_{m=i+1}^{i+\kappa} e_{H,m}, \tag{3.15}$$

and κ is the number of vertices to lookahead. In (3.14), the gains γ_L and γ_H set the rate of convergence while the forgetting factor γ_u gives the system the ability to forget nonrepetitive disturbances.

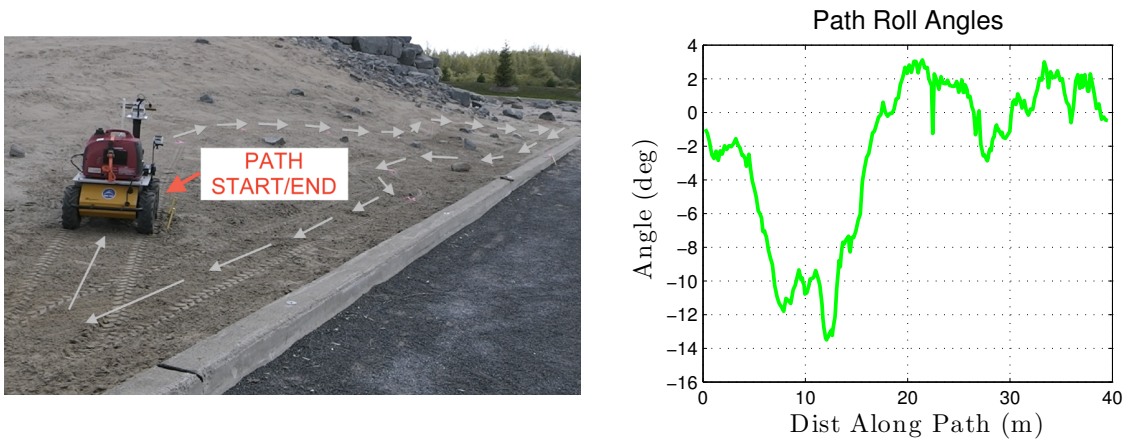


Figure 3.5: Experiment 1 test path overview (left) and side-slope (right) at the Canadian Space Agency's Mars Emulation Terrain in Longueuil, Québec, Canada. The terrain offers a large selection of surface topographies intended for robot mobility testing. The path selected included a steep side-slope around 10-15 m into the path. Another photo of the side-slope section is shown in Figure 3.1.

3.2.3 Dealing with Initial Conditions

Since the initial conditions of each trial were set by the conclusion of a previous trial, we could not guarantee identical initial conditions for each trial. As such, we followed the approach of Freeman et al. (2011) and smoothly modified the desired path to create a zero-error initial condition for the first m_{ic} vertices of each trial. For trial j, this is:

$$\mathbf{p}'_{d,i} = \begin{cases} \mathbf{p}_{d,i} - \frac{(m_{ic}-i)}{(m_{ic})}(\mathbf{p}_{d,i} - \mathbf{p}_0^{(j)}), & i < m_{ic} \\ \mathbf{p}_{d,i}, & i \geq m_{ic}. \end{cases} \tag{3.16}$$

3.3 Experimental Results

3.3.1 Overview

We tested the ILC algorithm in two different experiments. The first test was conducted using a 50 kg Husky robot travelling along a 40-m-long sandy path including an unmodelled side-slope. The side-slope test was conducted on the Canadian Space Agency's Mars Emulation Terrain in a sandy section with slopes of between 5-15° as shown in Figure 3.5. The second test was conducted using a 160 kg ROC6 robot sequentially navigating a 50-m-long path with speeds gradually increased to three times the speed for

which the feedback-linearized controller was tuned. The speed test was conducted in the University of Toronto Institute for Aerospace Studies (UTIAS) MarsDome along a path through tight valleys and over a combination of gravel and sandy surfaces.

In both cases, the controller described in Section 3.2 was implemented and run in addition to the VT&R software (Chapter 2) on a MacBook Pro with an Intel 2.4 Ghz Core2Duo processor and 4 GB of RAM. The input camera in both experiments was a Point Grey Bumblebee XB3 stereo camera. The resulting real-time localization and path-tracking control signals were generated at approximately 10 Hz. Since GPS was not available, the improvement due to the ILC feedforward control signal was quantified by the localization of the VT&R algorithm.

3.3.2 Experiment 1: Learning Kinematics

In the first experiment, the ILC algorithm successfully reduced the maximum lateral and heading errors by factors of roughly three after six trials (Figures 3.6 and 3.7). For demonstration purposes, γ_u was set to 1.0, encouraging quick learning. However, the rate of convergence was often delayed and influenced by the evolution of the environment in response to the robot's activity. For example, repeating the same path caused ruts to form and sand located on side-hills to shift thereby causing new unmodelled disturbances and delaying the reduction in path-tracking errors. In addition, it was found that the topography of the environment around the path resulted in changing disturbances from trial to trial. For example, if the path were along the ridge of a hill, the disturbances caused by the side-slope decreased from trial to trial as the robot path approached the ridge of the hill. While the ILC algorithm inherently continued to adjust its feedforward

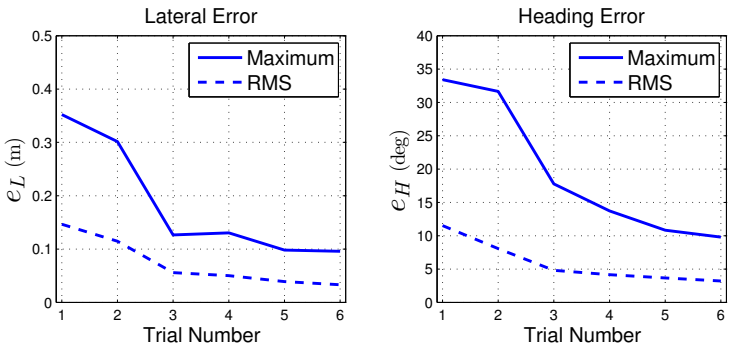


Figure 3.6: Lateral and heading error versus trial number in experiment 1. The maximum and Root-Mean-Square (RMS) errors are reduced significantly within the first few trials.

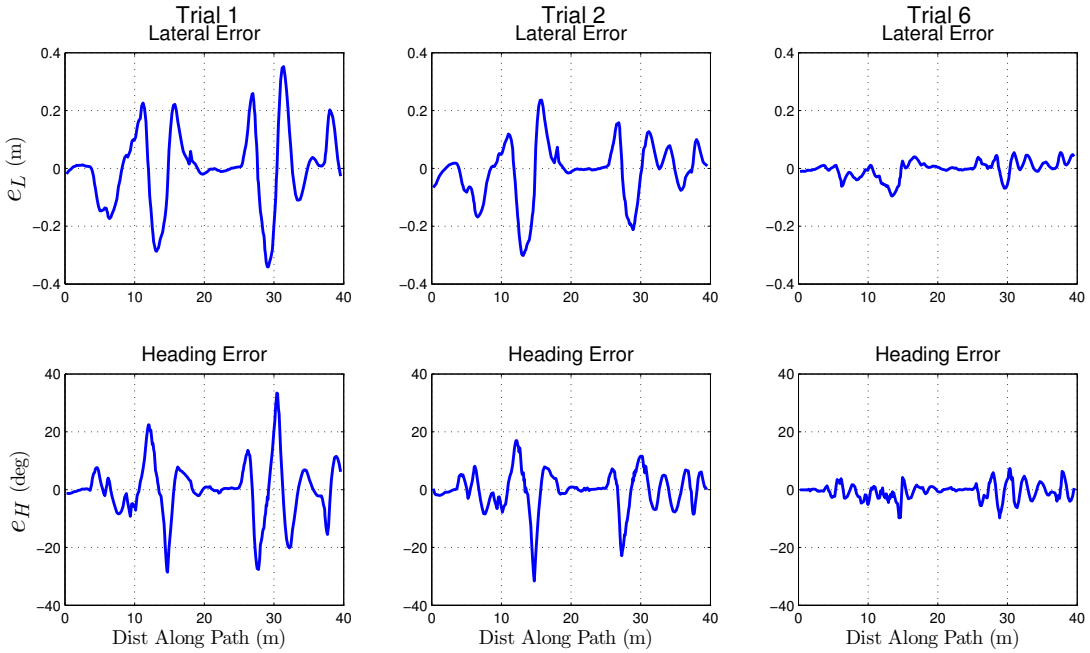


Figure 3.7: The ILC algorithm was able to quickly reduce the maximum lateral and heading errors but had a greater challenge eliminating the errors completely.

signal, in practice, γ_u should be set to less than 1.0 to give the system a greater ability to forget non-repeating disturbances and noise. Finally, in addition to posing a physical danger a robot, high path-tracking errors also cause reductions in vision-based localization reliability due to perspective shift. This can be seen visually in Figure 3.2, where we show examples of relocalization at the same point along the desired path in trials 1 and 6.

3.3.3 Experiment 2: Learning Dynamics

In the second experiment, the ILC algorithm successfully mitigated the effects of increased repeat speed over the course of ten trials in the UTIAS MarsDome (Figure 3.8). As can be seen in Figure 3.9, ILC was used to gradually learn to drive at 1.0 m/s by sequentially increasing the repeat speed each trial. After ten trials, the ILC algorithm had reduced the lateral and heading errors by factors of roughly three compared to the errors when driving at 1.0 m/s without the ILC algorithm. Furthermore, when driving at 1.0 m/s without learning, two manual interventions were required to bring the robot back on the path from locations on the sides of the valleys. Of note, without any other algorithm development, the process of learning to drive at a new speed implies forgetting the old. Interestingly,

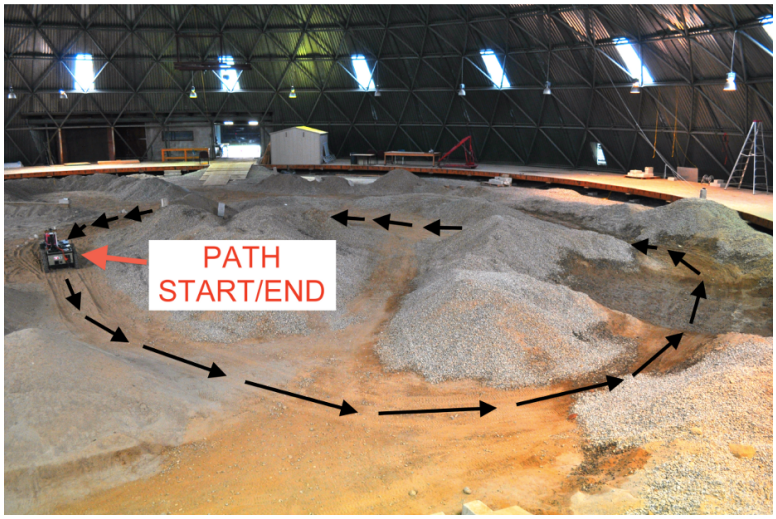


Figure 3.8: Experiment 2: ROC6 at the start of the path in the MarsDome facility in Toronto, Ontario, Canada. The facility provides a network of sandy and gravel floor valleys defined by a set of impassable gravel hills.

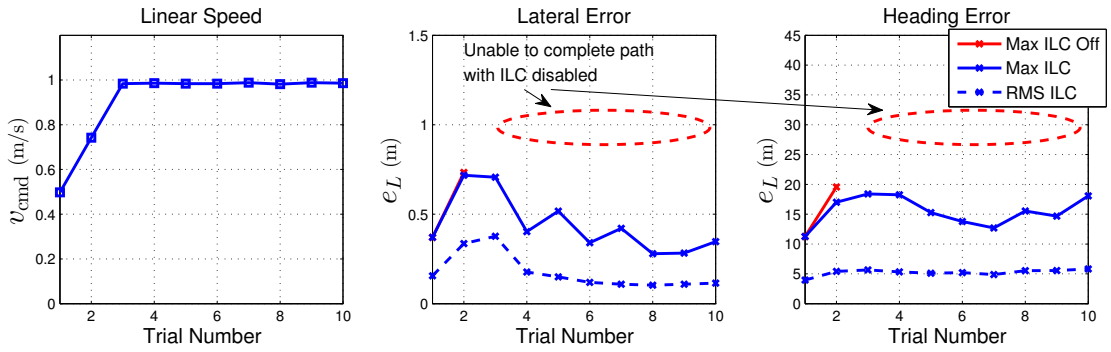


Figure 3.9: Even though the ROC6 mass was 110 kg greater than that of the Husky, using the same ILC tuning parameters as experiment 1 resulted in decent performance in experiment 2. With ILC disabled, the robot was unable to complete the path at 1.0 m/s.

γ_L and γ_H were identical to those for the Husky suggesting the ILC learning gain has some independence of platform and path. Overall, the choice to produce a feedforward signal indexed by vertex number and thus distance along the path, as opposed to time, was largely made to facilitate changes in robot speed and to conform to the vertex-based VT&R system. The result was computationally simple and effective at reducing tracking errors.

3.4 Discussion

In general, ILC offers an efficient and effective method of improving path-tracking through experience. Real-time inputs for mobile robots operating on long-distance paths are possible since the feedforward signal is computed offline. However, the algorithm learns only a single sequence of feedforward commands leading to two issues. Firstly, since the algorithm is only capable of maintaining a single sequence of commands, the algorithm is only applicable to a system operating on a single path with a single speed schedule. For example, we demonstrated the algorithm on a robot learning to drive quickly in the second experiment. Prior to each trial, the scheduled speed was increased relative to the previous trial until a desired max speed was achieved. However, if the desired speed were then reduced, the algorithm would need to relearn how to drive slowly. Moreover, it is not clear how to maintain a single sequence of feedforward signals for operation on a network of paths, where there would be many possible combinations of speeds and paths for any given trial. Secondly, the ILC algorithm requires the speed to be scheduled prior to operation since the feedforward signal is also computed prior to operation. As a result, ILC would not be suitable for operation in an environment with dynamic obstacles where speed is selected in real-time.

Our work on ILC with mobile robots led to the insight that in addition to providing real-time inputs over long-distance paths, learning-based algorithms for mobile robots should be capable of anytime learning and generalization. Ideally, the robot would be able to collect data anytime it moves (i.e., anytime learning) and maintain a database of such learned experiences. This would allow for the robot to collect a variety of experiences resulting from arbitrary initial conditions, terminal conditions, desired paths, and desired speeds. In addition to maintaining a database of learned experiences, the robot would then ideally be capable of generalizing from previous learned experiences. In this way, the learning-based controller would enable flexible and useful mobile robot operation in large-scale, outdoor environments. For example, this would allow for situations when the robot is forced to slow down due to dynamic obstacles or issues encountered during a trial. As a result, the learning-based algorithms presented in Chapters 5, 6, and 7 all enable anytime learning and generalization in addition to real-time inputs and operation on long-distance paths.

3.5 Conclusion

In summary, the contribution in this chapter is an added-benefit, proportional-type Iterative Learning Controller for a path-repeating, mobile robot negotiating off-road, challenging terrain. The feedforward ILC algorithm operates in parallel to a feedback-linearized controller. The feedback-linearized controller enables the robot to complete the path in the first trial and also compensates for non-repeating disturbances in later trials. The ILC algorithm produces a feedforward signal that acausally mitigates disturbances not modelled by the feedback controller. Further, we present a novel formulation whereby the ILC signal is spatially-indexed allowing the algorithm to learn to drive at higher speeds without needing to reindex the feedforward command sequence. Thus considering a single commanded linear speed, the resulting feedforward signals are assumed to be place-specific and are effectively indexed only by the distance along a desired path.

Two experiments, including over 700 m of travel, demonstrated the system's ability to handle unmodelled terrain and rover dynamics. During the first experiment, a four-wheeled 50 kg robot was taught a 40-m-long path including a 5-15° sandy side-slope. The ILC signal effectively reduced the lateral and heading errors by factors of three in the first six trials. During the second experiment, a six-wheeled 160 kg robot was taught a 50-m-long path with little margin for error across terrain with varying ground properties. ILC was used to gradually learn to drive at a speed three times the nominal repeat speed of 0.35 m/s. Compared to driving at 1.0 m/s without learning, the resulting lateral and heading path-tracking errors were reduced by a factor of three after ten trials.

Chapter 4

Experience-based Speed Scheduling

4.1 Introduction

This chapter presents a novel experience-based speed scheduler. In our work on ILC, the commanded linear speed was set manually prior to each trial. The problem of speed scheduling is generally addressed by a guidance algorithm as part of a planned trajectory. Speeds should be set appropriately in order to guarantee safe path-tracking and localization, in much the same way as speed limits on roads. The main issue is in predicting the performance of robot subsystems, such as vision-based localization systems, as a function of variables such as speed and robot state in order to identify a constraint-satisfying speed schedule *a priori*.

In this work, we present a speed scheduling algorithm that minimizes travel time and simultaneously guarantees feasibility of the trajectory despite unknown effects by incorporating experience from previous path traversals. For example, the speed scheduler uses localization experience to selectively increase the speed of sections of the path where localization reliability can be guaranteed. Motion blur at high speeds is one cause of reduced localization reliability and is very difficult to predict *a priori*. To our knowledge, this is the first algorithm to incorporate experience from previous path traversals when producing a speed schedule.

There are many speed scheduling approaches in the literature. For schedulers seeking minimum-time trajectories, the general approach is to identify limits on the robot speed and acceleration as a function of constraints, such as actuator limits, then plan a schedule to operate at the highest speed possible. As a result, most approaches differ primarily in which constraints are addressed.

Figure 4.1: Experiments are performed using a large Defence Research and Development Canada (DRDC) Mule Research Vehicle (DMRV) where the sole sensor used for localization is a Point Grey Bumblebee stereo camera (highlighted in red). The experience-based speed scheduler is able to identify speed limits considering requirements on path-tracking error limits and vision-based localization reliability.

Classic speed schedulers for mobile robots concentrate on generating smooth speed profiles prior to commencing path traversal while incorporating electromechanical constraints on speeds and accelerations (Fatouhi et al., 2002; Bianco, 2009). For example, Munoz et al. (1994) derive the maximum robot speed, acceleration, and deceleration from motor and brake system specifications. Prado et al. (2003, 2002) include speed and acceleration limits resulting from predicted motor temperature and battery power. Other schedulers predict lateral accelerations and friction forces considering weight transfers to constrain the speed based on slip and path-tracking error limits (Prado et al., 2003; Purwin and D'Andrea, 2006; Waheed and Fotouhi, 2009). In each of these papers, detailed models of the robot and environment are required to generate the appropriate speed or acceleration limits. In this work, in addition to simple *a priori* speed and acceleration limits, we use past observations of vision-system performance, path-tracking errors, and control inputs to iteratively identify speed and acceleration limits.

Other approaches schedule the speed while tracking a path, using experiences in real-time to identify speed limits. For example, the Stanley robot, designed for the 2006 DARPA Grand Challenge, slows down after encountering rough sections along a

planned path (Thrun et al., 2006). Path roughness is identified using measurements of the vertical acceleration of the vehicle (Brooks and Iagnemma, 2005). The approach is shown to decrease the damage to the robot caused by shock and vibration, thereby increasing the long-term system reliability. While the approach employed by Thrun et al. (2006) is designed for systems traversing a path for the first time, it nevertheless results in the robot experiencing serious shock and vibrations at the start of rough patches along a path. Our approach, on the other hand, schedules speed based on experience from previous path traversals. This gives our system the ability to slow down *before* encountering challenging sections of the path, rather than behaving reactively. Finally, in some speed schedulers, the speed is also determined in real-time in order to prevent collisions with dynamic obstacles (Kant and Zucker, 1986; Prado et al., 2003; Purwin and D'Andrea, 2006). In these cases, it is assumed that the dynamic obstacle is travelling across the path and therefore waiting a few moments will result in a clear path. An investigation into the sensing, control, and actuation requirements for real-time obstacle avoidance is presented by Kelly and Stentz (1998a,b). In this work, we assume the environment is free of dynamic obstacles and the planned path avoids static obstacles.

4.2 Mathematical Formulation

4.2.1 Overview

In this work, we consider the mobile robot to be tracking a desired trajectory consisting of a set of N desired poses (Figure 3.4),

$$\mathcal{P}_d = (\mathbf{x}_{d,1}, \ldots, \mathbf{x}_{d,N}),$$

where $\mathbf{x}_{d,i} = (x_{d,i}, y_{d,i}, \theta_{d,i})$, and N scheduled speeds during the jth trial,

$$\mathcal{V}^{(j)}_{\text{sched}} = (v^{(j)}_{\text{sched},1}, \ldots, v^{(j)}_{\text{sched},N}).$$

At time k, the VT&R algorithm produces an estimate of the robot's pose, $\mathbf{x}_k = (x_k, y_k, \theta_k)$, relative to the nearest path vertex by Euclidean distance. The path-tracking controller then sets the commanded linear velocity of the robot, $v_{\text{cmd},k}$, based on the scheduled speed at the nearest path vertex, $v_{\text{sched},i}$, and computes a commanded angular velocity, $\omega_{\text{cmd},k}$. In general, the controller aims to minimize path-tracking errors.

In this work, we present a novel method of producing speed schedules, $\mathcal{V}_{\text{sched}}^{(j+1)}$. In Section 4.2.2, we discuss the experience collected during trial j for use in computing schedules. In Sections 4.2.3 and 4.2.4, we present our method of proposing a speed schedule for trial $j+1$ based on experience and limiting it based on *a priori* speed and acceleration limits, respectively.

4.2.2 Collected Experience

During each trial, the robot drives the full path, gathering experience. Specifically, we collect vision-based localization, path-tracking, and control-input experience for use in speed scheduling. For consistency, experience is only stored as the robot passes each desired vertex.

Vision-based Localization Experience

When using vision-based localization systems, there exists a speed limit above which localization becomes unreliable and the safety of the robot can no longer be assured. This speed limit may come as a result of low light conditions, a degraded scene (relative to when the path was taught), large deviations from the path, or perhaps motion blur. Instead of trying to predict the effect of these conditions, we record the number of features matched by the VT&R system, $c_{\text{feat},i}^{(j)}$, when passing the ith vertex during the jth trial as an indicator of the conditions faced by the localization system,

$$\mathcal{C}_{\text{feat}}^{(j)} = (c_{\text{feat},1}^{(j)}, \ldots, c_{\text{feat},N}^{(j)}).$$

Intuitively, we make the assumption that there exists a relationship such that as the speed of the robot increases, the number of matched features at a given path vertex decreases. While this relationship is not known *a priori*, we use experience to judge whether the speed at the ith path vertex during the next trial can be increased further.

Path-tracking Experience

We also record the lateral and heading path-tracking errors, $e_{L,i}^{(j)}$ and $e_{H,i}^{(j)}$, respectively,

$$\begin{bmatrix} e_{L,i}^{(j)} \\ e_{H,i}^{(j)} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}_i, \tag{4.1}$$

when passing the ith vertex during the jth trial,

$$\mathcal{E}_L^{(j)} = (e_{L,1}^{(j)}, \ldots, e_{L,N}^{(j)}),$$
$$\mathcal{E}_H^{(j)} = (e_{H,1}^{(j)}, \ldots, e_{H,N}^{(j)}).$$

We do this for two reasons. First, we have assumed that the planned path is safe and free of obstacles. Therefore, it is important to maintain low path-tracking errors since we can only guarantee that the terrain near the planned path is free of obstacles. In the case that the path planner provides additional information about the lateral distance to obstacles, our speed scheduler could restrict the robot speed so as to ensure sufficiently low lateral path-tracking errors. Second, the vision system is sensitive to perspective changes between the teach pass and any repeat pass. Perspective changes are the direct result of path-tracking errors and reduce the reliability of the localization system. Prediction of path-tracking errors as a function of speed is challenging. Thus we use experience to judge the effect of speed on path-tracking errors.

Control-input Experience

Finally, the scheduled linear speed must also address constraints on angular velocities resulting from actuator limits. As a result, we record the commanded angular velocity, $\omega_{\text{cmd},i}^{(j)}$, when passing the ith vertex during the jth trial,

$$\Omega^{(j)} = (\omega_{\text{cmd},1}^{(j)}, \ldots, \omega_{\text{cmd},N}^{(j)}).$$

In order to track the desired path at a velocity, v_k, the robot must be capable of turning at an angular velocity, ω_k. In order to achieve this actual angular velocity, the path-tracking controller commands a commanded angular velocity, $\omega_{\text{cmd},k}$, compensating for wheel slip, side slopes, and other model discrepancies through state feedback. As the commanded linear velocity is increased over trials, so too the commanded angular velocity will increase in order to track the path. In practice, detailed models of wheel slip including the effects of side-slopes and ground texture are often only available for specific situations. For example, Nagatani et al. (2013) present models for the case when a robot is required to traverse a straight path across a side-slope. By using experience, we are able to predict that the commanded angular velocities remain within the actuator limits when generating new speed schedules with no *a priori* knowledge of the terrain topography.

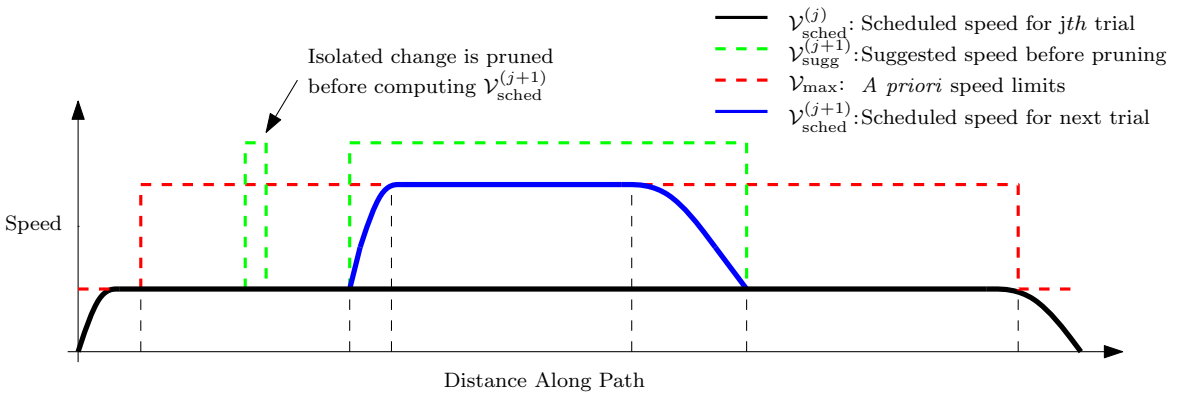


Figure 4.2: Experience-based speed scheduling occurs in two steps: (i) modifications to the previous speed schedule are suggested based on experience (green), (ii) the suggested speeds are constrained by *a priori* speed and acceleration constraints (blue).

4.2.3 Experience-based Speed Schedule Modification

After completing the desired path, the next step in the speed-scheduler algorithm is to suggest new desired speeds for the next trial based on the desired speeds during the jth trial and the collected experience described in Section 4.2.2. Specifically, if path-tracking errors are below (above) a threshold, control inputs are below (above) a threshold, and (or) there are a sufficient (insufficient) number of matched features in a certain section, the speed in that section is increased (decreased, respectively). Using tuned values for increasing and decreasing the scheduled speed, $\gamma_1 > 0$, $\gamma_2 > 0$, respectively, and thresholds, $\lambda_L \geq 0$, $\lambda_H \geq 0$, $\lambda_\omega > 0$, and $\lambda_{\text{feat}} \geq 3$, the scheduler follows rules to generate the suggested speeds for each path vertex:

$$v^{(j+1)}_{\text{sugg},i} = \begin{cases} v^{(j)}_{\text{sched},i} + \gamma_1 & \text{if } (|e^{(j)}_{L,i}| < \lambda_L) \wedge (|e^{(j)}_{H,i}| < \lambda_H) \wedge (|\omega^{(j)}_{\text{cmd},i}| < \lambda_\omega) \wedge (c^{(j)}_{\text{feat},i} > \lambda_{\text{feat}}), \\ v^{(j)}_{\text{sched},i} - \gamma_2 & \text{if } (|e^{(j)}_{L,i}| > \lambda_L \lambda_{\text{db}}) \vee (|e^{(j)}_{H,i}| > \lambda_H \lambda_{\text{db}}) \vee \\ & \quad (|\omega^{(j)}_{\text{cmd},i}| > \lambda_\omega \lambda_{\text{db}}) \vee (c^{(j)}_{\text{feat},i} < \lambda_{\text{feat}}/\lambda_{\text{db}}), \\ v^{(j)}_{\text{sched},i} & \text{otherwise.} \end{cases} \tag{4.2}$$

Effectively, the automated speed scheduler identifies sections of the path where the system can tolerate higher speeds and sections where it cannot, thus balancing the trade-off between speed, path-tracking errors, and vision-based localization reliability. We use $\lambda_{\text{db}} > 1$ to produce a deadband where the speed at a vertex is neither increased nor decreased. After generating suggested speeds for all path vertices in the next trial,

isolated increases are then pruned to encourage a smooth speed profile (Figure 4.2). An isolated increase occurs when too few path vertices in a section of the path are eligible for increased speeds. For the first trial, the scheduled speed at all vertices in the path is set to a fixed speed, $v^{(1)}_{\text{sched},i} = v_{\text{init}}$.

4.2.4 *A Priori* Speed and Acceleration Constraints

The final step in our experience-based speed scheduler is to incorporate *a priori* constraints on linear speed, acceleration, and deceleration. Firstly, the robot has a known actuator-based linear speed constraint. In addition, the robot must respect speed limits for safety, particularly during key sections of the path such as the start and end of the path. The combination of actuator and safety constraints results in a set of speed limits,

$$\mathcal{V}_{\max} = (v_{\max,1}, \ldots, v_{\max,N}),$$

as shown in Figure 4.2. These speed limits are applied to the suggested scheduled speed,

$$v^{(j+1)}_{\text{sugg},i} \leftarrow \min\left(v^{(j+1)}_{\text{sugg},i}, v_{\max,i}\right). \tag{4.3}$$

Finally, we limit the acceleration and deceleration to account for robot capability and safe operation. The scheduled speed must satisfy the acceleration and deceleration constraints, $a_{\max} > 0$ and $a_{\min} < 0$, respectively, at every path vertex,

$$\begin{aligned} v^{(j+1)}_{\text{sched},i} \leftarrow \min\Bigg(& v^{(j+1)}_{\text{sugg},i}, \\ & \sqrt{\left(v^{(j+1)}_{\text{sugg},i-1}\right)^2 + 2\, d_{i-1,i}\, a_{\max}}, \\ & \sqrt{\left(v^{(j+1)}_{\text{sugg},i+1}\right)^2 - 2\, d_{i,i+1}\, a_{\min}}\Bigg), \end{aligned} \tag{4.4}$$

where $d_{i,j}$ is the distance between two path vertices, $\mathbf{x}_{d,i}$ and $\mathbf{x}_{d,j}$,

$$d_{i,j} = \sqrt{(x_{d,j} - x_{d,i})^2 + (y_{d,j} - y_{d,i})^2}. \tag{4.5}$$

The resulting speed schedule satisfies all *a priori* speed and acceleration constraints, and also takes into account localization and path-tracking performance requirements.

4.3 Experimental Results

4.3.1 Overview

The speed-scheduling algorithm presented in Section 4.2 was tested in two experiments with 10 trials each, resulting in over 4 km of testing in outdoor environments. The test vehicle (Figure 4.1) was manually taught the two paths including sharp turns, a variety of slopes, and a variety of surfaces. Both paths were taught at the DRDC Experimental Proving Grounds in Suffield, Alberta, Canada. The resulting speed schedules varied in speed from 0.5 m/s to 2.0 m/s.

4.3.2 Tuning Parameters

The speed scheduler was set to maintain matched feature counts greater than 30, heading errors less than 10°, lateral errors less than 15 cm, and commanded angular velocities less than 1.0 rad/s. The speed scheduler increments, γ_1 and γ_2, were set to increase a scheduled speed by 0.2 m/s or decrease a scheduled speed by 0.24 m/s. The vehicle speed was limited to 0.4 m/s during the start and end segments, and 2.0 m/s otherwise. Finally, the vehicle acceleration was limited to 0.2 m/s^2 and the vehicle deceleration was limited to -0.05 m/s^2. Parameter settings were manually tuned in runs separate from the results section considering knowledge of the robot's sensor and actuator limits. The scheduling algorithm then maximized the speed along a path while respecting these limits.

4.3.3 Results

During the first experiment, the vehicle travelled the desired 100-m-long path (Figure 4.3) 10 times, resulting in 1 km of testing. During the first four trials, the number of matched features, path-tracking errors, and desired angular speeds were all within the specified limits. As a result, the scheduled speed was increased equally for all path vertices except those near the start and end of the path (Figure 4.4). In the fifth trial, the path-tracking error began limiting the scheduled speed at 50 m along the path (Figure 4.5). In general, the path-tracking errors coincided with the path slopes and curvatures. Finally, as the speed of the vehicle increased, the number of matched features decreased. However, the reduction in matched features was not high enough to limit the speed of the vehicle.

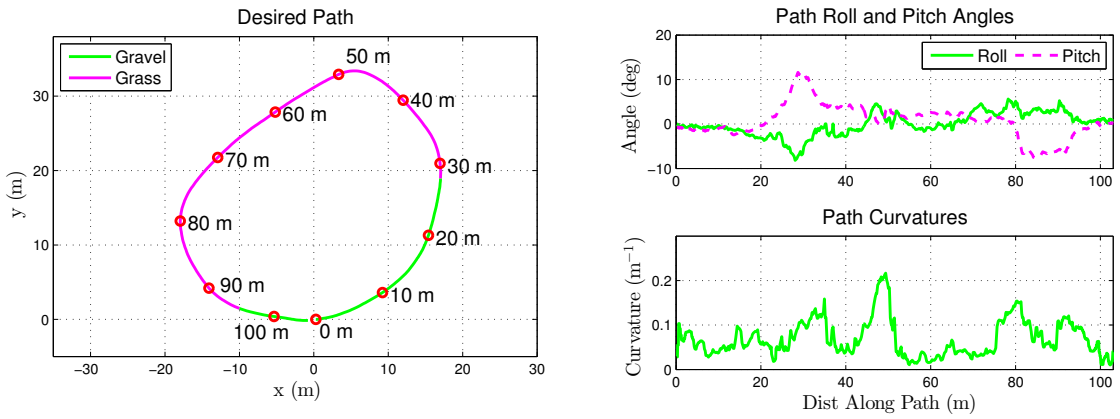


Figure 4.3: The path for the first experiment was approximately 100 m long and included gravel and grassy terrain. The path included slope angles up to 10°, side-slope angles up to 10°, and path curvatures up to 0.2 m^{-1}.

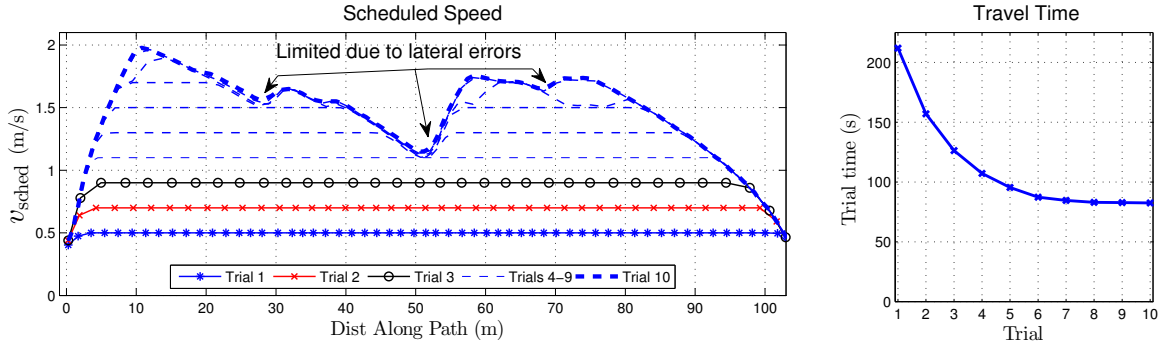


Figure 4.4: The speed scheduler maximizes speed while taking into account experience based on the vision system, path-tracking errors, and control inputs (Figure 4.5). During the 10th trial, the slowest scheduled speed occurs at around 50 m when the lateral path-tracking error is largest. The strategic increases in scheduled speed resulted in a significantly reduced travel time by the 10th trial. Once the speed schedule had converged, variations in the travel time were due largely to non-repeatable disturbances affecting the vehicle speed and path-tracking errors.

During the second experiment, the vehicle travelled a 375-m-long path 10 times, resulting in over 3 km of testing. As in the first experiment, the number of matched features, path-tracking errors, and commanded angular speeds were all within the specified limits for the first few trials resulting in a rapid increase in scheduled speeds (Figure 4.7). In this case, we were able to initialize the vehicle at a relatively safe speed of 0.5 m/s and the speed scheduling algorithm was able to autonomously reduce the travel time significantly (Figure 4.7).

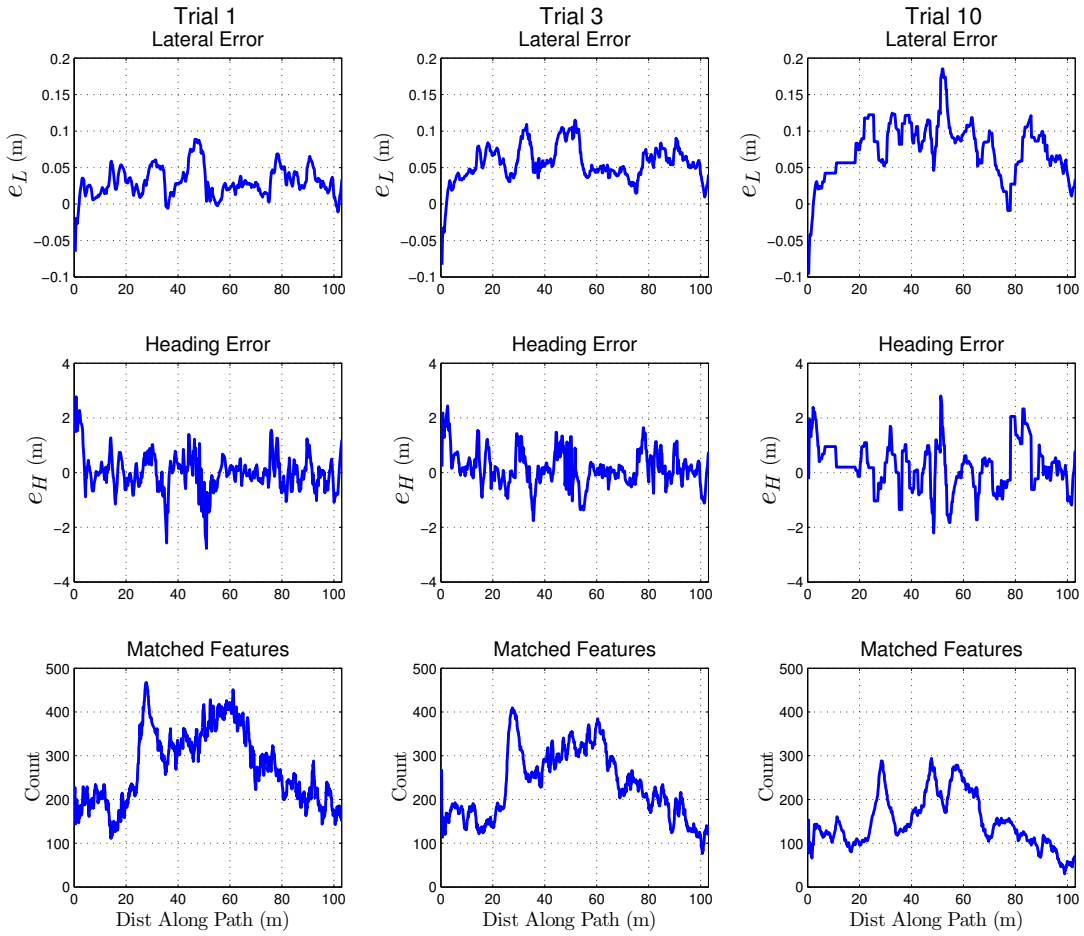


Figure 4.5: Experiment 1 Matched Features and Path-tracking Errors vs Distance: The number of matched features in trial 10 were reduced due to motion blur and variations in lighting relative to the first trial. However, the scheduled speed was not affected by the number of matched features or the heading errors. On the other hand, increased speed resulted in increased lateral path-tracking errors, which ended up being the common cause of limited speed.

4.4 Discussion

The algorithm proved to make appropriate decisions with respect to the localization system and path-tracking errors. In retrospect, however, it may have been possible to estimate the available speed increase more accurately by regressing from experience, including more than just the previous trial. This may have increased the speed increments during early trials. Furthermore, it may be useful to maintain experience as a function of time. For example, it is quite common for vision-based localization systems operating

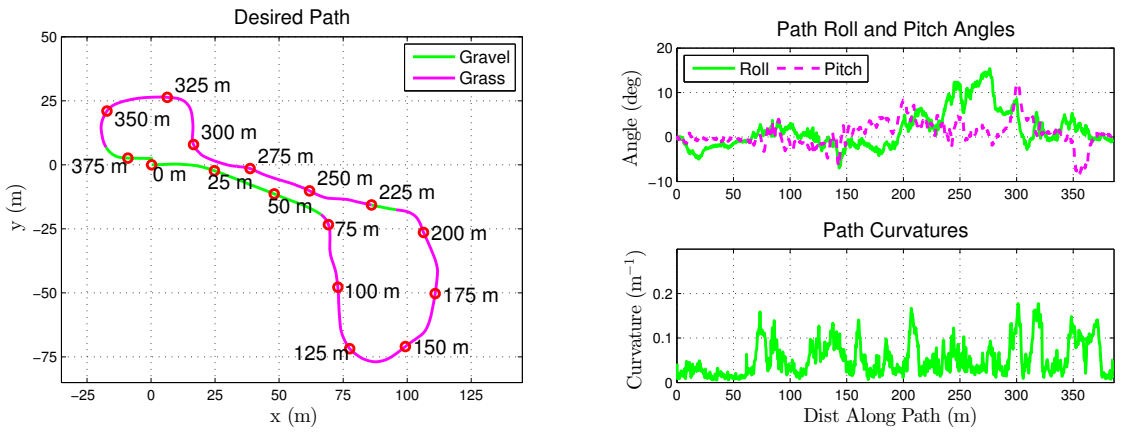


Figure 4.6: The path for the second experiment was approximately 375 m long and also included gravel and grassy terrain. The path included slope angles up to 10°, side-slope angles up to 15°, and path curvatures up to 0.2 m^{-1}.

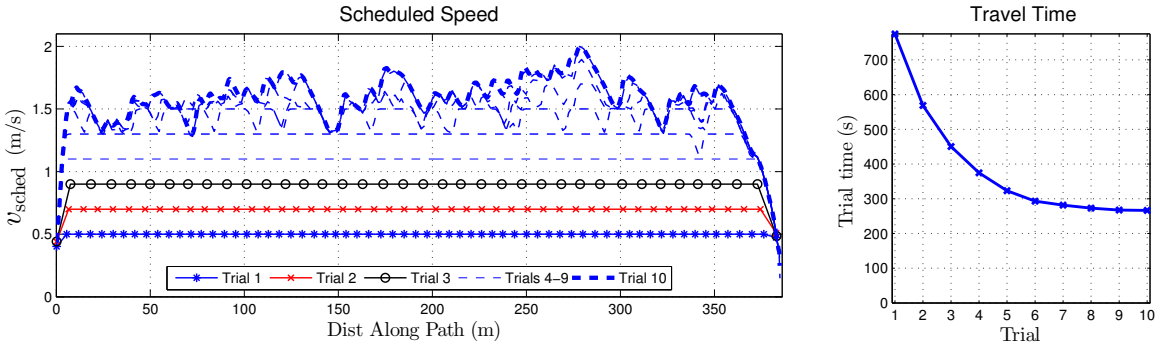


Figure 4.7: Experiment 2 Scheduled Speed vs Distance: During the second experiment, the experience-based speed scheduler generated a profile ranging from 0.5 m/s to 2.0 m/s after 10 trials. During the first four trials, the number of matched features, path-tracking errors, and commanded angular speeds were all within the specified limits resulting in a significant decrease in travel time.

outdoors to experience periodic lighting changes affecting the reliability of the system to localize. In such a case, one could imagine the system *anticipating* a poorly lit section of a path as a function of time-of-day and slowing down in advance. This may be especially beneficial for a system that does not repeat a path frequently enough to see the lighting change gradually.

4.5 Conclusion

In summary, this chapter presents an automated speed scheduler for path-repeating mobile robots. The novelty in the work is the incorporation of previous experience when com-

puting a speed schedule. Specifically, the algorithm collects localization, path-tracking, and control-input experience then generates a speed schedule for the next trial in two broad steps. First, the algorithm suggests a new speed profile based on the collected experience and the previous speed schedule. Second, the algorithm limits the suggested profile based on *a priori* speed and acceleration limits. The algorithm was implemented and tested on a large DMRV in two experiments with 10 trials each, resulting in over 4 km of driving. The algorithm proved to be effective at maximizing the robot speed while taking into account limits that are very difficult to predict in advance, such as those imposed by vision-based localization systems.

Chapter 5

Learning-based Nonlinear Model Predictive Control

5.1 Introduction

In this chapter, we present Learning-based Nonlinear Model Predictive Control (LB-NMPC) for a path-repeating mobile robot operating in challenging outdoor terrain[1]. The work was inspired by our previous work with ILC. While ILC is computationally appealing for real-time, large-scale applications since the feedforward signal is computed offline, we found through practical work that anytime learning and generalization are key requirements for flexible learning-based mobile robot operation in large-scale, off-road environments.

To enable anytime learning, the proposed LB-NMPC algorithm is based on a fixed, simple robot model and a learned disturbance model. Disturbances represent measured discrepancies between the *a priori* model and observed system behavior. As a result, experiences for the disturbance model can be collected anytime the algorithm has access to control inputs and accurate state estimates and thus can predict and measure the robot behavior. The algorithm may even gather experiences when the robot is under manual control and the localization algorithm provides state estimates. To enable generalization, disturbances are modelled as a Gaussian Process (GP) based on previous experience as a function of state, input, and other relevant variables. Modelling disturbances as a GP also enables the algorithm to learn complex model discrepancies without prior information.

[1] Associated video at http://tiny.cc/RobotLearnsDisturbances

	DMRV	**ROC6**	**Clearpath Husky**
Mass	600 kg	150 kg	50 kg
Size (LW)	2 m x 1.5 m	1.5 m x 0.5 m	0.9 m x 0.6 m
Steering	Ackermann Steering	Skid Steering	Skid Steering

Figure 5.1: Robots used to demonstrate the effectiveness of the learning controller. Despite significant differences in robot mass, wheel base, kinematics, and actuator designs, the algorithm uses the same nominal model for all three robots and learns disturbances over trials in order to accurately track desired paths.

Effectively, the goal is to use real-world experience to construct an accurate, low-uncertainty disturbance model instead of preprogramming accurate analytical models that are generally difficult to derive. Accurate mobile robot process models are typically difficult to derive *a priori* since (i) the terrain is often not known ahead of time, (ii) robot-terrain interaction models often do not exist, and (iii) even if such models did exist, finding model parameters is cumbersome. By providing the ability to learn anytime and to generalize from experience, the algorithm enables flexible and useful mobile robot operation in large-scale, outdoor environments. This work represents the first example to our knowledge of LB-NMPC applied to an autonomous mobile robot. The LB-NMPC algorithm also forms the basis for the remaining learning-based controllers presented in this thesis (Chapters 6 and 7).

Model Predictive Control (MPC) is a control framework that uses a process model directly. The current control action is obtained by solving, at each sampling instant, a finite-horizon optimal control problem using the current state of the plant as the initial state (Rawlings and Mayne, 2009). Kühne et al. (2005), Klančar and Škrjanc (2007), and

Xie and Fierro (2008) present MPC-based mobile robot controllers based on kinematic models and show results for robots travelling on smooth, flat surfaces. Howard et al. (2009) demonstrate MPC on a large-scale, outdoor robot navigating intricate paths. Finally, Peters and Iagnemma (2008) demonstrate MPC for a mobile robot where the process model includes effects such as tire deformation, wheel-terrain interaction, and suspension compliance. However, in each of these examples, the controllers are based on *a priori* models and, in some cases, rely on parameters whose determination in practice is challenging. In this work, our NMPC algorithm is based on a fixed nominal model and a learned, non-parametric disturbance model. This reduces the need for accurate *a priori* process models and parameter-specific observers while maintaining the benefits of MPC such as predictive behavior and constraint handling.

Unlike controllers based on fixed models, controllers using learned models gather data over time, incrementally constructing accurate approximations of the true system model. In this work, we model disturbances as a GP based on input-output data from previous trials. This approach enables both model flexibility and consistent uncertainty estimates (Rasmussen, 2006). For example, Kocijan et al. (2004) combine a GP model and MPC for the control of a simulated pH neutralization process. They represent the full dynamics of the system by a GP model trained on 400 observations of the chemical system. MPC is applied to control the system based on the offline-identified GP model (i.e., no online learning). While their work was restricted to offline simulation, our algorithm is used for real-time path-tracking and learns from trial to trial. Sparse GP approximations are one approach to enable fast, online GP evaluation, and do so by discarding some training points and keeping only 'inducing inputs', also known as 'support points' (Quiñonero-Candela and Rasmussen, 2005). An alternative are Local GP (LGP) methods, as implemented in this work, which enable online operation by dividing the GP input space into smaller subspaces and generating an LGP for each subspace (Rasmussen and Ghahramani, 2002; Snelson and Ghahramani, 2007). For example, Nguyen-Tuong et al. (2009) and Meier et al. (2014) focus on achieving online operation and use LGP models to approximate the inverse dynamics of manipulator arms. Unlike these two examples where many LGP models are generated for operation, we rapidly compute a single LGP model at every time-step based on a sliding window of learned data and use NMPC to enable predictive control. Finally, robustness of learning controllers is a large unanswered question. Aswani et al. (2013) focus on developing a safe and robust LB-MPC for linear systems. The approach provides guarantees on safety and robustness

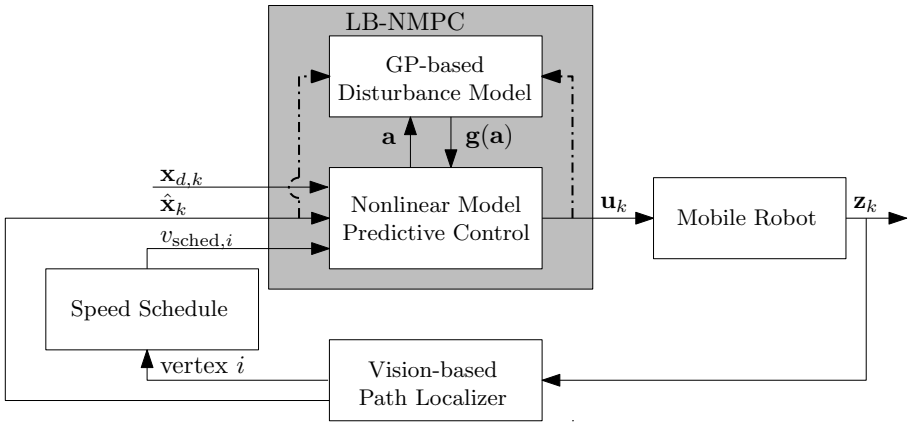


Figure 5.2: The LB-NMPC algorithm is composed of two parts: (i) the path-tracking NMPC algorithm that includes a nominal process model, and (ii) the GP-based Disturbance Model. During the first trial, the algorithm relies solely on the nominal process model to guide the vehicle along the desired path, $\mathbf{x}_d$. In subsequent trials, the NMPC algorithm uses the disturbance model as a correction to the nominal model at states, $\mathbf{a}$, to be defined in Section 7.2.1. Dashed lines indicate that the signals $\hat{\mathbf{x}}_k$ and $\mathbf{u}_k$ update the model. For experimental results, we combine the LB-NMPC algorithm (Section 7.2), an experience-based speed scheduler (Chapter 4), and the vision-based VT&R system.

by ensuring that the computed optimal control keeps the nominal model stable when it is subject to uncertainty. In this work, we do not explicitly consider the robustness of the controller but focus on the practical application of LB-NMPC to mobile robots. This requires continuous operation from the first trial and representation of complex disturbances by the learned model.

5.2 Mathematical Formulation

5.2.1 Nonlinear Model Predictive Control

At a given sample time, the NMPC algorithm finds a sequence of control inputs that optimizes the plant behaviour over a prediction horizon based on the current state. The first input in the optimal sequence is then applied to the system, resulting in a new system state. The entire process is repeated at the next sample time for the new system state. In traditional NMPC implementations (Rawlings and Mayne, 2009), the process model is specified *a priori* and remains unchanged during operation. In this work, we augment the process model with a disturbance model generated from experience in order to compensate for effects not captured by the fixed process model, such as environmental disturbances and unknown dynamics (Figure 5.2).

Full-State Feedback Control

Consider the following nonlinear, state-space system:

$$\mathbf{z}_{k+1} = \mathbf{f}_{\text{true}}(\mathbf{z}_k, \mathbf{u}_k), \tag{5.1}$$

with observable system state, $\mathbf{z}_k \in \mathbb{R}^n$, and control input, $\mathbf{u}_k \in \mathbb{R}^m$, both at time k. In this work, the true system is not known exactly and is represented by the sum of an *a priori* model and an experience-based, learned model,

$$\mathbf{z}_{k+1} = \overbrace{\mathbf{f}(\mathbf{z}_k, \mathbf{u}_k)}^{a\ priori\ \text{model}} + \overbrace{\mathbf{g}(\mathbf{z}_k, \mathbf{u}_k)}^{\text{learned disturbance model}}. \tag{5.2}$$

The models $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ are nonlinear process models: $\mathbf{f}(\cdot)$ is a known nominal process model representing our knowledge of $\mathbf{f}_{\text{true}}(\cdot)$, $\mathbf{g}(\cdot)$ is an (initially unknown) disturbance model representing discrepancies between the nominal model and the actual system behavior. In practice, disturbances are modelled as a GP (Section 5.2.2). The system is further assumed to be Markovian, thus the processes $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ involve only states from the current time.

As previously mentioned, the objective of the NMPC algorithm is to find a set of controls that optimizes the plant behaviour over a given prediction horizon. To this end, we define the cost function to be minimized over the next K time-steps as

$$J(\mathbf{u}) = (\mathbf{z}_d - \mathbf{z})^T \mathbf{Q} \, (\mathbf{z}_d - \mathbf{z}) + \mathbf{u}^T \mathbf{R} \, \mathbf{u}, \tag{5.3}$$

where $\mathbf{Q} \in \mathbb{R}^{Kn \times Kn}$ is positive semi-definite, $\mathbf{R} \in \mathbb{R}^{Km \times Km}$ is positive definite, $\mathbf{u}$ is a sequence of control inputs, $\mathbf{u} = (\mathbf{u}_k, \ldots, \mathbf{u}_{k+K-1})$, $\mathbf{z}_d$ is a sequence of desired states, $\mathbf{z}_d = (\mathbf{z}_{d,k+1}, \ldots, \mathbf{z}_{d,k+K})$, and $\mathbf{z}$ is a sequence of predicted states, $\mathbf{z} = (\mathbf{z}_{k+1}, \ldots, \mathbf{z}_{k+K})$, obtained from (5.2) when applying $\mathbf{u}$,

$$\begin{aligned} \begin{bmatrix} \mathbf{z}_{k+1} \\ \mathbf{z}_{k+2} \\ \vdots \\ \mathbf{z}_{k+K} \end{bmatrix} &= \begin{bmatrix} \mathbf{f}(\mathbf{z}_k, \mathbf{u}_k) + \mathbf{g}(\mathbf{z}_k, \mathbf{u}_k) \\ \mathbf{f}(\mathbf{z}_{k+1}, \mathbf{u}_{k+1}) + \mathbf{g}(\mathbf{z}_{k+1}, \mathbf{u}_{k+1}) \\ \vdots \\ \mathbf{f}(\mathbf{z}_{k+K-1}, \mathbf{u}_{k+K-1}) + \mathbf{g}(\mathbf{z}_{k+K-1}, \mathbf{u}_{k+K-1}) \end{bmatrix} \\ &= \mathbf{h}(\mathbf{z}_k, \mathbf{u}). \end{aligned} \tag{5.4}$$

Weighting on the state in (5.3) begins at time $k+1$ since the state at time k can no longer be affected by the control input. Also, by requiring $\mathbf{R}$ to be positive definite, inputs are guaranteed to be finite. Further restrictions on control inputs or states are commonly imposed using constraints when solving for the optimal control input (Diehl et al., 2009).

Since both our process model and disturbance model are nonlinear, the minimum of $J(\mathbf{u})$ must be found iteratively using a nonlinear optimization technique. In this work, we use unconstrained Gauss-Newton minimization (Nocedal and Wright, 1999) to solve the nonlinear least-squares problem. We begin by linearizing (5.4) around an initial guess for the optimal control input sequence, $\tilde{\mathbf{u}}$, with $\mathbf{u} = \tilde{\mathbf{u}} + \delta\mathbf{u}$. A good initial guess for $\tilde{\mathbf{u}}$ is the sequence of optimal inputs calculated in the previous time-step. For the first time-step, we use $\tilde{\mathbf{u}} = \mathbf{0}$. Now with $\mathbf{z}_k$ representing the current pose of the robot, $\tilde{\mathbf{z}}$ representing a sequence of states obtained from (5.4) when applying $\tilde{\mathbf{u}}$, and $\mathbf{z} = \tilde{\mathbf{z}} + \delta\mathbf{z}$, we find

$$\mathbf{z} \approx \tilde{\mathbf{z}} + \mathbf{H}\,\delta\mathbf{u}, \tag{5.5}$$

where $\mathbf{H}$ is the block-Jacobian of (5.4) with respect to $\mathbf{u}$,

$$\mathbf{H} = \left.\frac{\partial \mathbf{h}(\mathbf{z}_k, \mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{z}_k, \tilde{\mathbf{u}}}. \tag{5.6}$$

Evaluating this involves computing partials of $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$. In the case of $\mathbf{f}(\cdot)$, we have an analytical model and in the case of $\mathbf{g}(\cdot)$, the derivatives are tractable so long as an appropriate kernel function is chosen for use in the Gaussian process model (see Section 5.2.2).

Substituting $\mathbf{z} = \tilde{\mathbf{z}} + \mathbf{H}\,\delta\mathbf{u}$ and $\mathbf{u} = \tilde{\mathbf{u}} + \delta\mathbf{u}$ into (5.3) results in $J(\cdot)$ being quadratic in $\delta\mathbf{u}$,

$$J(\mathbf{u}) = (\mathbf{z}_d - \tilde{\mathbf{z}} - \delta\mathbf{z})^T \mathbf{Q}\,(\mathbf{z}_d - \tilde{\mathbf{z}} - \delta\mathbf{z}) + (\tilde{\mathbf{u}} + \delta\mathbf{u})^T \mathbf{R}\,(\tilde{\mathbf{u}} + \delta\mathbf{u}) \tag{5.7}$$

$$\approx (\mathbf{z}_d - \tilde{\mathbf{z}} - \mathbf{H}\,\delta\mathbf{u})^T \mathbf{Q}\,(\mathbf{z}_d - \tilde{\mathbf{z}} - \mathbf{H}\,\delta\mathbf{u}) + (\tilde{\mathbf{u}} + \delta\mathbf{u})^T \mathbf{R}\,(\tilde{\mathbf{u}} + \delta\mathbf{u}). \tag{5.8}$$

We can find the value of $\delta\mathbf{u}$ that minimizes $J(\cdot)$ by solving

$$\frac{\partial J(\mathbf{u})}{\partial \delta\mathbf{u}} = \mathbf{0} \tag{5.9}$$

for $\delta\mathbf{u}$, and compute the control input about which (3) is linearized in the next iteration,

$$\tilde{\mathbf{u}} \leftarrow \tilde{\mathbf{u}} + \delta\mathbf{u}. \tag{5.10}$$

After iterating to convergence, we apply the first element of the resulting optimal control input sequence for one time-step, and start all over at the next time-step.

Partial-State Feedback Control

Consider the following system, covering most robotic systems, where the dynamics cascade into the kinematics:

$$\text{kinematics:} \quad \mathbf{x}_{k+1} = \mathbf{f}_{\mathbf{x},\text{true}}(\mathbf{x}_k, \mathbf{v}_k) \tag{5.11}$$

$$\text{dynamics:} \quad \mathbf{v}_{k+1} = \mathbf{f}_{\mathbf{v},\text{true}}(\mathbf{v}_k, \mathbf{u}_k), \tag{5.12}$$

with system state, $\mathbf{z}_k = (\mathbf{x}_k, \mathbf{v}_k)$, representing pose, $\mathbf{x}_k \in \mathbb{R}^{n_\mathbf{x}}$, and velocity, $\mathbf{v}_k \in \mathbb{R}^{n_\mathbf{v}}$, separately, and control input, $\mathbf{u}_k$, all at time k. By substituting $\mathbf{v}_k = \mathbf{f}_{\mathbf{v},\text{true}}(\mathbf{v}_{k-1}, \mathbf{u}_{k-1})$ into (5.11), we can write

$$\mathbf{x}_{k+1} = \mathbf{f}'_{\text{true}}(\mathbf{x}_k, \mathbf{v}_{k-1}, \mathbf{u}_{k-1}). \tag{5.13}$$

Now, if we assume that our *a priori* model represents the robot kinematics with $\mathbf{v}_k = \mathbf{u}_k$ (i.e., the *a priori* model assumes robot dynamics are negligeable), and that the true process, $\mathbf{f}'_{\text{true}}(\cdot)$, can be represented by the sum of our *a priori* and learned models, we find

$$\mathbf{x}_{k+1} = \overbrace{\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)}^{\textit{a priori}\text{ model}} + \overbrace{\mathbf{g}(\underbrace{\mathbf{x}_k, \mathbf{v}_{k-1}, \mathbf{u}_k, \mathbf{u}_{k-1}}_{\mathbf{a}_k})}^{\text{learned model}}, \tag{5.14}$$

with disturbance query state, $\mathbf{a}_k \in \mathbb{R}^p$,

$$\mathbf{a}_k = (\mathbf{x}_k, \mathbf{v}_{k-1}, \mathbf{u}_k, \mathbf{u}_{k-1}). \tag{5.15}$$

In other words, in order to capture the dynamics of the system, the disturbance query state, $\mathbf{a}_k$, is now required to include historic states. We can further define the corre-

sponding cost function to be

$$J(\mathbf{u}) = (\mathbf{x}_d - \mathbf{x})^T \mathbf{Q}_{\mathbf{x}} (\mathbf{x}_d - \mathbf{x}) + \mathbf{u}^T \mathbf{R}\, \mathbf{u}, \tag{5.16}$$

where $\mathbf{Q}_{\mathbf{x}} \in \mathbb{R}^{Kn_{\mathbf{x}} \times Kn_{\mathbf{x}}}$ is positive semi-definite, $\mathbf{R}$ and $\mathbf{u}$ are as in (5.3), $\mathbf{x}_d$ is a sequence of desired states, $\mathbf{x}_d = (\mathbf{x}_{d,k+1}, \ldots, \mathbf{x}_{d,k+K})$, $\mathbf{x}$ is a sequence of predicted states, $\mathbf{x} = (\mathbf{x}_{k+1}, \ldots, \mathbf{x}_{k+K})$, and K is the given prediction horizon length. The state, $\mathbf{x}_k$, and learned model, $\mathbf{g}(\cdot)$, are now of reduced dimension, $n_{\mathbf{x}} \leq n$, while still capturing both unknown disturbances and unmodelled dynamics. This approach enables a user to provide a simple *a priori* model with few parameters, if any. Further, the derivation suggests that the approach is applicable to processes with even higher-order dynamics by continuing to add historic states to the disturbance dependency.

5.2.2 Gaussian Process Disturbance Model

We model the disturbance, $\mathbf{g}(\cdot)$, as a GP, which is a function of a disturbance dependency, $\mathbf{a}$. The model depends on disturbance observations collected during previous trials, representing attempts at achieving a control objective, such as tracking a path from start to finish. At time k, we use the estimated poses, $\hat{\mathbf{x}}_k$ and $\hat{\mathbf{x}}_{k-1}$, from the VT&R vision-based localization system, and the control input, $\mathbf{u}_{k-1}$, to isolate (5.14) for $\hat{\mathbf{g}}(\mathbf{a}_{k-1})$,

$$\hat{\mathbf{g}}(\mathbf{a}_{k-1}) = \hat{\mathbf{x}}_k - \mathbf{f}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}). \tag{5.17}$$

We collect observations for all sample times in a trial and organize the data from trial j into a set of data pairs, $\mathcal{D}^{(j)} = \big\{(\mathbf{a}_0, \hat{\mathbf{g}}(\mathbf{a}_0)), \ldots, (\mathbf{a}_k, \hat{\mathbf{g}}(\mathbf{a}_k)), \ldots, (\mathbf{a}_{N_j-1}, \hat{\mathbf{g}}(\mathbf{a}_{N_j-1}))\big\}$, where $N^{(j)}$ is the number of time-steps it took to travel the length of the path during trial j, and $\mathbf{a}_k$ is as defined in (5.15). After j trials we have multiple datasets, $\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(j)}$, that we combine into a single database, $\mathcal{D}$, with $N = N^{(1)} + \cdots + N^{(j)}$ observations. We also drop the time-step index, k, on each data pair in $\mathcal{D}$, so that when referring to $\mathbf{a}_{\mathcal{D},i}$ or $\hat{\mathbf{g}}_{\mathcal{D},i}$, we mean the ith pair of data in the superset $\mathcal{D}$. Note that there is no requirement that $N^{(j)} = N^{(j-1)}$ as the system simply collects observations as they occur for the length of time that it takes to complete a trial. Moreover, all experiences are treated equally as observations of the underlying unmodelled disturbance. In fact, the system collects experience data whenever it moves while repeating the desired path. As a result, the system does not require identical initial or terminal conditions, or speed schedules.

In this work, we train a separate GP for each dimension in $\mathbf{g}(\cdot) \in \mathbb{R}^n$ to model disturbances as the robot travels along a path. This approach makes the assumption that disturbances are uncorrelated. For simplicity of discussion, we will assume for now that $n = 1$ and denote $\hat{\mathbf{g}}_{\mathcal{D},i}$ by $\hat{g}_{\mathcal{D},i}$. The learned model assumes a measured disturbance originates from a Gaussian process model,

$$\hat{g}(\mathbf{a}_{\mathcal{D},i}) \sim \mathcal{GP}\left(0, k(\mathbf{a}_{\mathcal{D},i}, \mathbf{a}_{\mathcal{D},i})\right), \tag{5.18}$$

with zero mean and kernel function, $k(\mathbf{a}_{\mathcal{D},i}, \mathbf{a}_{\mathcal{D},i})$, to be defined. We assume that each disturbance measurement is corrupted by zero-mean additive noise with variance, σ_n^2, so that $\hat{g}_{\mathcal{D},i} = g_{\mathcal{D},i} + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma_n^2)$. Then a modelled disturbance, $g(\mathbf{a}_k)$, and the N observed disturbances, $\hat{\mathbf{g}} = (\hat{g}_{\mathcal{D},1}, \ldots, \hat{g}_{\mathcal{D},N})$, are jointly Gaussian,

$$\begin{bmatrix} \hat{\mathbf{g}} \\ g(\mathbf{a}_k) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k}(\mathbf{a}_k)^T \\ \mathbf{k}(\mathbf{a}_k) & k(\mathbf{a}_k, \mathbf{a}_k) \end{bmatrix}\right), \tag{5.19}$$

where $\mathbf{k}(\mathbf{a}_k) = [k(\mathbf{a}_k, \mathbf{a}_{\mathcal{D},1}), k(\mathbf{a}_k, \mathbf{a}_{\mathcal{D},2}), \ldots, k(\mathbf{a}_k, \mathbf{a}_{\mathcal{D},N})]$, and $\mathbf{K} \in \mathbb{R}^{N \times N}$ with $(\mathbf{K})_{i,j} = k(\mathbf{a}_{\mathcal{D},i}, \mathbf{a}_{\mathcal{D},j})$. In our case, we use the Squared-Exponential (SE) kernel (Rasmussen, 2006),

$$k(\mathbf{a}_i, \mathbf{a}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{a}_i - \mathbf{a}_j)^T \mathbf{M}^{-2} (\mathbf{a}_i - \mathbf{a}_j)\right) + \sigma_n^2 \, \delta_{ij}, \tag{5.20}$$

where δ_{ij} is the Kronecker delta, that is 1 if and only if $i = j$ and 0 otherwise, and the constants $\mathbf{M}$, σ_f, and σ_n are hyperparameters. The SE kernel function is an example of a Radial Basis Function (Rasmussen, 2006) and is commonly used to represent continuous functions based on dense data. Further, the SE kernel is continuously and analytically differentiable, enabling the rapid computation of derivatives for the Gauss-Newton optimization algorithm. In our implementation with $\mathbf{a}_k \in \mathbb{R}^p$, the constant $\mathbf{M}$ is a diagonal matrix, $\mathbf{M} = \text{diag}(\mathbf{m})$, $\mathbf{m} \in \mathbb{R}^p$, representing the relevance of each component in $\mathbf{a}_k$, while the constants, σ_f^2 and σ_n^2, represent the process variation and measurement noise, respectively. Finally, we have that the prediction, $g(\mathbf{a}_k)$, of the disturbance at an arbitrary state, $\mathbf{a}_k$, is also Gaussian distributed,

$$g(\mathbf{a}_k)|\hat{\mathbf{g}} \sim \mathcal{N}\left(\mathbf{k}(\mathbf{a}_k)\mathbf{K}^{-1}\hat{\mathbf{g}}, \; k(\mathbf{a}_k, \mathbf{a}_k) - \mathbf{k}(\mathbf{a}_k)\mathbf{K}^{-1}\mathbf{k}(\mathbf{a}_k)^T\right). \tag{5.21}$$

In this work, we only make use of the predicted mean value of disturbances. However,

in later work, the predicted variance is used as an indication of the uncertainty in the learned model for robust NMPC formulations. Finally, we include further detail on the storage and retrieval of observations for online operation in Section 5.3.2.

5.2.3 Gaussian Process Hyperparameter Selection

Having defined the NMPC algorithm and disturbance model, $\mathbf{g}(\mathbf{a}_k)$, it remains to define the source of the hyperparameters, $\mathbf{M}$, σ_f^2, and σ_n^2. Solving for optimal hyperparameters is not currently a real-time process in our experiments. As such, we assume that a suitable set of hyperparameters has been determined prior to each trial based on previous experience (i.e., from previous trials). For the first trial, when the robot has no experience, the predicted disturbance is zero. Given a set of experiences, we find the optimal hyperparameters offline by maximizing the log marginal likelihood of collected experiences using a gradient ascent algorithm (Rasmussen, 2006). In order to avoid local maxima, the algorithm is repeated several times, initialized with different initial values, and the set of hyperparameters resulting in the greatest likelihood is selected.

5.2.4 Illustrative Example

In this section, we highlight the benefits of LB-NMPC and present an illustrative example comparing: (i) fixed feedback control, (ii) non-learning NMPC, and (iii) LB-NMPC. Consider the following process model:

$$z_{k+1} = \alpha\, z_k + \Delta t\, \beta\, u_k + \Delta t\, d_k, \tag{5.22}$$

with system state, $z_k \in \mathbb{R}$, control input, $u_k \in \mathbb{R}$, and time-dependent disturbance, $d_k \in \mathbb{R}$, shown in Figure 5.3. Further, α, $\beta \in \mathbb{R}$ are unknown constants. In simulation, they are 0.99 and 0.5, respectively. The goal is to track a sequence of desired states, $z_{\mathrm{d},k}$, as shown in green in Figure 5.3, which is known to the example controllers prior to starting. The feedback controller uses a simple feedback law,

$$u_{\mathrm{fb},k} = k_{\mathrm{fb}}\,(z_{d,k} - z_k). \tag{5.23}$$

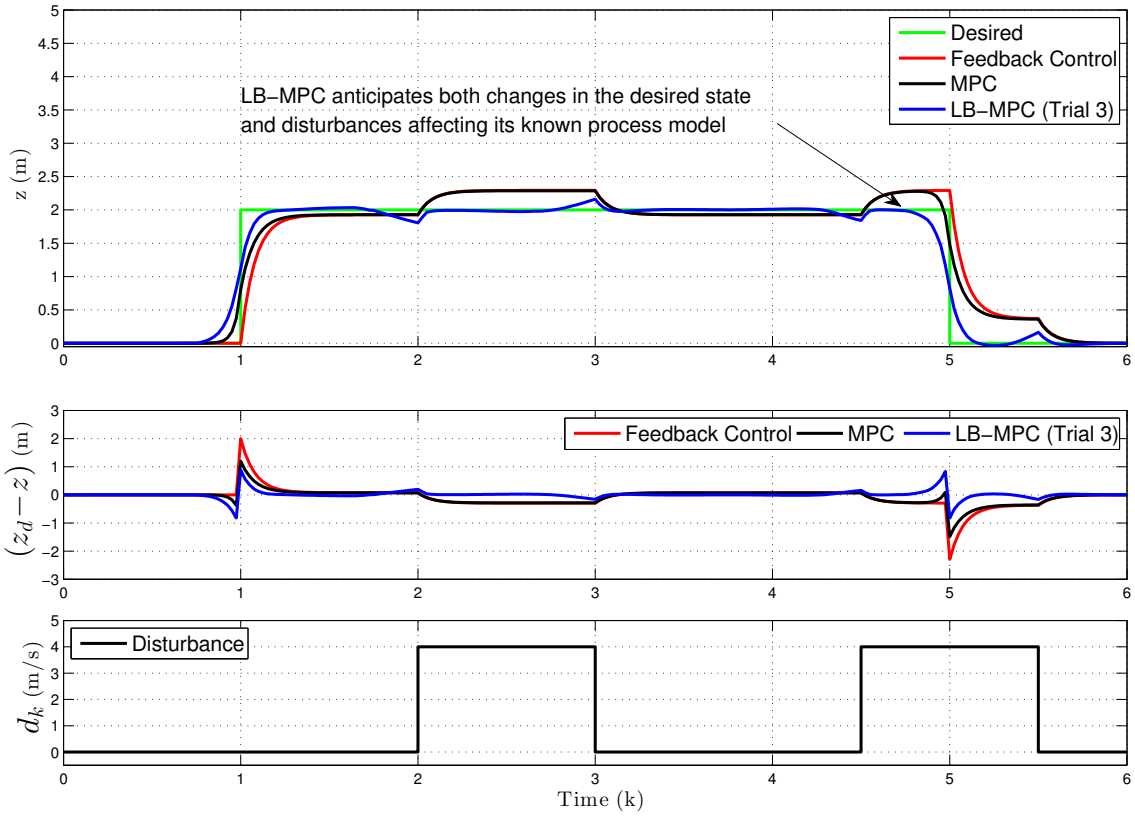


Figure 5.3: Compared to both simple feedback control (red) and MPC (black), LB-NMPC (blue) is able to anticipate and reduce errors caused by changes in the desired state and unmodelled disturbances.

Both the NMPC and LB-NMPC controllers assume a nominal process model,

$$z_{k+1} = z_k + \Delta t\, u_k, \tag{5.24}$$

a prediction horizon, $K = 10$, and a cost function (5.3), with $\mathbf{Q} = 10 \times \mathbf{1}$ and $\mathbf{R} = 0.01 \times \mathbf{1}$, where $\mathbf{1}$ is the identity matrix. The LB-NMPC algorithm also includes a learned disturbance model, as described in Section 5.2.1, such that the complete system model used by the LB-NMPC algorithm is

$$z_{k+1} = z_k + \Delta t\, u_k + g(z_k, u_k, k). \tag{5.25}$$

In this simple example, the disturbances are a function of time (5.22) and hence the learned disturbance is a function of time, k. However, in practice, we assume disturbances are time-invariant (5.14).

As expected, the feedback controller is incapable of anticipating errors caused by

either changes in desired state, $z_{d,k}$, or disturbances, d_k (Figure 5.3). On the other hand, the NMPC controller (without a learned model) enables some amount of predictive control to reduce tracking errors due to changes in desired state. However, tracking errors are not cancelled completely because the NMPC algorithm does not have the correct process model. Finally, the LB-NMPC algorithm exploits its previous experience to predict and compensate for both changes in the desired state and unknown disturbances not anticipated by the *a priori* process model.

5.3 Implementation

5.3.1 Robot Model

In this work, robots are modelled as unicycle-type vehicles (Figure 3.4) with state (5.14), $\mathbf{x}_k = (x_k, y_k, \theta_k)$. At every time-step, the VT&R localization algorithm provides the position of the robot, $\mathbf{x}_k$, relative to the nearest desired pose by Euclidean distance (Figure 5.2). The robots have two control inputs, their linear and angular velocities, $\mathbf{u}_k = (v_{\text{cmd},k}, \omega_{\text{cmd},k})$. Prior to each trial, scheduled path speeds are optimized by the experience-based speed scheduler (Chapter 4). Then the commanded linear velocity, $v_{\text{cmd},k}$, is constrained to the scheduled speed for the jth trial at the nearest path vertex, $v_{\text{cmd},k} = v^{(j)}_{\text{sched},i}$, leaving only the angular velocity, $\omega_{\text{cmd},k}$, for the NMPC algorithm to optimize considering (5.16).

When the time between control signal updates is defined as Δt, the resulting nominal process model employed by the NMPC algorithm is

$$\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_k + \Delta t \begin{bmatrix} \cos\theta_k & 0 \\ \sin\theta_k & 0 \\ 0 & 1 \end{bmatrix} \mathbf{u}_k, \tag{5.26}$$

which represents a simple kinematic model for our robot; it does not account for dynamics or environmental disturbances. We use the same *a priori* model for all robots in our experiments, despite them being quite different in scale (Figure 5.1).

The velocity state variables are $\mathbf{v}_k = (v_{\text{act},k}, \omega_{\text{act},k})$, which represent the actual linear and rotational speeds of the robot. These will differ from the commanded ones, $\mathbf{u}_k$, owing to the fact that the robots we are working with have underlying control loops that attempt to drive the robot at the commanded velocities. However, the combined

dynamics of the robot and these rate controllers are not modelled. We allow the LB-NMPC algorithm to learn these dynamics, as well as any other systematic disturbances, based on experience.

In order to build and query the learned model, $\mathbf{g}(\cdot)$, throughout the prediction horizon, we require all of the quantities in (5.15): $\mathbf{a}_{k+i} = (\mathbf{x}_{k+i}, \mathbf{v}_{k-1+i}, \mathbf{u}_{k+i}, \mathbf{u}_{k-1+i})$, $i = 0 \dots K-1$. We know $\mathbf{u}_{k+i}$ and $\mathbf{u}_{k-1+i}$, as these are commanded inputs. We initially obtain the robot position from our vision-based localization system, $\mathbf{x}_k = \hat{\mathbf{x}}_k$, then from our system model (5.14), $\mathbf{x}_{k+i+1} = \mathbf{f}(\mathbf{x}_{k+i}, \mathbf{u}_{k+i}) + \mathbf{g}(\mathbf{a}_{k+i})$. Finally, we compute the velocity state variables, $\mathbf{v}_{k-1+i} = (v_{\text{act},k-1+i}, \omega_{\text{act},k-1+i})$, based on the computed robot positions,

$$v_{\text{act},k-1+i} = \frac{\sqrt{(x_{k+i} - x_{k-1+i})^2 + (y_{k+i} - y_{k-1+i})^2}}{\Delta t},$$
$$\omega_{\text{act},k-1+i} = \frac{(\theta_{k+i} - \theta_{k-1+i})}{\Delta t}.$$

Since $\hat{\mathbf{x}}_k$ and $\hat{\mathbf{x}}_{k-1}$ come from our vision-based localization system, we are able to initialize the predictive controller with accurate velocity estimates with respect to the ground regardless of situations with large amounts of wheel slip.

5.3.2 Managing Experiences

In order to ensure the LB-NMPC algorithm is executed in constant computation time, our implementation requires the ability to use a subset of the observed experiences when computing a disturbance. Similar to work by Nguyen-Tuong et al. (2009) and Meier et al. (2014), we employ a local model. However, unlike their work, we use a single sliding local model. As experiences are collected, they are stored in bins, $\mathcal{D}_{i,l}$, by path vertex, i, and commanded velocity, $l = \lfloor v_{\text{cmd},k}/v_{\text{bin}} \rfloor$, where v_{bin} represents the velocity discretization and $\lfloor \cdot \rfloor$ represents the floor function. When the number of experiences in a bin exceeds a threshold, c_{bin}, the oldest experience in the bin is discarded. Then, when computing a control input at the ith vertex, a 'local' dataset is created, drawing experiences from bins at nearby path vertices and commanded velocities, $\mathcal{D} = \{\mathcal{D}_{a,b} | \, a \in \{i - c_{\text{vertex}}, \dots, i + c_{\text{vertex}}\}, b \in \{l - c_{\text{velocity}}, \dots, l + c_{\text{velocity}}\}\}$. Thus, models are effectively assembled on demand rather than precomputing hundreds of local models, enabling a constant-time algorithm independent of path length or deployment time.

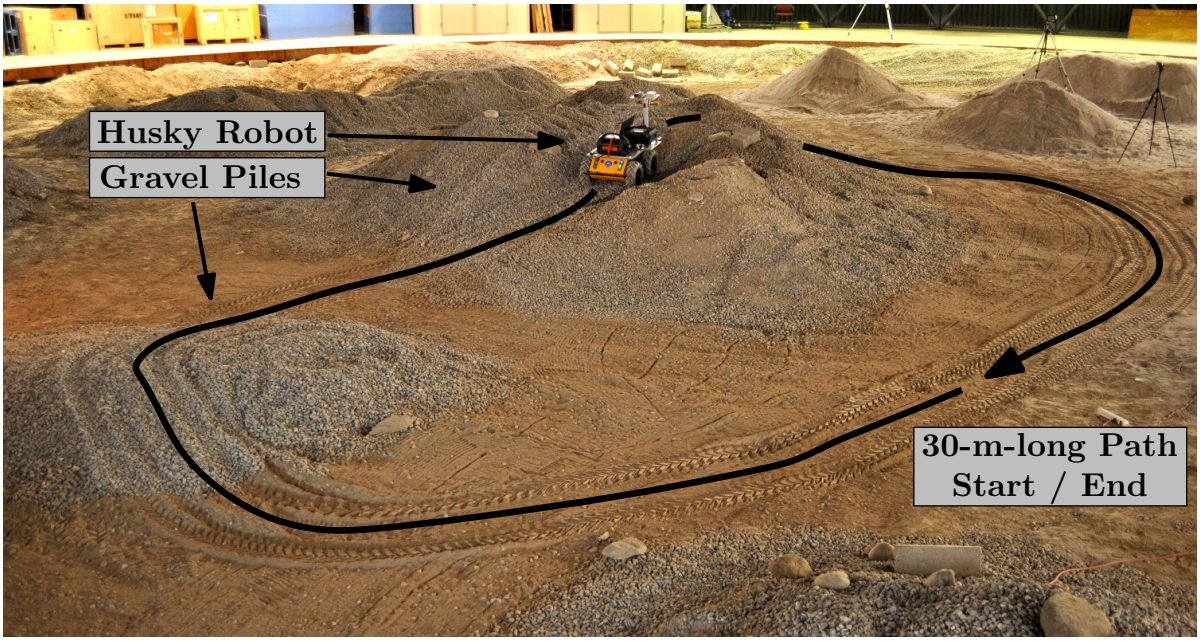


Figure 5.4: The first and second experiments were conducted inside the University of Toronto Institute for Aerospace Studies (UTIAS) MarsDome on gravel, sand, and loose dirt. The 30-m-long path, shown here, was used for the first experiment. In all experiments, the nominal unicycle model used in our LB-NMPC algorithm included no prior information on wheel-terrain interactions or robot dynamics.

5.4 Experimental Results

5.4.1 Overview

We tested the LB-NMPC algorithm in three different experiments involving three significantly different mobile robots (Figure 5.1) and paths with dirt, gravel, sand, grass, inclines, and side slopes. This resulted in over 3 km of learning-enabled path-tracking in outdoor, off-road environments. The three tests demonstrate the algorithm's effectiveness at reducing path-tracking errors with only cursory prior knowledge of the robot's behaviour (i.e., that it could be treated as a unicycle robot, Section 5.3.1). Details on the tuning parameters are presented in Section 5.4.2.

The first experiment (Section 5.4.3) demonstrated the algorithm's ability to learn unmodelled environmental disturbances. We tested on a 30-m-long path including slopes, dusty ground, and loose gravel surfaces (Figure 5.4). The robot was a 50 kg, four-wheeled Clearpath Husky robot travelling at a desired speed of 0.4 m/s (i.e., the automated speed scheduler was disabled for the first experiment). With a 0.5 m wheelbase, Husky robots

are relatively small and agile skid-steered mobile robots. As such, the path included slope angles up to 15°, side-slope angles up to 15°, and path curvatures up to $1\,\mathrm{m}^{-1}$.

The second experiment (Section 5.4.4) demonstrated the algorithm's ability to interpolate and extrapolate from previous experience. We used a 150 kg, six-wheeled ROC6 robot (Figure 5.1) learning to drive at a range of scheduled speeds over 20 trials on a 60-m-long path. Like the Husky, the ROC6 robot is a skid-steered platform. However, the ROC6 is heavier and longer, with a 1.5 m wheelbase, and is better suited to operate in more open terrains at higher speeds. Scheduled speeds for each trial were provided by the automated scheduler presented in Chapter 4. The scheduler used matched features, path-tracking errors, and control inputs from previous trials to determine safe speeds for the next trial.

Finally, the third experiment (Section 5.4.5) further demonstrated the algorithm's ability to learn disturbances due to robot design. Whereas the first two experiments involved skid-steered robots, this experiment used a 600 kg, Ackermann-steered DMRV robot (Figure 5.1). Traditional path-tracking controllers would represent the robot using a bicycle model (Figure 5.5) with steering angle, $\delta_{\mathrm{cmd},k}$, and linear velocity, $v_{\mathrm{cmd},k}$, as control inputs. However, in this work, the LB-NMPC algorithm treats the Ackermann-steered robot as a unicycle robot with linear and angular velocity commands, $v_{\mathrm{cmd},k}$ and $\omega_{\mathrm{cmd},k}$, respectively. The robot then converted these velocity commands to a steering angle, $\delta_{\mathrm{cmd},k}$,

$$\delta_{\mathrm{cmd},k} = \tan^{-1}\left(\frac{L\,\omega_{\mathrm{cmd},k}}{v_{\mathrm{cmd},k}}\right), \tag{5.27}$$

where L is defined as the wheelbase of the Ackermann-steered robot. The robot learned to drive at a range of scheduled speeds over 10 trials on a 100-m-long path. Like experiment 2, the scheduled speeds for each trial, $v_{\mathrm{sched},k}$, were generated using the same automated speed scheduler as was used in the second experiment (Chapter 4).

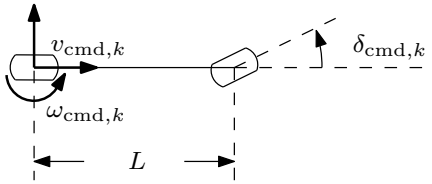


Figure 5.5: Here we show the relationship between steering angle of an Ackermann-steered robot, $\delta_{\mathrm{cmd},k}$, and the linear and angular velocities, $v_{\mathrm{cmd},k}$ and $\omega_{\mathrm{cmd},k}$, respectively. In experiment 3, we used the LB-NMPC algorithm for path-tracking on a 600 kg, Ackermann-steered mobile robot.

The first and second experiments were performed in the University of Toronto Institute for Aerospace Studies (UTIAS) MarsDome in Toronto, Ontario, Canada (Figure 5.4). The third experiment was performed at the Defence Research and Development Canada (DRDC) Experimental Proving Grounds in Suffield, Alberta, Canada. In all experiments, the controller described in Section 5.2 was implemented and run in addition to the VT&R software (Chapter 2) on a Lenovo W530 laptop with an Intel 2.6 Ghz Core i7 processor with 16 GB of RAM. The camera in all tests was a Point Grey Bumblebee XB3 stereo camera. The resulting real-time localization and path-tracking control signals were generated at approximately 10 Hz. As previously mentioned, hyperparameter selection is currently an offline process, taking up to 5 minutes in the later trials of an experiment when the system had accumulated approximately 5000 experiences. Since GPS was not available, the improvement due to the LB-NMPC algorithm was quantified by the localization of the VT&R algorithm.

5.4.2 Tuning Parameters

The performance of the system was adjusted using the NMPC weighting matrices $\mathbf{Q_x}$ and $\mathbf{R}$, the experience management parameters, and the speed scheduler gains and thresholds. The weighting matrices for each test were selected in advance ranging from roughly a 3:1 ratio weighting path-tracking errors and control inputs for the 50 kg Husky to a 1:1 ratio for the 600 kg DMRV robot. The increased weighting on the control inputs for the heavier robots was selected to ensure controller stability at higher speeds. Local GP models were generated based on a sliding window of size, $c_{\text{vertex}} = 5$ and $c_{\text{velocity}} = 1$, where velocities were discretized by $v_{\text{bin}} = 0.25\,\text{m/s}$. The maximum number of experiences per bin, c_{bin}, was set to 4, resulting in local models based on up to 180 experiences. Finally, the speed scheduler parameters we used are shown in Table 5.1.

	γ_1	γ_2	λ_L	λ_H	λ_{feat}	λ_ω	λ_{db}
Experiment 1	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Experiment 2	0.15	0.1	0.15 m	10°	30	1.0 rad/s	1.1
Experiment 3	0.2	0.15	0.15 m	10°	30	1.0 rad/s	1.1

Table 5.1: Scheduler gains and thresholds. The scheduler was not used in experiment 1.

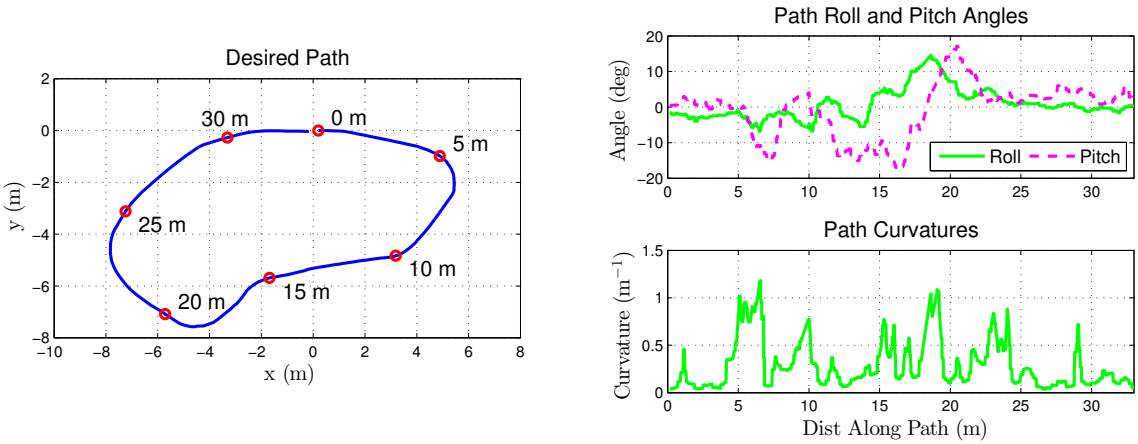


Figure 5.6: The path for the first experiment was approximately 30 m long and included sandy and gravel terrain. The path included slope angles up to 10°, side-slope angles up to 15°, and path curvatures up to 1.2 m^{-1}.

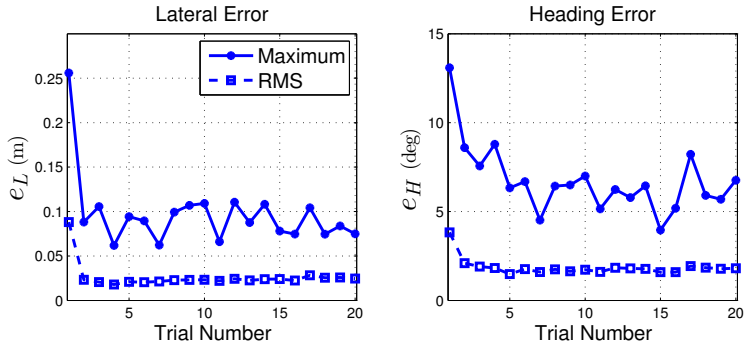


Figure 5.7: The maximum and Root-Mean-Square (RMS) path-tracking errors in experiment 1 are reduced significantly within the first few trials. Since MPC is an optimal controller balancing path-tracking errors and control input, we do not expect the path-tracking errors to be eliminated completely.

5.4.3 Experiment 1: Learning to Follow a Path with a Fixed Speed Schedule

In the first experiment, the 50 kg Husky robot autonomously travelled the length of a 30-m-long path for 20 trials at a fixed speed of 0.4 m/s resulting in 600 m of travel (Figure 5.6). In Figure 5.7, we show plots of the maximum and Root-Mean-Square (RMS) path-tracking errors vs. trial number. By disabling the speed scheduler for experiment 1, the learned model was allowed to converge. As a result, the LB-NMPC algorithm successfully reduced the maximum lateral and heading errors by roughly 75% in the first

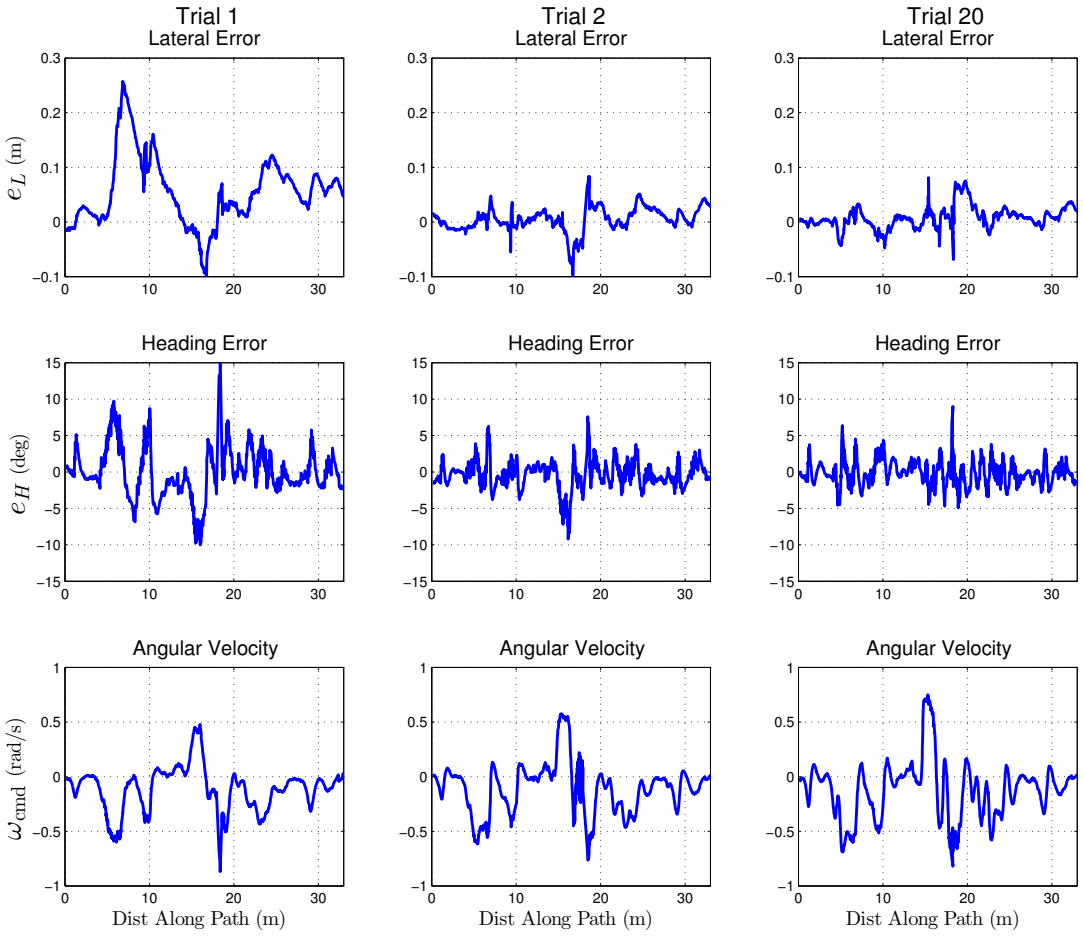


Figure 5.8: He we show the experiment 1 lateral and heading path-tracking errors, $e_L^{(j)}$ and $e_H^{(j)}$, and the commanded angular velocity, $\omega_{\text{cmd}}^{(j)}$. The model-based approach to learning allows for generalization from previous experience and thus rapid reductions in path-tracking errors over trials.

few trials, then maintained these errors for the next 15 trials. However, even after many trials, the maximum and RMS errors continued to vary. We suspect that these changes were due mainly to evolving path conditions (e.g., ruts, dirt piles) and our experience management scheme, that enables real-time computation and changing disturbances by forgetting experiences over time. Figure 5.8 shows plots of path-tracking errors and angular control input vs. distance along the path. The plots show the difference between the first trial, when the learned model has no experience from which to draw, and the 20^{th} trial, when the learned model has a significant amount of experience from which to draw. The heading and lateral errors, e_H and e_L, respectively, reached their peaks

in trial 20 around 14-22 m along the path where the path pitched forward, rolled to the right, and turned to the right. This section also corresponded to the largest changes in control input between the first and last trial.

5.4.4 Experiment 2: Learning to Follow a Path at Increasing Speeds

In the second experiment, the 150 kg ROC6 robot autonomously travelled the length of a 60-m-long path at a range of scheduled speeds over 20 trials to demonstrate the ability of the algorithm to interpolate and extrapolate from learned experiences (Figure 5.9). In this experiment, speeds were automatically scheduled using the experience-based algorithm presented in Chapter 4. Figure 5.10 shows plots of the overall travel time, maximum path-tracking errors, and RMS path-tracking errors vs. trial number. The LB-NMPC algorithm reduced the lateral and heading errors by roughly 50% over the course of the 20 trials while learning disturbances at speeds ranging from 0.35 to 1.0 m/s. For trials 16 through 20, the speed scheduler was disabled and instead the speed was

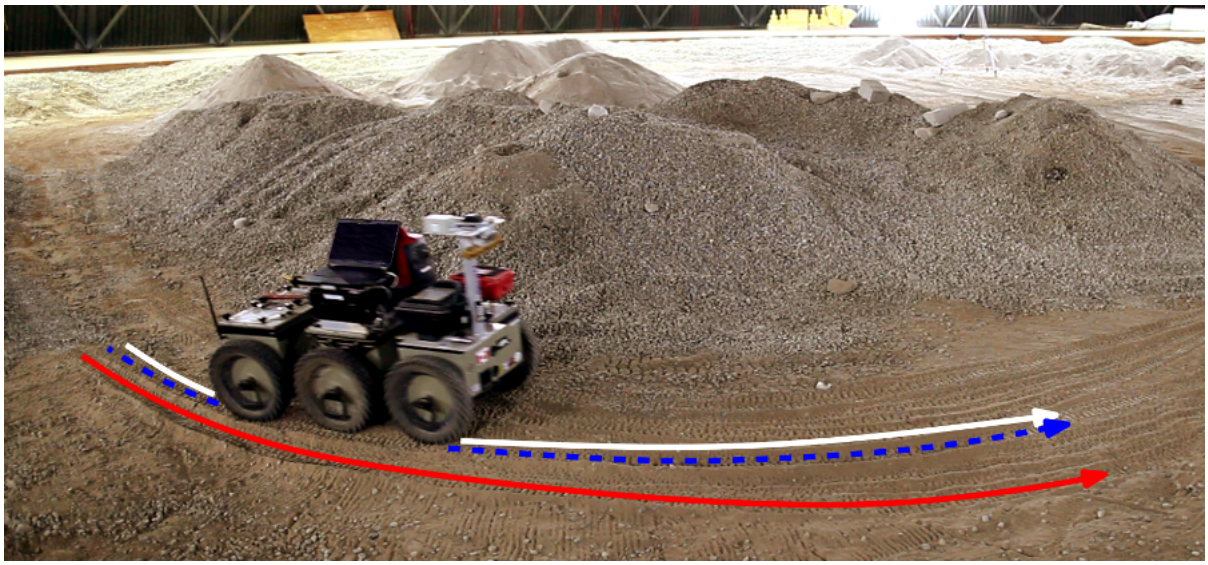


Figure 5.9: The second and third experiments focused on the algorithm's ability to learn unmodelled robot dynamics. Here we show the skid steered ROC6 robot driving at 0.6 m/s with learning enabled. The white line shows the desired trajectory (tire tracks), the red line shows a trajectory with learning disabled, while the dashed blue line shows a trajectory with learning enabled and reduced path-tracking errors.

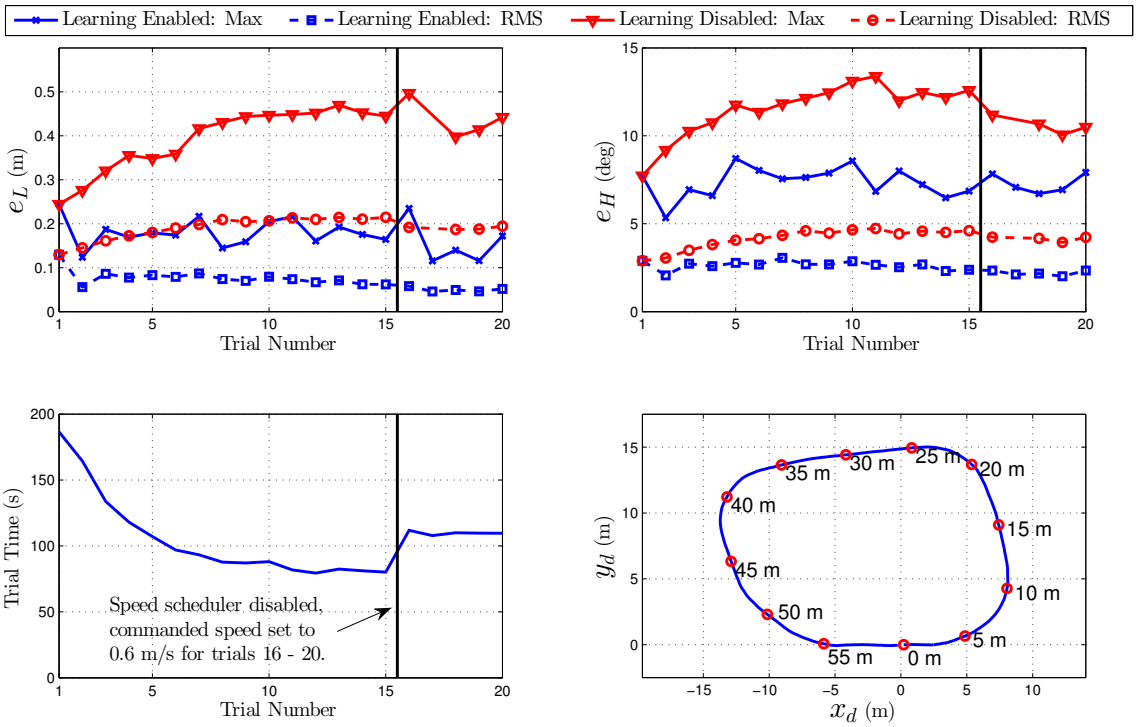


Figure 5.10: Here we show the reduction in maximum and RMS lateral and heading path-tracking errors vs. trial. Unlike the first experiment, the scheduled speeds for each trial were adjusted throughout the second experiment resulting in a range of travel times when tracking the loop-shaped, 60-m-long path.

set to a fixed value of 0.6 m/s. Unlike the ILC algorithm presented in Chapter 3, the LB-NMPC algorithm maintains a model including all learned experiences and thus is able to maintain low tracking errors through interpolation and extrapolation.

Figure 5.11 shows plots of path characteristics, scheduled speeds, and VT&R matched features vs. distance along the path. The path for the second experiment was mainly on level ground but included path curvatures up to 0.5 m^{-1}, suiting the capabilities of the ROC6 robot. The speed scheduler (Chapter 4) determined where along the path the system could tolerate higher speeds using experience from previous traversals, thus minimizing the travel time in sequential trials. In some sections of the path (e.g., at ~22 m), the system took up to 3 trials before safely increasing the scheduled speed. This does not necessarily mean the learned model in these sections had converged, but only that the path-tracking errors, the matched feature counts, and the control inputs were within the specified limits for the speed scheduler (Section 5.4.2). In general, the speed schedules resulted in the robot learning to drive the path faster, increasing speeds from 0.35 to 1.0 m/s. Sections of the path with poor lighting and high curvature, such

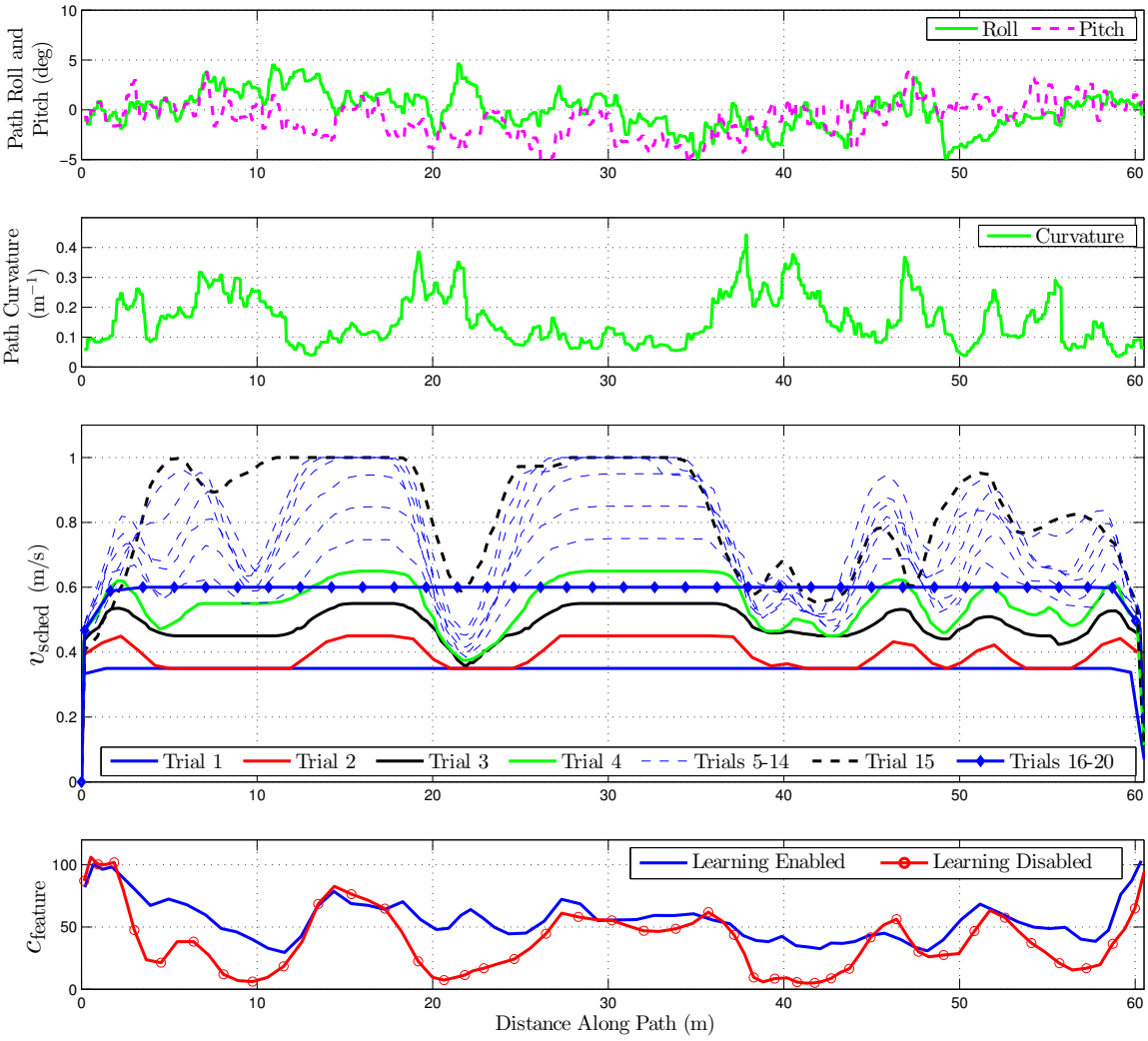


Figure 5.11: The test path for experiment 2 was mainly on level ground, but included path curvatures up to $0.5\,\text{m}^{-1}$. Here we also show the scheduled speeds, $v^{(j)}_{\text{sched}}$, for trials 1 through 20, and the VT&R matched feature counts, $c^{(j)}_{\text{feature}}$, for trial 15.

as at 10, 20, and 40 m along the path, had relatively low VT&R matched features. In these sections, the speed scheduler suggested increased speeds, though not as high as sections with good lighting and low curvature, such as at 15 or 30 m along the path. Further, with learning enabled and reduced path-tracking errors, the average number of matched features was increased from 38.33 to 55.77. Since the VT&R localization algorithm depends on matching features between the live-view and teach-pass view, an increase in matches tends to result in an increase in the localization reliability for the vision-based mapping and localization system.

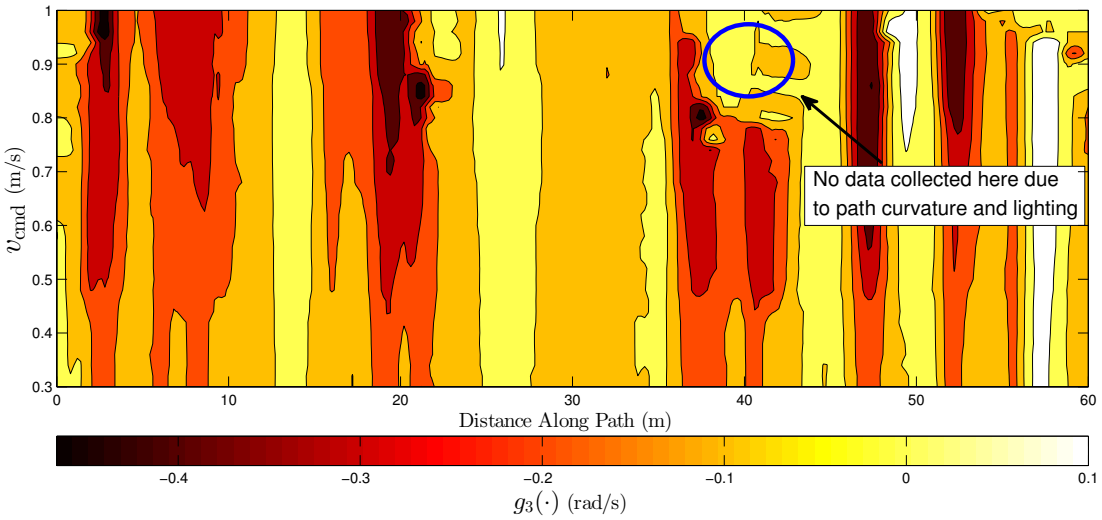


Figure 5.12: Here we show the learned values for the heading rate disturbance (i.e., the third element of $\mathbf{g}(\cdot)$) vs. commanded speed and distance along the path. Above 0.8 m/s, 40 m along the path (blue ellipse), there is very little data and the model is untrustworthy (Figure 5.13).

Figure 5.12 shows the learned model output vs. distance along the path and commanded speed. As previously mentioned, we model disturbances as a GP based on input-output data collected during previous trials. Even though our system collects discrete measurements of the underlying disturbance function, it is able to continuously interpolate and extrapolate from the data. In this way, we are able to maximize the information extracted from all gathered experience. In the second experiment, the system had collected roughly 20,000 observations for the learned model, retaining only 5,000 observations after 20 trials based on our experience management scheme.

Furthermore, by modelling disturbances as a GP, we are able to model both the mean and uncertainty of model discrepancies. For example, the system was unable to travel faster than 0.8 m/s at 40 m along the path due to the path's curvature. As a result, the system was not able to collect experience above 0.8 m/s for this section of the path and the resulting modelled disturbance is close to zero with relatively high uncertainty (Figure 5.13). However, the focus of our work on LB-NMPC was to investigate a real-time learning-based controller capable of generalization and as such, the algorithm presented here does not make use of the modelled uncertainty. As will be shown in Chapters 6 and 7, we can leverage the ability to predict the mean and uncertainty of disturbances in order generate high-performance, robust learning-based controllers.

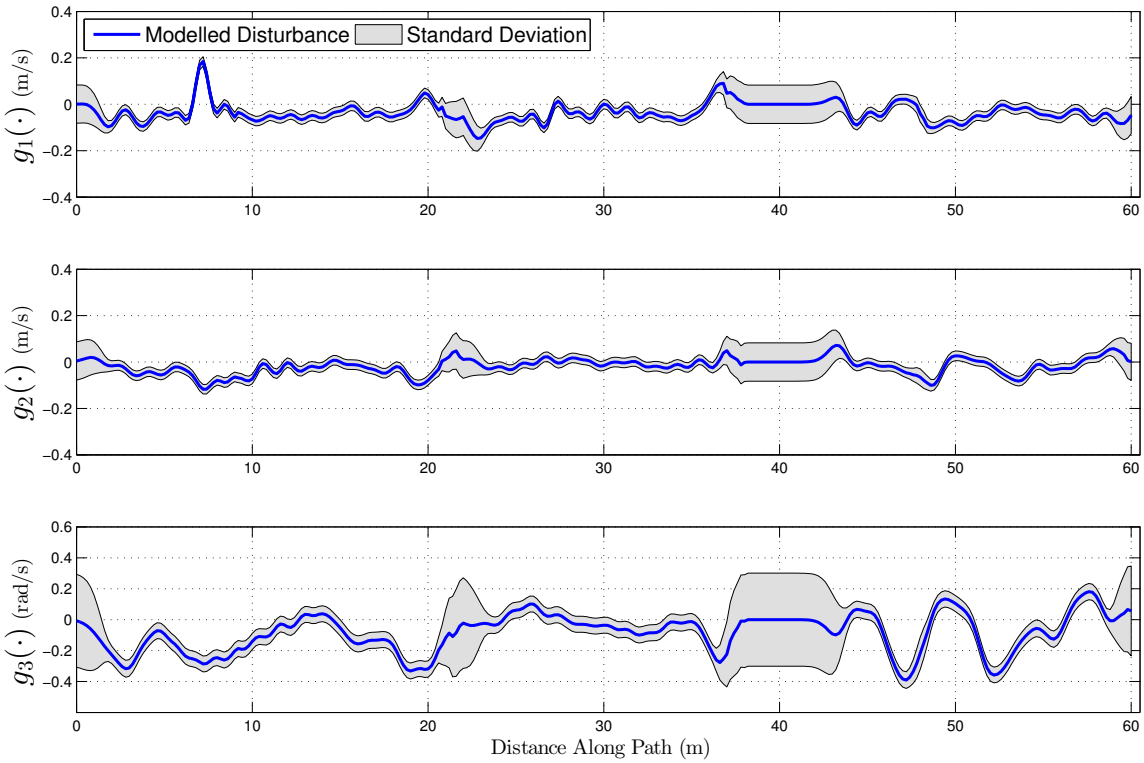


Figure 5.13: Modelled disturbances, $\mathbf{g}(\cdot) = (g_1(\cdot), g_2(\cdot), g_3(\cdot))$, for $v_{\text{cmd}} = 0.9\,\text{m/s}$. With no experience above 0.8 m/s, 40 m along the path, the modelled disturbance is zero and relatively uncertain.

5.4.5 Experiment 3: Learning to Follow a Path at Increasing Speeds with an Ackermann-steered Robot

In the third experiment, the 600 kg Ackermann-steered robot autonomously travelled the length of a 100-m-long path demonstrating the ability of the disturbance model to learn kinematics and dynamics of a significantly different mass and robot design (Figure 5.14). Figure 5.15 shows plots of the overall travel time, and maximum and RMS path-tracking errors vs. trial number. As in experiment 2, this test also used the speed scheduler presented in Chapter 4 to compute time-optimal schedules based on experience. Thus, as path-tracking errors were reduced by the learning-based controller, the scheduler increased the speed until the maximum allowed speed. Specifically, the LB-NMPC algorithm reduced the lateral and heading errors by more than 50% over the course of the 20 trials while learning disturbances at speeds ranging from 0.5 to 1.2 m/s.

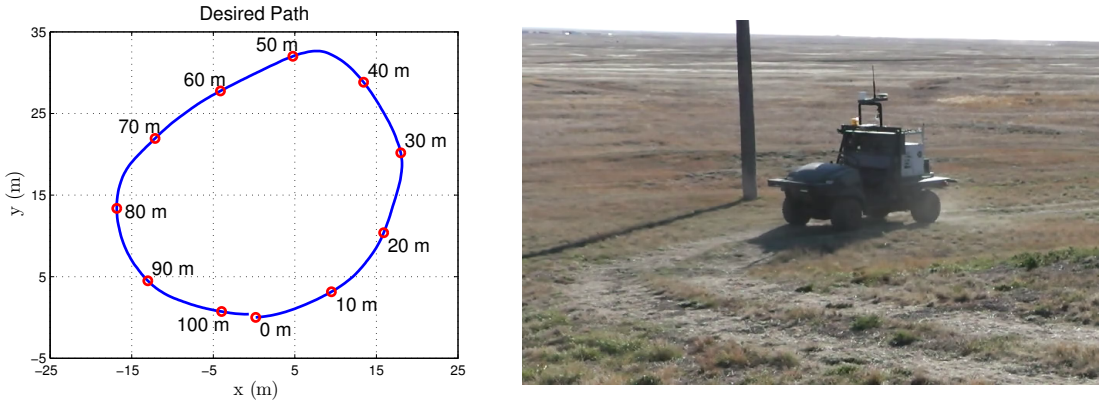


Figure 5.14: (Left) The desired path for experiment 3 formed a large loop. (Right) In this case, we tested with a large 600 kg, Ackermann-steered robot repeating the 100-m-long path. Like the previous experiments, the nominal model used by the LB-NMPC algorithm was a unicycle model, demonstrating the algorithm's ability to be applied to robots with significantly different designs.

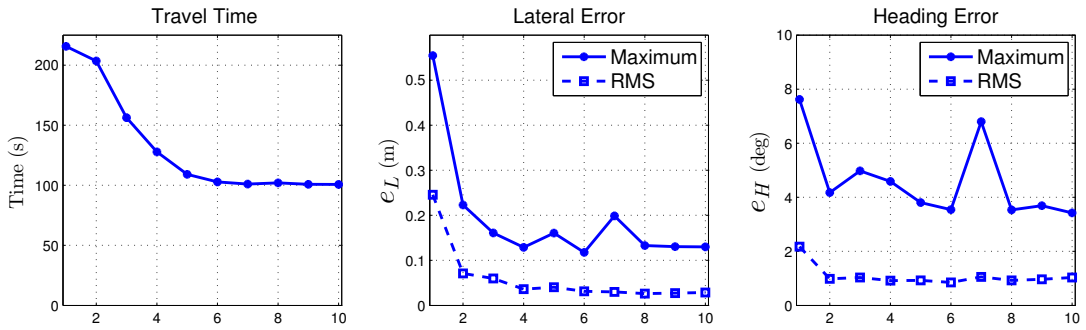


Figure 5.15: Over the course of 10 trials, the LB-NMPC algorithm reduced the lateral and heading path-tracking errors by over 50%, while simultaneously learning to drive at faster speeds around the desired path. The desired speeds were provided by the automated speed scheduler (Chapter 4).

Figure 5.16 shows plots of path characteristics, scheduled speed and path-tracking errors vs. distance along the path. The path included turns at 30, 48, and 75-95 m along the path that also corresponded with the locations on the path that took the most number of trials to reach the maximum allowed speed. While handling increased speeds, the LB-NMPC algorithm also reduced path-tracking errors over the course of the 10 trials.

This last experiment highlighted the need for work on controller robustness. Between 85 and 90 m along the path, in all trials of experiment 3, the results showed a sharp change in path-tracking errors. For example, in trial 1, the VT&R state estimate produced a

step-change in lateral path-tracking error of ∼25 cm in a single time-step (Figure 5.16). In reality the robot made no such movement. This artificial motion estimate was triggered by what Furgale and Barfoot (2010) called a 'teach pass failure', resulting in a discontinuity in the state estimate during relocalization. In this case, the LB-NMPC algorithm treated the side-step as a modelling error and learned to turn in anticipation of

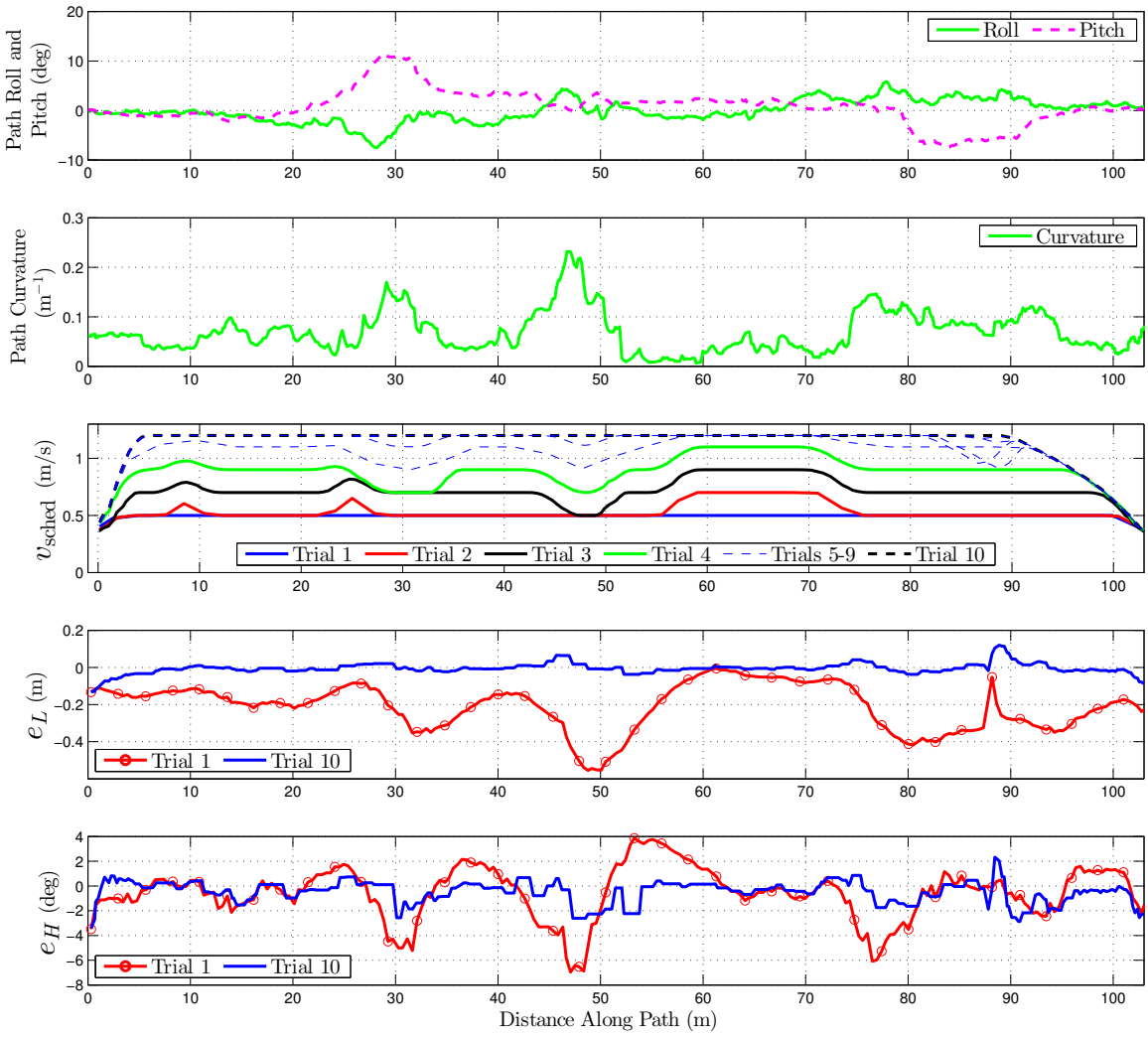


Figure 5.16: The path for the third experiment formed a large loop with turns at 30, 45, and 75 m along the path. As in experiment 2, the robot learned to drive the path at a range of speeds, from 0.5 to 1.2 m/s, generated by the automated speed scheduler. At around 50 m along the path, it took three trials before path-tracking errors were reduced sufficiently such that the scheduled speed could be increased.

the (artificial) disturbance, thereby causing subsequent (real) path-tracking errors. While practical state estimation algorithms should avoid providing faulty estimates, the stakes are higher with learning algorithms that are capable of inadvertently incorporating such outlier measurements into the learned model and then acting on incorrect data. This is one motivation for our experience management scheme, which forgets experiences over time.

5.5 Discussion

The ILC algorithm presented in Chapter 3 offered an efficient and effective method of improving path-tracking through experience. It satisfied two of our desired qualities. Namely, it offered the opportunity for real-time inputs over long-distance paths since the feedforward signal is computed offline. However, the algorithm is not well suited to flexible mobile robot operation since only a single feedforward signal can be maintained. In this chapter, we presented LB-NMPC, a learning-based algorithm capable of providing real-time control over long-distance paths, anytime learning, and generalizing from learned data. In this way, the algorithm is able to maximize the value from training time relative to the ILC algorithm. However, as shown in experiment 3, the algorithm leaves three open questions: (i) controller robustness, (ii) the tradeoff between exploration exploitation, and (iii) convergence rates.

In general, our LB-NMPC algorithm is initialized with a known nominal model and learns the discrepancies between the known model and the actual robot behavior. Therefore by its very structure, the augmented process model used by our NMPC algorithm has varying levels of uncertainty while learning. Controller robustness, i.e. the capability of a controller to stabilize a system in spite of model uncertainty, is an open question for learning controllers in general (Schaal et al., 2010). In this work, we do not explicitly consider the robustness of the controller but focus on the practical application of LB-NMPC to mobile robots. However, having established the effectiveness of the LB-NMPC algorithm at reducing control errors with few *a priori* assumptions, Chapters 6 and 7 focus on techniques to leverage the uncertainty estimates provided by our GP-based disturbance model in robust control frameworks.

In Chapter 3, experimental results were presented based on ILC where a single speed was selected prior to each trial manually. This led to the development of the experience-based speed scheduler in Chapter 4 that produces a speed schedule for each trial (i.e.,

a desired speed for each vertex along the path based on previous tracking errors and localization reliability). In this work, the use of the automated speed scheduler effectively begins to address the exploration vs. exploitation problem. Specifically, the algorithm chooses when to exploit known experiences to reduce tracking errors and when to schedule higher speeds to explore new states. However, future work might focus on determining how much to increase speeds. More over, an effective strategy to improve performance might also involve exploring a variety of path offsets, such as a specific (safe) level of tracking error, in order to improve the model of robot behavior around the path.

Finally, determination of convergence rates is also an open problem in model-based learning controllers (Nguyen-Tuong and Peters, 2011). Unlike techniques such as Iterative Learning Control (Bristow et al., 2006; Ahn et al., 2007), which assume identical initial conditions and desired trajectories for all trials in order to make claims on convergence rates, model-based learning controllers, such as the work presented here, address a more general problem trying to learn with arbitrary initial conditions, paths, and speed schedules. This enables a more flexible robot use since it is able to learn more than one path at more than one speed. However, it also presents essentially a sporadic approach to learning, in that it is not guaranteed when or if a state will be revisited for continued learning. Furthermore, convergence rates are complicated by the evolution of the environment caused by the robot's activity. For example, repeating the same path caused ruts to form which resulted in a change in the disturbances affecting the nominal process model. This was also a motivation in using only the most recent observations (Section 5.3.2).

5.6 Conclusion

In summary, the major contribution in this chapter is a Learning-based Nonlinear Model Predictive Control (LB-NMPC) algorithm for a path-repeating, mobile robot negotiating large-scale, GPS-denied, outdoor environments. The goal is to reduce path-tracking errors using real-world experience instead of pre-programming accurate analytical models of wheel-terrain interaction, terrain topography, or robot dynamics. The LB-NMPC controller is based on a fixed, simple process model and a learned disturbance model. Disturbances effectively represent measured discrepancies between the given nominal model and the observed system behaviour. Modelling disturbances as a GP enables the algorithm to learn complex nonlinear model discrepancies and to generalize to novel situ-

ations. Unlike our work on ILC where learned signals were indexed only by the distance along a desired path, here disturbances are modelled as a function of the system state (i.e., pose and velocity) and input, enabling the system to learn disturbances for many paths and speed schedules. Furthermore, we present a novel approach where disturbance predictions are computed based on a sliding window of training data. This enables real-time operation over long-distance paths. Localization for the controller is provided by an on-board, Visual Teach & Repeat mapping and navigation system.

Three experiments on three significantly different robots, including over 3 km of travel on challenging paths, demonstrated the system's ability to handle unmodelled terrain and robot dynamics, and also to interpolate and extrapolate from learned disturbances. In the second and third experiments, the experience-based speed scheduler addressed the classic exploration vs. exploitation trade-off balancing speed, path-tracking errors, and localization reliability. Effectively, the speed scheduler greedily increased the scheduled speed wherever it could, a bias towards exploration and quick decreases in travel time. The LB-NMPC approach proved to be flexible and effective at reducing path-tracking errors and increasing the reliability of the localization system. Even beginning with only the simple unicycle model, the algorithm was capable of being deployed to multiple platforms where it learned to reduce vehicle- and trajectory-specific path-tracking errors using experience.

Chapter 6

Robust Min-Max Learning-based Model Predictive Control

6.1 Introduction

In this chapter, we present a robust Min-Max LB-NMPC (MM-LB-NMPC) algorithm for a path-repeating mobile robot. Previously, we presented LB-NMPC as an algorithm capable of providing real-time control inputs over long-distance paths while enabling anytime learning and generalization from learned experiences. However, in completing the experimental work, we identified controller robustness as a fourth desired quality for mobile robot learning-based controllers. In practice, the LB-NMPC algorithm begins with a simple *a priori* model and learns model discrepancies between the known model and reality. For practical operation, learning-based algorithms should guarantee controller robustness throughout the learning process since the learned model will be quite uncertain at times.

Robust control maintains stability and performance for a fixed amount of model uncertainty but can be conservative since the model is not updated online. Learning-based control, on the other hand, uses data to improve the model over time but is not typically guaranteed to be robust throughout the process. In this work, we present a extension to our LB-NMPC algorithm. The goal is to merge the best of both worlds: robust, conservative control reducing worst-case tracking errors during initial trials when model uncertainty is high, converging to optimal control during later trials when model uncertainty is reduced. We leverage recent work on robust MPC where the cost function is modified to optimize for plausible scenarios as opposed to the nominal predicted sequence.

Figure 6.1: A Clearpath Husky robot autonomously negotiating challenging terrain with side slopes, inclines, and variable wheel traction. In practice, simple vehicle models rarely represent reality and limit the performance and stability of model-based, path-tracking controllers. In this work, we present a robust learning-based controller that automatically transitions from robust to optimal control throughout the process of learning to track a path.

Min-Max MPC maintains controller stability despite model uncertainty by altering the performance function to optimize for worst-case scenarios (Witsenhausen, 1968; Campo and Morari, 1987). The Min-Max approach was further developed by Scokaert and Mayne (1998). Bemporad et al. (2003) and Kerrigan and Maciejowski (2004) present algorithms that reduce the computation time of Min-Max MPC. However, these examples require *a priori* assumptions of bounded disturbances with uniform distributions, likely representing conservative estimates of the true probability distributions. In an effort to reduce conservativeness, Raimondo et al. (2009) present a nonlinear Min-Max algorithm that separates state-dependent and state-independent disturbances. We also employ a Min-Max approach. However, as opposed to these examples, our approach models disturbances as a GP and as a result we consider worst-case scenarios bounding the nominal 3σ confidence region. Moreover, we use learning to reduce uncertainty over time, thereby smoothly transitioning from conservative, robust control to optimal, high-performance control using experience. This work represents the first example, to our knowledge, of MM-LB-NMPC.

In the case that disturbances are not bounded, but rather are described by a known probability distribution, Scenario MPC considers a finite set of randomly sampled sequences (Blackmore, 2006; Matsuko and Borrelli, 2012). For example, Schildbach et al.

(2014) propose an approach where the performance function is optimized considering all sampled disturbance sequences simultaneously. In order to ensure that the algorithm is computationally tractable, they exploit a key structure of the optimal control problem leading to a substantial reduction of the problem dimension. Moreover, they provide insights into the practice of *a posteriori* outlier sample removal that typically occurs after the outcome of all samples has been observed. However, Scenario MPC relies on a (typically) large number of randomly sampled state sequences over the prediction horizon in order to adequately represent future behavior. Calafiore and Fagiano (2013) leverage recent results in optimization to provide an explicit link between the number of required samples and the probability of success. In our work, we assume disturbances are normally distributed and use a Sigma-Point Transform (Julier and Uhlmann, 2004) to predict the mean and uncertainty of future robot behavior. This represents the first work to our knowledge to use a Sigma-Point Transform to predict the mean and uncertainty of state sequences for MM-MPC. Unlike Scenario MPC, our algorithm relies on a small number of worst-case scenarios bounding the nominal 3σ confidence region. The resulting algorithm is an efficient and effective extension from our LB-NMPC algorithm, enabling robust control with relatively little increase in computation time.

Like our previous work on LB-NMPC, the robust MM-LB-NMPC algorithm is based on a simple *a priori* process model and a learned, non-parametric disturbance model. Disturbances are also modelled as a Gaussian Process (Rasmussen, 2006) based on previous experience as a function of state, input, and other relevant variables. However, in this work we predict both the mean and uncertainty of disturbances affecting the *a priori* process model. As mentioned, we use a Sigma-Point Transform to efficiently compute the mean and variance of the nominal predicted sequence given the two-component, learned, stochastic model. The MM-LB-NMPC cost function is then optimized for the worst-case sequence bounding the nominal 3σ confidence region. As the learned model becomes less uncertain with experience, worst-case sequences transition towards accurate nominal sequences resulting in optimal control. Finally, we demonstrate the robust, learning controller on a 50 kg Clearpath Husky robot and show reductions of worst-case path-tracking errors by up to 30% and a clear transition from robust control towards optimal control with only a 5% increase in computation time.

6.2 Mathematical Formulation

NMPC finds a sequence of control inputs that optimizes the plant behavior over a prediction horizon based on the current state. The first control input in the optimal sequence is then applied to the system, resulting in a new system state. The entire process is then repeated at the next sample time for the new system state. In traditional NMPC implementations, the process model is specified *a priori* and remains unchanged during operation. In Chapter 5 we presented LB-NMPC, where we augmented a simple process model with the mean of an experience-based disturbance model. Effectively, the controller used experience to reduce path-tracking errors, compensating for effects not captured by the simple process model. In this work, we incorporate both the disturbance mean and uncertainty into the NMPC algorithm by modifying the cost function to optimize for worst-case scenarios, resulting in an efficient robust extension that reduces the worst-case errors (Figure 6.2).

6.2.1 Min-Max Nonlinear Model Predictive Control

Consider the following stochastic, learned process model,

$$\mathbf{x}_{k+1} = \overbrace{\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)}^{a\ priori\ \text{model}} + \overbrace{\mathbf{g}(\mathbf{a}_k)}^{\text{learned model}}, \tag{6.1}$$

with a normally distributed system state, $\mathbf{x}_k \sim \mathcal{N}(\bar{\mathbf{x}}_k, \mathbf{\Sigma}_k)$, $\mathbf{x}_k \in \mathbb{R}^n$, disturbance dependency, $\mathbf{a}_k \in \mathbb{R}^p$, and control input, $\mathbf{u}_k \in \mathbb{R}^m$, all at time k. The models $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ are nonlinear models: $\mathbf{f}(\cdot)$ is a simple, *a priori* vehicle model and $\mathbf{g}(\cdot)$ is an (initially unknown) disturbance model representing discrepancies between the nominal model and the actual system behavior. As in Chapter 5, disturbances are modelled as a Gaussian Process (Section 5.2.2), thus $\mathbf{g}(\cdot)$ is normally distributed, $\mathbf{g}(\cdot) \sim \mathcal{N}(\boldsymbol{\mu}(\cdot), \mathbf{\Sigma}_{\text{gp}}(\cdot))$. In our previous work, we showed that $\mathbf{g}(\cdot)$ could be used to learn higher-order dynamics by including historic states in the disturbance dependency. However, for simplicity, we assume for now that $\mathbf{a}_k = (\bar{\mathbf{x}}_k, \mathbf{u}_k)$.

As previously mentioned, the goal of NMPC is to find a set of controls that optimizes the plant behavior over a given prediction horizon. To this end, we define the cost

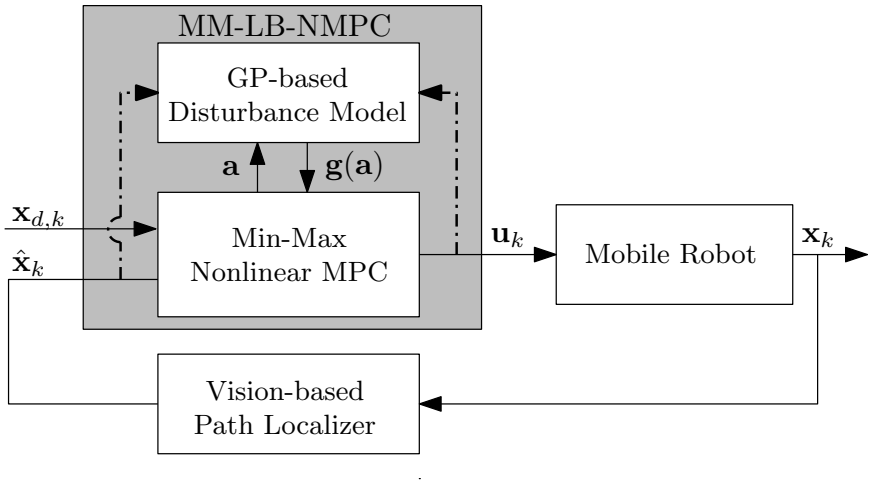


Figure 6.2: The controller is composed of two components: (i) the robust, path-tracking, Min-Max NMPC algorithm, and (ii) the GP-based Disturbance Model, providing experience-based disturbance estimates.

function to be minimized over the next K time-steps as

$$J(\check{\mathbf{x}}, \mathbf{u}) = (\mathbf{x}_d - \check{\mathbf{x}})^T \mathbf{Q} \, (\mathbf{x}_d - \check{\mathbf{x}}) + \mathbf{u}^T \mathbf{R} \, \mathbf{u}, \tag{6.2}$$

where $\mathbf{Q}$ is positive semi-definite, $\mathbf{R}$ is positive definite, $\mathbf{u}$ is a sequence of inputs, $\mathbf{u} = (\mathbf{u}_k, \ldots, \mathbf{u}_{k+K-1})$, $\mathbf{x}_d$ is a sequence of desired states, $\mathbf{x}_d = (\mathbf{x}_{d,k+1}, \ldots, \mathbf{x}_{d,k+K})$, and $\check{\mathbf{x}}$ is a sequence of predicted states, $\check{\mathbf{x}} = (\check{\mathbf{x}}_{k+1}, \ldots, \check{\mathbf{x}}_{k+K})$. In our work on LB-NMPC, the objective was optimized for the mean of the nominal predicted sequence, $\check{\mathbf{x}} = \bar{\mathbf{x}}$, where $\mathbf{x} = (\mathbf{x}_{k+1}, \ldots, \mathbf{x}_{k+K})$ is a sequence of uncertain predicted states, and $\bar{\mathbf{x}}$ is the sequence of mean values based on $\mathbf{x}$. In this work, the objective is optimized for the worst-case sequence given the uncertainty in the learned model. Specifically, we define 2^n worst-case scenarios, $\check{\mathbf{x}}^{\{l\}}$, $l = 1 \ldots 2^n$, as sequences bounding the nominal 3σ confidence region. Finally, the optimal control sequence is given by

$$\mathbf{u}_{\text{opt}} = \arg \min_{\mathbf{u}} \max_{l} J(\check{\mathbf{x}}^{\{l\}}, \mathbf{u}). \tag{6.3}$$

The addition of the 'max' in (6.3) is the extension from our work on LB-NMPC: the algorithm now takes into account the worst-case boundary sequence, limiting the worst-case errors and guaranteeing stability for large model uncertainties. Moreover, there is an automatic transition as uncertainty decreases, from robust control with an uncertain model, to optimal control with a rich and accurate model.

Algorithm 1: MM-LB-NMPC

Data: $\hat{\mathbf{x}}_k$, $\mathbf{x}_d$, and $\mathbf{u}_{\text{init}}$
Result: $\mathbf{u}_{\text{opt}}$
1 initialization: $\tilde{\mathbf{u}} = \mathbf{u}_{\text{init}}$;
2 **while** $\|\delta\mathbf{u}\| > \alpha$ **do**
3 Compute $\mathbf{x}$ given $\tilde{\mathbf{u}}$ and (6.1);
4 Compute boundary sequences;
5 Find worst-case boundary sequence;
6 Linearize (6.2) around worst-case sequence and solve for $\delta\mathbf{u}$;
7 Update control, $\tilde{\mathbf{u}} \leftarrow \tilde{\mathbf{u}} + \delta\mathbf{u}$;

Since both our process model and disturbance model are nonlinear, the optimal control sequence, $\mathbf{u}_{\text{opt}}$, is found iteratively (Algorithm 1) using a nonlinear optimization technique. In this work, we use unconstrained Gauss-Newton minimization (Nocedal and Wright, 1999).

At each time-step, we begin with the current state, $\mathbf{x}_k = \hat{\mathbf{x}}_k$, provided by the vision-based localization system, and an initial guess for the optimal control input sequence, $\tilde{\mathbf{u}}$, such as the sequence of optimal inputs computed in the previous time-step (Algorithm 1, Step 1). We then use a Sigma-Point Transform (Section 6.2.2) to compute the nominal sequence (Algorithm 1, Step 3),

$$\begin{aligned}\begin{bmatrix} \mathbf{x}_{k+1} \\ \mathbf{x}_{k+2} \\ \vdots \\ \mathbf{x}_{k+K} \end{bmatrix} &= \begin{bmatrix} \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{g}(\mathbf{x}_k, \mathbf{u}_k) \\ \mathbf{f}(\mathbf{x}_{k+1}, \mathbf{u}_{k+1}) + \mathbf{g}(\mathbf{x}_{k+1}, \mathbf{u}_{k+1}) \\ \vdots \\ \mathbf{f}(\mathbf{z}_{k+K-1}, \mathbf{u}_{k+K-1}) + \mathbf{g}(\mathbf{x}_{k+K-1}, \mathbf{u}_{k+K-1}) \end{bmatrix} \\ &= \mathbf{h}(\mathbf{x}_k, \mathbf{u}), \end{aligned} \tag{6.4}$$

where the predicted states, $\mathbf{x}_{k+i}$, $i = 1 \ldots K$, include both mean and uncertainty. During situations when the robot has little experience, such as during the first trial, the uncertainty of predicted sequences will be relatively high. As the robot collects experience, uncertainty will be reduced. We then compute 2^n worst-case sequences bounding the nominal 3σ confidence region (Algorithm 1, Step 4). Assuming for now $n = 3$, and

defining $\boldsymbol{\sigma}_{k+i} = (\sqrt{\boldsymbol{\Sigma}_{k+i}(1,1)}, \ldots, \sqrt{\boldsymbol{\Sigma}_{k+i}(3,3)})$, and

$$\begin{aligned}
\boldsymbol{\Gamma}_{\text{sign}}^{\{1\}} &= \text{diag}(1,1,1), & \boldsymbol{\Gamma}_{\text{sign}}^{\{5\}} &= \text{diag}(-1,1,1),\\
\boldsymbol{\Gamma}_{\text{sign}}^{\{2\}} &= \text{diag}(1,1,-1), & \boldsymbol{\Gamma}_{\text{sign}}^{\{6\}} &= \text{diag}(-1,1,-1),\\
\boldsymbol{\Gamma}_{\text{sign}}^{\{3\}} &= \text{diag}(1,-1,1), & \boldsymbol{\Gamma}_{\text{sign}}^{\{7\}} &= \text{diag}(-1,-1,1),\\
\boldsymbol{\Gamma}_{\text{sign}}^{\{4\}} &= \text{diag}(1,-1,-1), & \boldsymbol{\Gamma}_{\text{sign}}^{\{8\}} &= \text{diag}(-1,-1,-1),
\end{aligned} \tag{6.5}$$

then $\check{\mathbf{x}}_{k+i}^{\{l\}} = \bar{\mathbf{x}}_{k+i} + \boldsymbol{\Gamma}_{\text{sign}}^{\{l\}} 3\boldsymbol{\sigma}_{k+i}$, $i = 1 \ldots K$. The worst-case scenario is then determined by evaluating the cost of each boundary sequence (Algorithm 1, Step 5),

$$l^* = \arg\max_{l} J(\check{\mathbf{x}}^{\{l\}}, \tilde{\mathbf{u}}). \tag{6.6}$$

We now linearize the cost function (6.2) considering the current solution, $\mathbf{u} = \tilde{\mathbf{u}} + \delta\mathbf{u}$, and the worst-case scenario,

$$\begin{aligned}
\check{\mathbf{x}} &= \check{\mathbf{x}}^{\{l^*\}} + \delta\mathbf{x}\\
&\approx \check{\mathbf{x}}^{\{l^*\}} + \mathbf{H}\,\delta\mathbf{u},
\end{aligned} \tag{6.7}$$

where $\mathbf{H}$ is the block-Jacobian of (6.4) with respect to $\mathbf{u}$,

$$\mathbf{H} = \left.\frac{\partial \mathbf{h}(\mathbf{x}_k, \mathbf{u})}{\partial \mathbf{u}}\right|_{\hat{\mathbf{x}}_k, \tilde{\mathbf{u}}}. \tag{6.8}$$

Substituting (6.7) and $\mathbf{u} = \tilde{\mathbf{u}} + \delta\mathbf{u}$ into (6.2) results in $J(\cdot)$ being quadratic in $\delta\mathbf{u}$ (Algorithm 1, Step 6). We can find the value of $\delta\mathbf{u}$ that minimizes $J(\check{\mathbf{x}}, \mathbf{u})$, update our control input (Algorithm 1, Step 7),

$$\tilde{\mathbf{u}} \leftarrow \tilde{\mathbf{u}} + \beta\,\delta\mathbf{u}, \tag{6.9}$$

and iterate to convergence, $\|\delta\mathbf{u}\| < \alpha$, with tuned value, α. In this approach to solving the min-max optimization problem, the step-size β must be carefully selected so as to minimize chatter caused by switching between scenarios. Finally, in accordance with NMPC, we apply the resulting control input for one time-step and start all over at the next time-step.

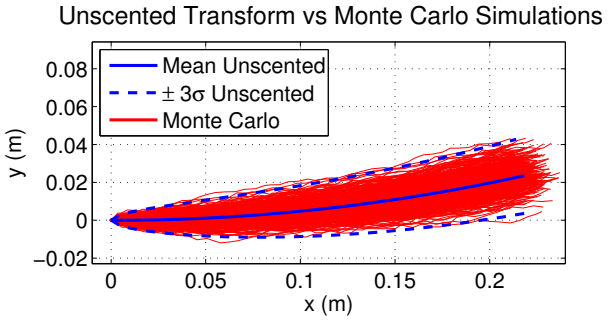


Figure 6.3: Here we show the lateral 3σ boundaries of the mean sequence. Since our model uncertainty is normally distributed, we use a Sigma-Point Transform to compute the mean and uncertainty of a predicted state sequence. However, unlike Scenario MPC, where many scenarios are sampled in order to find boundary sequences (red), the Sigma-Point Transform efficiently estimates the 3σ confidence region (dashed blue).

6.2.2 Sigma-Point Transform

In our previous work, the cost function is optimized based on the mean of the nominal state sequence. In this work, the cost function is optimized for the worst-case sequence bounding the nominal 3σ confidence region. Since the system state, $\mathbf{x}_k$, is normally distributed and the process model (6.1) is nonlinear, we use a Sigma-Point Transform (Julier and Uhlmann, 2004) to iteratively predict the nominal state sequence, $\mathbf{x}$, given $\mathbf{u}$ and an initial state with known mean and uncertainty, $\hat{\mathbf{x}}_k \sim \mathcal{N}(\bar{\mathbf{x}}_k, \boldsymbol{\Sigma}_k)$, (Figure 6.3). At a high level, the approach is to compute a set of Sigma Points representing the state distribution at time k. We then pass the Sigma Points through the nonlinear process model and recompute the state distribution for time $k+1$. As a result, we define an initial state, $\mathbf{z}_k = (\bar{\mathbf{x}}_k, \boldsymbol{\mu}(\mathbf{a}_k)) \in \mathbb{R}^{2n}$, representing the mean state and disturbance at time k. We also define the corresponding uncertainty, $\mathbf{P}_k = \text{diag}(\boldsymbol{\Sigma}_k, \boldsymbol{\Sigma}_{\text{gp}}(\mathbf{a}_k))$. We compute $4n+1$ sigma points, $\mathcal{Z}_{k,i} = (\mathcal{X}_{k,i}, \mathcal{M}_{k,i})$, where $\mathcal{X}_{k,i}$ and $\mathcal{M}_{k,i}$ are the sigma points of $\mathbf{x}_k$ and $\boldsymbol{\mu}(\mathbf{a}_k)$,

$$\mathcal{Z}_{k,0} = \mathbf{z}_k \tag{6.10}$$

$$\mathcal{Z}_{k,i} = \mathbf{z}_k + \sqrt{2n+\gamma}\,\text{col}_i\,\mathbf{S}_k, \quad i = 1 \dots 2n \tag{6.11}$$

$$\mathcal{Z}_{k,i+2n} = \mathbf{z}_k - \sqrt{2n+\gamma}\,\text{col}_i\,\mathbf{S}_k, \quad i = 1 \dots 2n \tag{6.12}$$

where $\mathbf{S}_k\mathbf{S}_k^T = \mathbf{P}_k$ with $\mathbf{S}_k$ derived from the Cholesky decomposition of $\mathbf{P}_k$, $\text{col}_i\,\mathbf{S}_k$ is the ith column of $\mathbf{S}_k$, and γ is a tuning parameter. The sigma points are then passed through

the nonlinear model,

$$\mathcal{X}_{k+1,i} = \mathbf{f}(\mathcal{X}_{k,i}, \mathbf{u}_k) + \mathcal{M}_{k,i}, \quad i = 0 \dots 4n, \tag{6.13}$$

where $\mathbf{f}(\cdot)$ is our *a priori* vehicle model. Finally, we combine the sigma points into the predicted mean and uncertainty,

$$\bar{\mathbf{x}}_{k+1} = \frac{1}{2n+\gamma}\left(\gamma\,\mathcal{X}_{k+1,0} + \frac{1}{2}\sum_{i=1}^{4n}\mathcal{X}_{k+1,i}\right) \tag{6.14}$$

$$\begin{aligned}\mathbf{\Sigma}_{k+1} = \frac{1}{2n+\gamma}\Bigg(&\gamma\,(\mathcal{X}_{k+1,0} - \bar{\mathbf{x}}_{k+1})(\mathcal{X}_{k+1,0} - \bar{\mathbf{x}}_{k+1})^T \\ &+ \frac{1}{2}\sum_{i=1}^{4n}(\mathcal{X}_{k+1,i} - \bar{\mathbf{x}}_{k+1})(\mathcal{X}_{k+1,i} - \bar{\mathbf{x}}_{k+1})^T\Bigg).\end{aligned} \tag{6.15}$$

This process is repeated K times, until the complete nominal sequence, $\mathbf{x}$, is computed. In this way, the 3σ confidence region accounts for uncertainty arising from both localization and modelling.

6.3 Implementation

The work presented in this chapter is an extension of the insights and work presented in Chapter 5. As in LB-NMPC, disturbances are modelled as a GP enabling anytime learning and generalization from learned experiences (Section 5.2.2). However, in this work we compute both the mean and uncertainty of disturbances affecting the nominal process model. This represents only a small increase in computation time since the expensive data products required to compute the uncertainty, namely the inverse of the Gram matrix, $\mathbf{K}^{-1}$, are already generated in computing the mean disturbance. Moreover, when modelling disturbances as a GP, it is necessary to reduce computational complexity by carefully selecting support points. As a result, we continue to use the experience management scheme presented in Section 5.3.2. Robots are also modelled as unicycle-type vehicles (5.26) with position, $\mathbf{x}_k = (x_k, y_k, \theta_k)$, calculated relative to the nearest path vertex by Euclidean distance, and velocity, $\mathbf{v}_k = (v_{\text{act},k}, \omega_{\text{act},k})$ (Figure 3.4). The commanded linear velocity is set to a desired speed leaving only the angular velocity, $\omega_{\text{cmd},k}$, for the NMPC algorithm to choose. However, unlike previous work, experimental results for the MM-LB-NMPC algorithm are based on a single speed (i.e., we do not use

the automated speed scheduler presented in (Chapter 4). Finally, we continue to use an extended disturbance dependency in practice, $\mathbf{a}_k = (\bar{\mathbf{x}}_k, \bar{\mathbf{v}}_{k-1}, \mathbf{u}_k, \mathbf{u}_{k-1})$, enabling the algorithm to learn higher-order disturbances in addition to kinematics. In practice, we approximate $\bar{\mathbf{v}}_k$ based on mean states,

$$\bar{v}_{\text{act},k-1} = \frac{\sqrt{(\bar{x}_k - \bar{x}_{k-1})^2 + (\bar{y}_k - \bar{y}_{k-1})^2}}{\Delta t}, \tag{6.16}$$

$$\bar{\omega}_{\text{act},k-1} = \frac{(\bar{\theta}_k - \bar{\theta}_{k-1})}{\Delta t}. \tag{6.17}$$

Effectively, this represents the same underlying model and disturbance dependency as included in LB-NMPC (Chapter 5). This reflects our goal of building on and improving the previous work by accounting for model uncertainty in the optimization process.

6.4 Experimental Results

6.4.1 Overview

We tested the MM-LB-NMPC algorithm using a 50 kg Clearpath Husky robot travelling at 0.5 m/s on a short, demonstrative path (Figure 6.4). The controller described in Section 6.2 was implemented and run in addition to the VT&R software (Chapter 2) on a Lenovo W530 laptop with an Intel 2.6 Ghz Core i7 processor with 16 GB of RAM. The camera used for localization was a Point Grey Bumblebee XB3 stereo camera. Since GPS was not available, the improvement due to the MM-LB-NMPC algorithm was quantified by the localization of the VT&R algorithm. The resulting real-time localization and path-tracking control signals were generated at approximately 10 Hz.

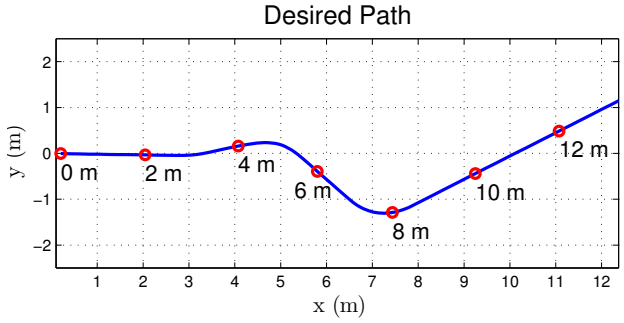


Figure 6.4: The test path for the MM-LB-NMPC algorithm. A short, demonstrative path was selected to highlight the improvements due to the Min-Max algorithm.

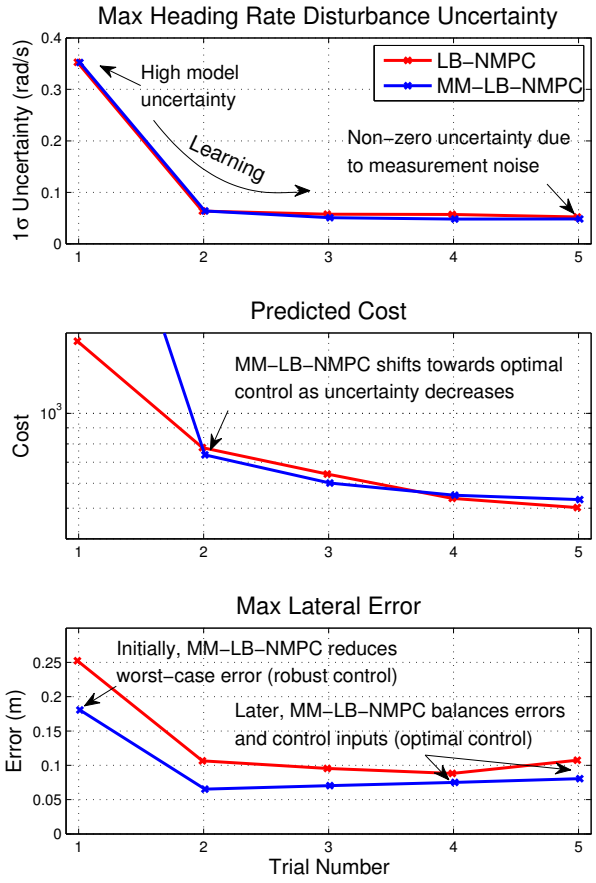


Figure 6.5: Here we show the procession of model uncertainty (e.g, the maximum heading rate disturbance uncertainty), predicted costs, and maximum lateral error over several trials. The plots show the automatic transition between robust control, when uncertainty is high and maximum errors are reduced significantly, to optimal control, when model uncertainty is low and the controller finds a balance between errors and control inputs.

6.4.2 Tuning Parameters

The performance of the system was primarily adjusted using the NMPC weighting matrices, $\mathbf{Q}$ and $\mathbf{R}$. We selected a 3:3:1 ratio balancing heading errors, position errors, and control inputs. Otherwise, the prediction horizon, $K = 10$, and the convergence criterion, $\alpha = 0.01$, were selected to enable online operation. The Sigma-Point scaling parameter, $\gamma = 2$, was selected to enable accurate prediction of mean and uncertainty.

6.4.3 Results

Over five trials, the 50 kg Husky robot autonomously travelled the 13-m-long path (Figure 6.4) and successfully reduced the worst-case lateral and heading errors by up to 30%.

Figure 6.5 highlights the procession from robust control, when model uncertainty was high, to optimal control, when the system had acquired experience and model uncertainty was reduced. In practice, the learned model uncertainty never goes to zero due to measurement noise. As a result, the MM-LB-NMPC algorithm shifted towards optimal control but ultimately struck a balance between robust and optimal control over time. Finally, the algorithm reduces errors due to non-repetitive noise, such as measurement noise, that the learning algorithm is incapable of predicting.

The measured disturbance affecting the heading rate of the robot peaked at approximately 0.5 rad/s, representing nearly 50% of the commanded input 7 m along the path (Figure 6.6). The general trend observed is that path-tracking errors were reduced over the entire path (Figure 6.7). However, the errors were not cancelled completely due to the optimality of the NMPC algorithm, which balances tracking errors and control inputs.

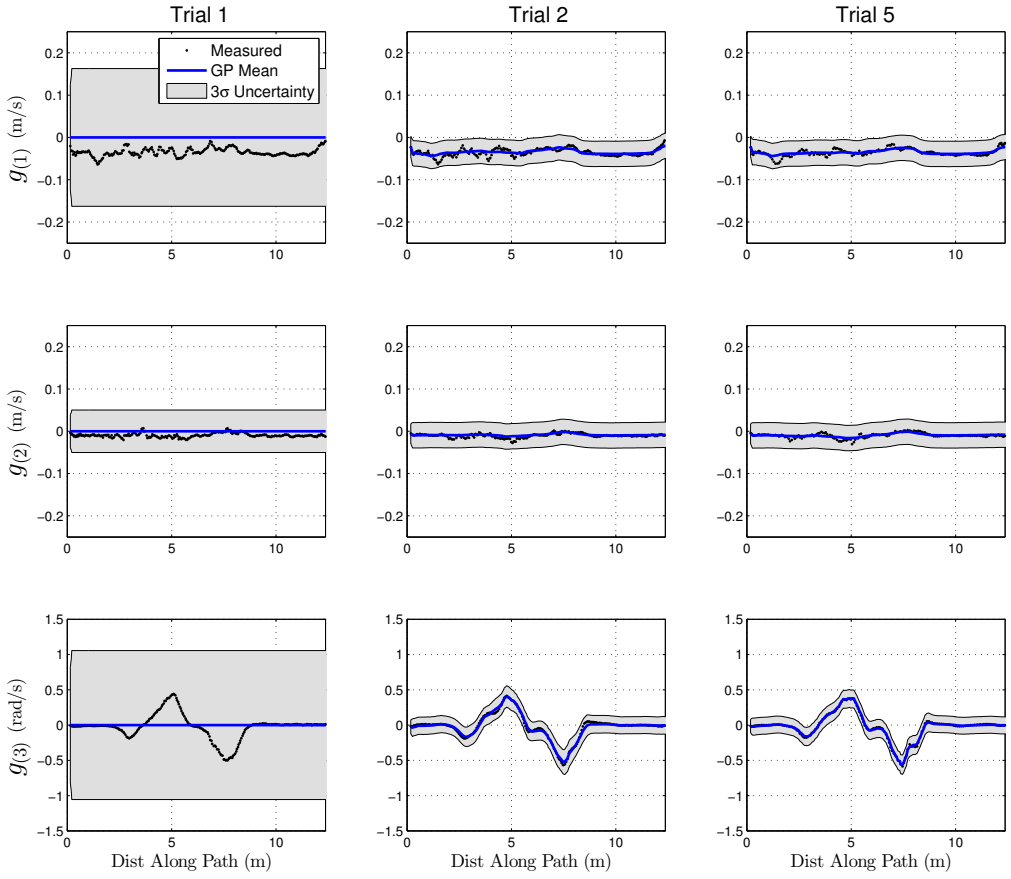


Figure 6.6: Modelled disturbances, $\mathbf{g}(\cdot) = (g_{(1)}(\cdot), g_{(2)}(\cdot), g_{(3)}(\cdot))$, during the first, second, and fifth trials. As the algorithm gathers experience, the model uncertainty decreases and model accuracy increases.

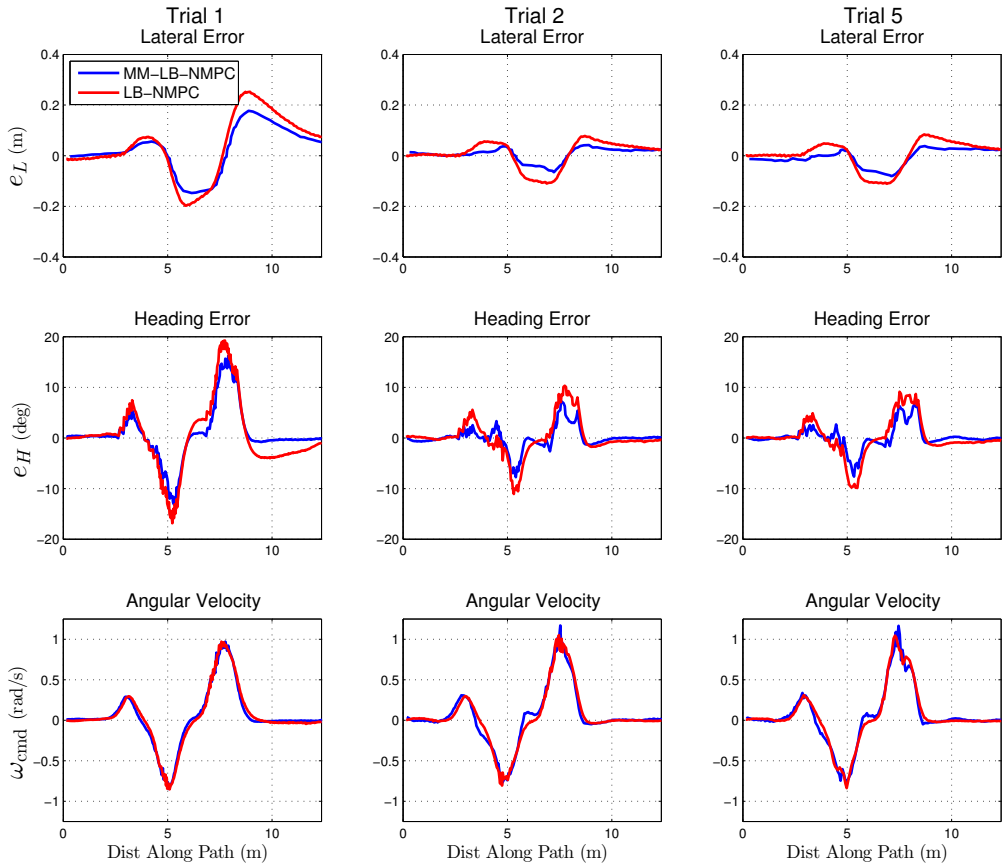


Figure 6.7: Tracking errors and commanded inputs vs. distance during the first, second, and fifth trials. Reducing errors when model uncertainty is high (i.e., trial 1) is important for controller stability and perspective-dependent, vision-based localization algorithms. As model uncertainty decreases, the MM-LB-NMPC algorithm naturally transitions towards an optimal control, balancing tracking errors and control inputs.

In general, the Min-Max algorithm incurred an increase in computation time of only 5%. This confirms our selection of a Sigma-Point Transform (Section 6.2.2) as an efficient method of predicting the 3σ confidence region without resorting to generating a large number of scenarios.

6.5 Discussion

The LB-NMPC algorithm presented in Chapter 5 offered a flexible method of improving path tracking through experience. It satisfied three of our desired qualities. Namely, it offered real-time control inputs over long-distance paths while enabling anytime learning and generalization from learned experiences. In this chapter, we extend our work

on LB-NMPC to address model uncertainty. Specifically, we showed how worst-case path-tracking errors can be reduced by optimizing for worst-case predicted scenarios. However, experimental work with the algorithm highlighted two opportunities for future work. Firstly, the robust learning-based controller continues to rely on *a priori* scheduled speeds. Ideally, the learning-based controller would be capable of optimally adjusting the robot speed as a reflection of model uncertainty, localization performance, and other real-time inputs. This would further improve the robustness of the overall system. Secondly, the reduction in worst-case path-tracking errors was indirectly chosen based on our choice of 3σ noise estimates and tuning of the MPC cost function. Ideally, an engineer would be provided with the ability to provide specific tracking error limits in relation to the local environment or localization algorithm. For example, mobile robots operating in open fields can generally tolerate relatively higher lateral tracking errors than robots operating in dense forests, where the robot must track the desired path as closely as possible in order to avoid collisions. As a result, Chapter 7 focuses on robust constraints in an efforts to guarantee controller stability and integrate with on-board guidance and navigation algorithms.

6.6 Conclusion

In summary, this chapter presents a novel, robust Min-Max Learning-Based Nonlinear Model Predictive Control (MM-LB-NMPC) algorithm. The MM-LB-NMPC algorithm offers an effective method of simultaneously exploiting and decreasing model uncertainty to improve controller performance and guarantee stability. We derive an efficient and robust extension to the LB-NMPC algorithm, altering the performance objective to optimize for the worst-case scenario. The algorithm uses a simple *a priori* vehicle model and a learned disturbance model. Disturbances are modelled as a Gaussian Process (GP) based on experience collected during previous traversals as a function of system state, input and other relevant variables. Furthermore, the novelty in this work also includes the efficient prediction of the mean and uncertainty of state sequences considering the learned model using a Sigma-Point Transform. Finally, worst-case scenarios are defined as sequences bounding the 3σ confidence region.

Experimental results are provided from tests with a 50 kg Clearpath Husky robot on a demonstrative path. The results show reductions in maximum lateral and heading path-tracking errors by up to 30% and a clear transition from robust control reduc-

ing worst-case errors, when the model uncertainty is high, to optimal control balancing tracking errors and control inputs, when model uncertainty is reduced. Furthermore, the algorithm requires only a 5% increase in computation time relative to the learning algorithm. Leveraging this work, our future work focuses on using the 3σ confidence region to provide safety guarantees and real-time speed scheduling.

Chapter 7

Robust Constrained Learning-based Model Predictive Control

7.1 Introduction

In this chapter, we present a Robust Constrained LB-NMPC (RC-LB-NMPC) algorithm for a path-repeating mobile robot[1]. Previously, we presented MM-LB-NMPC in order to reduce worst-case tracking errors compared to the unconstrained, non-robust LB-NMPC algorithm. The NMPC cost function was altered to optimize for the worst-case predicted sequence. However, the Min-Max approach does not provide a direct tuning knob to guarantee a specific level of tracking errors (the only tuning inputs are the MPC weights). In general, MPC is a powerful algorithm capable of computing optimal inputs while seamlessly addressing state and input constraints (Rawlings and Mayne, 2009). State constraints may represent physical obstacles or localization limits and as a result constraint satisfaction is tantamount to safety. However, system models used in practice are often uncertain and thus constraint satisfaction is difficult to guarantee in general.

Robust Constrained MPC (RC-MPC) is an active area of research and accounts for model uncertainty when considering state and input constraints (Mayne, 2014). The approach applies tightened constraints to nominal predictions at each time-step such that all plausible predicted sequences satisfy the given constraints considering a fixed estimate of model uncertainty. For example, Marruedo et al. (2002) present a robust constrained algorithm considering a nonlinear process model where tightened constraints

[1]Associated video at http://tiny.cc/RobotLearnsRobustly

Figure 7.1: State constraints for mobile robots frequently represent physical obstacles and as a result constraint satisfaction is required for safety. We present a Robust Constrained Learning-based Nonlinear MPC algorithm to guarantee constraint satisfaction while improving performance through learning. The algorithm is tested on a 900 kg Clearpath Grizzly travelling up to 2.0 m/s on off-road paths with tight constraints.

are computed offline, prior to operation. Pin et al. (2009) extend the work of Marruedo et al. (2002), computing tightened constraints online and enabling reductions in conservatism in the case that uncertainties are state-dependent. However, the uncertainty of predicted sequences given a sequence of control inputs can be quite conservative since future localization updates are not taken into account by the so-called open-loop trajectories. In parallel, Chisci et al. (2001) and Langson et al. (2004) propose Tube MPC for linear systems, a robust constrained algorithm solving for a sequence of control policies in an effort to account for future localization updates and reduce conservatism. Mayne et al. (2011) later extend Tube MPC to nonlinear systems. In practice, González et al. (2011) and Farrokhsiar et al. (2013) present applications of Tube MPC to mobile robots. However, Tube MPC is generally conservative since the models are not updated online. As a result, the fixed models often include a significant amount of uncertainty in order to account for unmodelled effects.

In this work, we investigate a Robust Constrained LB-NMPC (RC-LB-NMPC) algorithm for a path-repeating mobile robot operating in challenging outdoor terrain. The goal is to leverage robust constraints to guarantee stability throughout the learning process. The algorithm computes optimal angular and linear speeds at every time-step such that all plausible predicted sequences satisfy the given constraints considering a learned model. During initial trials when model uncertainty is high, the algorithm produces conservative, low-speed inputs. The algorithm then produces high-performance inputs during later trials when model uncertainty is reduced through learning. As a result, this work represents the only learning-based algorithm presented in this thesis to compute

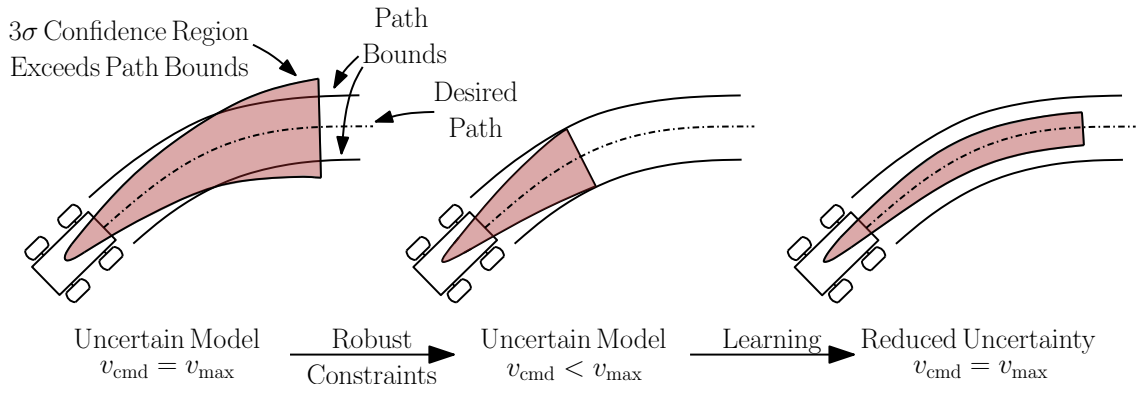


Figure 7.2: The predictive controller optimizes both the linear and angular velocities over a prediction horizon such that the 3σ confidence region is contained within the path bounds. (Left) At full speed with an uncertain model, the controller cannot guarantee constraint satisfaction throughout the prediction horizon. (Center) As a result, the controller selects slower, more conservative inputs. This is typical behavior of robust controllers that do not update model uncertainty over time. (Right) Over successive trials, learning then reduces model error and uncertainty and the controller is able to guarantee constraint satisfaction at the maximum allowed speed.

optimal linear speeds at every time-step, effectively representing real-time speed scheduling. Ideally, the incorporation of robust constraints in real-time also provides a direct mechanism for guidance or navigation algorithms to contribute state and input limits considering the current operating environment or nearby obstacles. However, we leave such integration work to future research.

Like our previous work on LB-NMPC and MM-LB-NMPC (Chapters 5 and 6), the algorithm is based on a fixed *a priori* model and a learned disturbance GP-based model enabling the prediction of the mean and uncertainty of disturbances. Furthermore, the algorithm also uses a Sigma-Point Transform to compute the mean and variance of predicted sequences given the two-component, learned model. However, as in existing research on robust constrained algorithms, we apply restricted constraints to the mean predicted sequence such that the 3σ confidence region is contained within the desired constraints (Figure 7.2). Aswani et al. (2013) also propose a RC-LB-MPC algorithm with a bounded, learned model. However, they use computationally expensive reachability analysis to compute restricted constraints. As a result, our algorithm enables future integration work handling constraints provided by guidance or localization algorithms in real-time. Finally, we present extensive experimental results including over 5 km of travel by a 900 kg skid-steered robot at speeds up to 2.0 m/s showing constraint satisfaction and performance improvements over time.

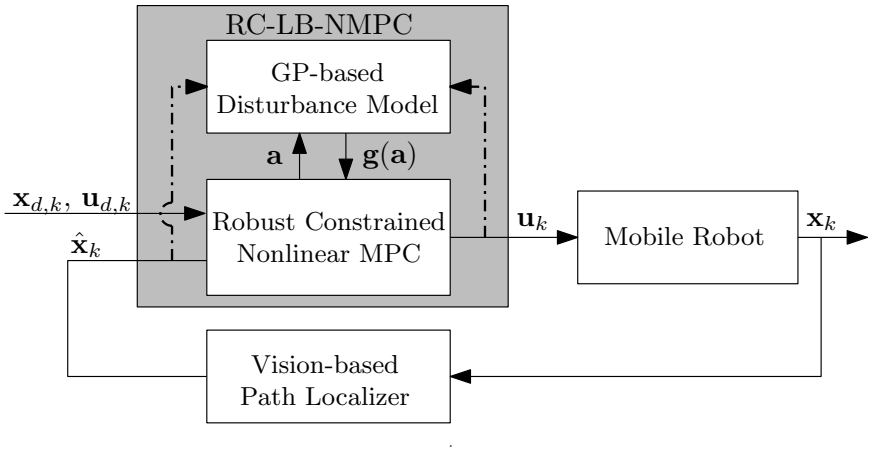


Figure 7.3: The RC-LB-NMPC algorithm is composed of two parts: (i) the robust constrained, path-tracking NMPC algorithm based on an *a priori* process model, and (ii) the GP-based disturbance model. During the first trial, the RC-NMPC algorithm relies solely on the *a priori* process model to follow the desired path, $\mathbf{x}_d$, considering the nominal inputs, $\mathbf{u}_d$. In later trials, the RC-NMPC algorithm uses the disturbance model as a correction to the *a priori* model. Dashed lines indicate that the signals update the model. In practice, our system combines the RC-LB-NMPC algorithm with localization from the vision-based VT&R system (Furgale and Barfoot, 2010) for off-road path tracking.

7.2 Mathematical Formulation

At a given sample time, NMPC finds a sequence of control inputs that optimizes the plant behavior over a prediction horizon based on the current state. The first input in the optimal sequence is then applied to the system. The entire process is repeated at the next sample time for the new system state. In Chapter 6, we presented MM-LB-NMPC, where we reduced worst-case errors by optimizing the NMPC cost function for worst-case sequences. In this work, we limit worst-case errors by applying robust constraints to the nominal predicted sequence (Figure 7.3).

7.2.1 Robust Constrained Nonlinear Model Predictive Control

Consider the following stochastic, learned process model,

$$\mathbf{x}_{k+1} = \overbrace{\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)}^{\textit{a priori}\text{ model}} + \overbrace{\mathbf{g}(\mathbf{a}_k)}^{\text{learned disturbance model}}, \tag{7.1}$$

with observable system state, $\mathbf{x}_k \sim \mathcal{N}(\bar{\mathbf{x}}_k, \mathbf{\Sigma}_k)$, $\mathbf{x}_k \in \mathbb{R}^n$, disturbance dependency, $\mathbf{a}_k \in \mathbb{R}^p$, and control input, $\mathbf{u}_k \in \mathbb{R}^m$, all at time k. The models $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ are

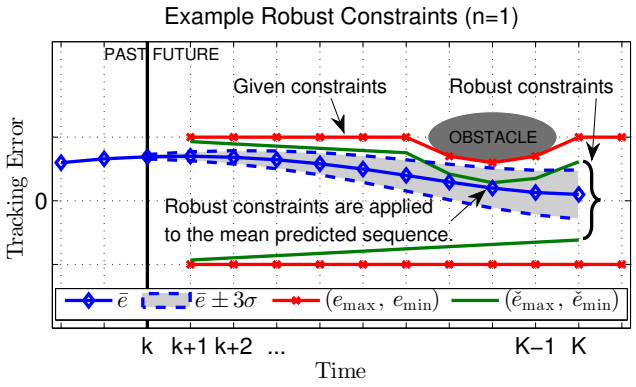


Figure 7.4: Here we show a 1D example of the computation of robust constraints, $\check{e}_{\max,k}$ and $\check{e}_{\min,k}$, that are applied to the mean predicted sequence, $\bar{e}$, based on the given constraints, $e_{\max,k}$ and $e_{\min,k}$, and the predicted sequence uncertainty, σ_k. As the predicted state becomes more uncertain, the robust constraints are increasingly restricted in order to guarantee that the 3σ confidence window, and thus the true state as well, are contained within the given constraints.

nonlinear process models: $\mathbf{f}(\cdot)$ is a known process model representing our knowledge of $\mathbf{f}_{\text{true}}(\cdot)$ and $\mathbf{g}(\cdot)$ is an (initially unknown) disturbance model representing discrepancies between the *a priori* model and the actual system behavior. Disturbances are modelled as a GP (Section 5.2.2), thus $\mathbf{g}(\cdot)$ is normally distributed, $\mathbf{g}(\cdot) \sim \mathcal{N}(\boldsymbol{\mu}(\cdot), \boldsymbol{\Sigma}_{\text{gp}}(\cdot))$. Previously, we showed that $\mathbf{g}(\cdot)$ could be used to learn higher-order dynamics by including historic states in the disturbance dependency (Chapter 5). However, for simplicity, we assume for now that $\mathbf{a}_k = (\bar{\mathbf{x}}_k, \mathbf{u}_k)$.

As previously mentioned, the goal of NMPC is to find a set of controls that optimizes the plant behavior over a given prediction horizon. We define the cost function to be minimized over the next K time-steps as,

$$J(\bar{\mathbf{x}}, \mathbf{u}) = (\mathbf{x}_d - \bar{\mathbf{x}})^T \mathbf{Q} (\mathbf{x}_d - \bar{\mathbf{x}}) + (\mathbf{u}_d - \mathbf{u})^T \mathbf{R} (\mathbf{u}_d - \mathbf{u}), \tag{7.2}$$

where $\mathbf{Q} \in \mathbb{R}^{Kn \times Kn}$ is positive semi-definite, $\mathbf{R} \in \mathbb{R}^{Km \times Km}$ is positive definite, $\mathbf{x}_d = (\mathbf{x}_{d,k+1}, \ldots, \mathbf{x}_{d,k+K})$ is a sequence of desired states, $\mathbf{x} = (\mathbf{x}_{k+1}, \ldots, \mathbf{x}_{k+K})$ is a sequence of uncertain predicted states, $\bar{\mathbf{x}}$ is the sequence of mean values based on $\mathbf{x}$, $\mathbf{u}_d = (\mathbf{u}_{d,k}, \ldots, \mathbf{u}_{d,k+K-1})$ is a sequence of desired inputs, and $\mathbf{u} = (\mathbf{u}_k, \ldots, \mathbf{u}_{k+K-1})$ is a sequence of inputs. Since we now optimize for both angular and linear speeds in order to achieve constraint satisfaction, we use desired inputs to specify the *a priori* scheduled speed for the robot. Linear speeds can be scheduled prior to driving according to knowl-

edge of place-specific path characteristics such as roughness or localization reliability (i.e., Experience-based Speed Scheduling as presented in Chapter 4).

We also define $2Kn$ probabilistic state constraints representing upper and lower tracking limits (Figure 7.4),

$$\left.\begin{array}{l} p(\mathbf{e}_{\max,k+i} - \mathbf{e}_{k+i} > \mathbf{0}) > l \\ p(\mathbf{e}_{k+i} - \mathbf{e}_{\min,k+i} > \mathbf{0}) > l \end{array}\right\} \quad i = 1 \dots K, \tag{7.3}$$

where $\mathbf{e}_{k+i} = \mathbf{x}_{d,k+i} - \mathbf{x}_{k+i}$ is the predicted tracking error, $\mathbf{e}_{\max,k+i}$ and $\mathbf{e}_{\min,k+i}$, are upper and lower tracking limits, respectively, $p(A)$ represents the probability of the event A, l is a desired confidence level, and the inequalities are evaluated component-wise. We then compute robust deterministic limits, $\check{\mathbf{e}}_{\max,k+i} = \mathbf{e}_{\max,k+i} - \alpha\boldsymbol{\sigma}_{k+i}$ and $\check{\mathbf{e}}_{\min,k+i} = \mathbf{e}_{\min,k+i} + \alpha\boldsymbol{\sigma}_{k+i}$, given the predicted uncertainty, $\boldsymbol{\sigma}_{k+i} = (\sqrt{\boldsymbol{\Sigma}_{k+i}(1,1)}, \dots, \sqrt{\boldsymbol{\Sigma}_{k+i}(n,n)})$, and apply them to the mean predicted states,

$$\check{\mathbf{e}}_{\max,k+i} > \bar{\mathbf{e}}_{k+i} > \check{\mathbf{e}}_{\min,k+i}, \quad i = 1 \dots K, \tag{7.4}$$

where $\bar{\mathbf{e}}_{k+i} = \mathbf{x}_{d,k+i} - \bar{\mathbf{x}}_{k+i}$, and the inequalities are evaluated component-wise. Typically, $\alpha = 3$ is chosen representing a confidence level of approximately 0.997. In the case that the predicted uncertainty is larger than the given tracking limits, the system is set to stop and request a user input. For example, if the vision-based localization becomes lost, the state uncertainty will likely exceed the given constraints.

In addition, we consider $2Km$ actuator constraints,

$$\mathbf{u}_{\max,k+i} > \mathbf{u}_{k+i} > \mathbf{u}_{\min,k+i}, \quad i = 0 \dots K-1, \tag{7.5}$$

where $\mathbf{u}_{\max,k+i}$ and $\mathbf{u}_{\min,k+i}$ are defined by the actuator limits, and the inequalities are evaluated component-wise. Finally, we define the complete set of inequality constraints, $c_i(\bar{\mathbf{x}}, \mathbf{u}) > 0, i = 1 \dots 2K(n+m)$, composed of both robust state (7.4) and input constraints (7.5).

Considering the current estimated system state, $\hat{\mathbf{x}}_k$, the process model, the constraints, and the cost function, we define the following constrained optimization problem,

$$\{\mathbf{x}_{\text{opt}}, \mathbf{u}_{\text{opt}}\} = \arg\min_{\mathbf{x},\mathbf{u}} J(\bar{\mathbf{x}}, \mathbf{u}) \tag{7.6a}$$

$$\text{subject to} \quad \bar{\mathbf{x}}_{k+i+1} - \mathbf{f}(\bar{\mathbf{x}}_{k+i}, \mathbf{u}_{k+i}) - \boldsymbol{\mu}(\mathbf{a}_{k+i}) = \mathbf{0}, \quad i = 0 \dots K-1, \tag{7.6b}$$

$$c_i(\bar{\mathbf{x}}, \mathbf{u}) > 0, \quad i = 1 \dots n_c, \tag{7.6c}$$

where the equality constraints are used to enforce the process model, and $n_c = 2K(n + m)$. We approximate the inequality constraints by introducing a slack variable, $\mathbf{w} = (w_1, \dots, w_{n_c})$, and a logarithmic barrier term (Boyd and Vandenberghe, 2004),

$$\{\mathbf{x}_{\text{opt}}, \mathbf{u}_{\text{opt}}, \mathbf{w}_{\text{opt}}\} = \arg\min_{\mathbf{x},\mathbf{u},\mathbf{w}} J(\bar{\mathbf{x}}, \mathbf{u}) - \eta \sum_{i=1}^{n_c} \log(w_i) \tag{7.7a}$$

$$\text{subject to} \quad \bar{\mathbf{x}}_{k+i+1} - \mathbf{f}(\bar{\mathbf{x}}_{k+i}, \mathbf{u}_{k+i}) - \boldsymbol{\mu}(\mathbf{a}_{k+i}) = \mathbf{0}, \quad i = 0 \dots K-1, \tag{7.7b}$$

$$c_i(\bar{\mathbf{x}}, \mathbf{u}) - w_i = 0, \quad i = 1 \dots n_c, \tag{7.7c}$$

where η is a small positive scalar adjusted towards zero throughout the optimization process. Finally, we use Lagrange methods (Boyd and Vandenberghe, 2004) to solve (7.7) and define the Lagrangian, $L(\mathbf{s})$,

$$\begin{aligned} L(\mathbf{s}) = J(\bar{\mathbf{x}}, \mathbf{u}) &- \sum_{i=0}^{K-1} \boldsymbol{\lambda}_i^T (\bar{\mathbf{x}}_{k+i+1} - \mathbf{f}(\bar{\mathbf{x}}_{k+i}, \mathbf{u}_{k+i}) - \boldsymbol{\mu}(\mathbf{a}_{k+i})) \\ &- \sum_{i=1}^{n_c} \gamma_i (c_i(\bar{\mathbf{x}}, \mathbf{u}) - w_i) - \eta \sum_{i=1}^{n_c} \log(w_i), \end{aligned} \tag{7.8}$$

where $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\mathbf{s} = (\bar{\mathbf{x}}, \mathbf{u}, \mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\gamma})$, $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_0, \dots, \boldsymbol{\lambda}_{K-1})$, and $\boldsymbol{\gamma} = (\gamma_1, \dots, \gamma_{n_c})$. Considering the necessary condition for optimality, $\nabla L(\mathbf{s}) = \mathbf{0}$, we employ Newton's method and iteratively linearize about an initial guess, $\mathbf{s} = \tilde{\mathbf{s}} + \delta\mathbf{s}$,

$$\nabla L(\mathbf{s}) \approx \nabla L(\tilde{\mathbf{s}}) + \left. \frac{\partial \nabla L(\mathbf{s})}{\partial \mathbf{s}} \right|_{\tilde{\mathbf{s}}} \delta\mathbf{s}, \tag{7.9}$$

solve for the value of $\delta\mathbf{s}$ such that,

$$\nabla L(\tilde{\mathbf{s}}) + \left. \frac{\partial \nabla L(\mathbf{s})}{\partial \mathbf{s}} \right|_{\tilde{\mathbf{s}}} \delta\mathbf{s} = \mathbf{0}, \tag{7.10}$$

Algorithm 2: RC-LB-NMPC

Data: $\hat{\mathbf{x}}_k$, $\mathbf{x}_d$, $\mathbf{u}_d$, and $\mathbf{s}_{\text{init}}$
Result: $\mathbf{x}_{\text{opt}}$,$\mathbf{u}_{\text{opt}}$,$\mathbf{w}_{\text{opt}}$
initialization: $\tilde{\mathbf{s}} = \mathbf{s}_{\text{init}}$;
while $\|\delta\mathbf{s}\| > \eta$ **do**
    Compute $\mathbf{x}$ using a Sigma-Point Transform (Section 6.2.2) given $\tilde{\mathbf{u}}$ and (7.1);
    Compute robust constraints;
    Linearize $\nabla L(\mathbf{s}) = \mathbf{0}$ around $\tilde{\mathbf{s}}$ and solve for $\delta\mathbf{s}$;
    Update $\tilde{\mathbf{s}}$, $\tilde{\mathbf{s}} \leftarrow \tilde{\mathbf{s}} + \beta\,\delta\mathbf{s}$;

and compute an updated value for $\tilde{\mathbf{s}}$,

$$\tilde{\mathbf{s}} \leftarrow \tilde{\mathbf{s}} + \beta\,\delta\mathbf{s}, \tag{7.11}$$

where β is selected at each iteration such that $w_i > 0, i = 1 \dots n_c$. A good initial guess for $\tilde{\mathbf{s}}$ can be drawn from the previous time-step. After iterating to convergence (Algorithm 2), we apply the first element of the resulting optimal control input sequence for one time-step, and start all over at the next time-step.

7.3 Implementation

7.3.1 Learning-based Controller

The work presented in this chapter is an extension of the insights and work presented in Chapters 5 and 6. As in LB-NMPC, disturbances are modelled as a GP enabling the prediction of the mean and uncertainty of model discrepancies (Section 5.2.2). Our work on both LB-NMPC and MM-LB-NMPC showed that modelling disturbances as a GP enables consistent estimates of the mean and uncertainty of disturbances. Moreover, when modelling disturbances as a GP, it is necessary to reduce computational complexity by carefully selecting support points. As a result, we continue to use the experience management scheme presented in Section 5.3.2. We also leverage the work presented in Chapter 6 and use a Sigma-Point Transform to efficiently predict state sequences (Section 6.2.2). However, unlike our work on MM-LB-NMPC, here we use the predicted uncertainty to compute robust constraints in order to provide guarantees on constraint satisfaction. Finally, we continue to model robots as unicycle vehicles (Section 5.3.1) with the same disturbance dependency, $\mathbf{a}_k = (\bar{\mathbf{x}}_k, \bar{\mathbf{v}}_{k-1}, \mathbf{u}_k, \mathbf{u}_{k-1})$, as used in Chapter 6. With these

fundamental building blocks, the RC-LB-NMPC algorithm represents a learning-based algorithm capable of real-time inputs over long-distance paths, anytime learning, and generalization while guaranteeing constraint satisfaction in spite of model uncertainty.

7.3.2 Localization-delay Compensation

This work also includes efforts to mitigate the effect of delayed localization. In practice, state estimates are provided by the path localizer in real-time (i.e., we require a minimum of 10 Hz). However, since the localization algorithm is the product of a sequence of steps, the state estimate based on a stereo-pair can be delivered up to 0.5 s after a stereo image-pair was acquired. Let time k represent the current time and suppose the RC-LB-NMPC receives the state estimate $\hat{\mathbf{x}}_{k-D}$ at time k (i.e., D represents the number of time-steps required to compute a state estimate based on an image-pair). Effectively, the algorithm collects historic inputs, $\{\mathbf{u}_{k-D}, \ldots, \mathbf{u}_{k-1}\}$, then uses the state estimate, $\hat{\mathbf{x}}_{k-D}$, and the learned model to estimate the state mean and uncertainty at time k,

$$\begin{aligned}
\hat{\mathbf{x}}_{k-D+1} &= \mathbf{f}(\hat{\mathbf{x}}_{k-D}, \mathbf{u}_{k-D}) + \mathbf{g}(\hat{\mathbf{a}}_{k-D}), \\
\hat{\mathbf{x}}_{k-D+2} &= \mathbf{f}(\hat{\mathbf{x}}_{k-D+1}, \mathbf{u}_{k-D+1}) + \mathbf{g}(\hat{\mathbf{a}}_{k-D+1}), \\
&\vdots \\
\hat{\mathbf{x}}_{k} &= \mathbf{f}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{g}(\hat{\mathbf{a}}_{k-1}).
\end{aligned} \tag{7.12}$$

Effectively, we account for changes in both the mean and uncertainty since the stereo image-pair was captured. In this way, we seamlessly integrate with localization systems and allow for variation in computation time. For example, under normal operation when state estimates are provided in a timely fashion, the localization-delay compensation accurately propagates the state for smooth operation. In the case that localization fails to provide any state estimates, the algorithm becomes increasingly uncertain of the current state of the robot until the RC-LB-NMPC algorithm is unable to guarantee constraint satisfaction and stops the robot. While it does not represent a significant breakthrough, the ability of the RC-LB-NMPC modelling and prediction framework to compensate for localization delay represents a simple extension and another benefit of our work on creating a flexible, learned model capable of generalization.

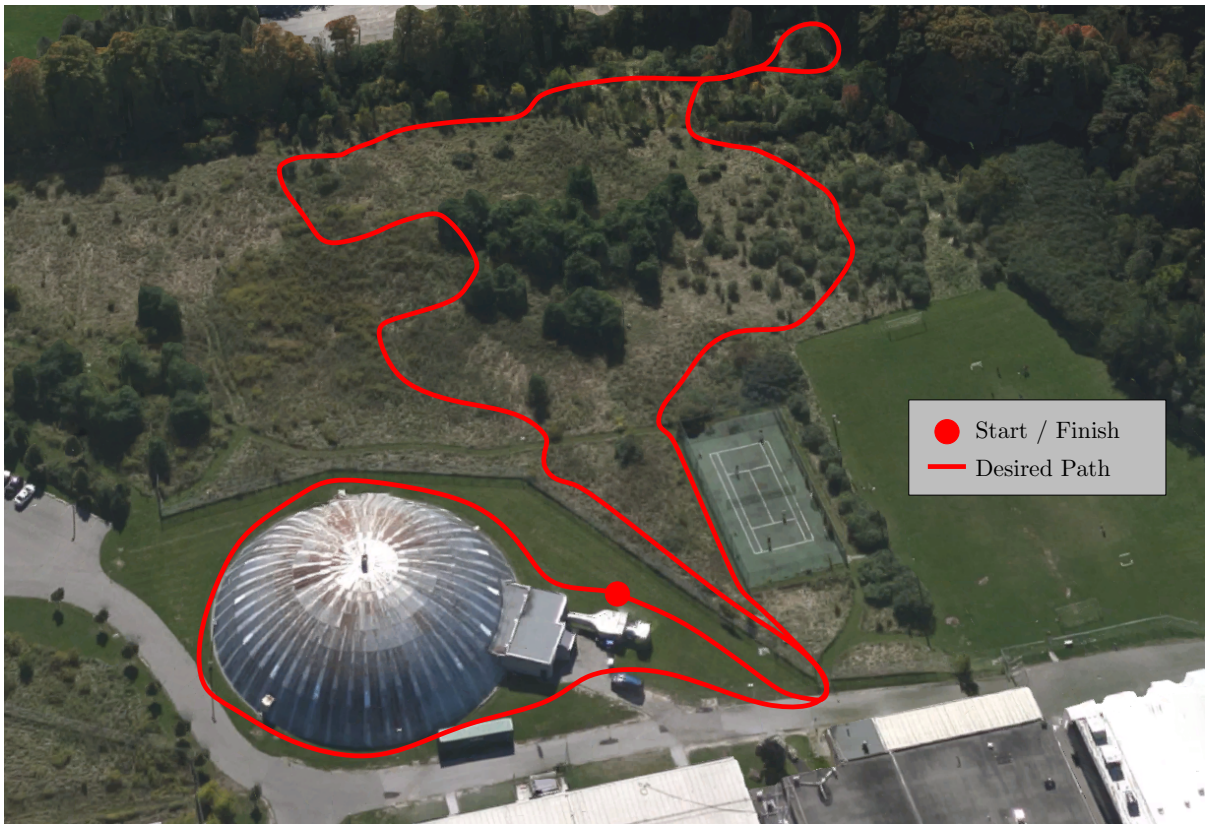


Figure 7.5: Testing culminated in the third experiment where the path, shown here, was defined around the University of Toronto Institute for Aerospace Studies (UTIAS) campus. The 900-m-long path passed in between trees and structures and over vegetation, pavement, inclines, and side-slopes. In all experiments, the nominal unicycle model used in our RC-LB-NMPC algorithm included no prior information on wheel-terrain interactions or robot dynamics. (Imagery: Google)

7.4 Experimental Results

7.4.1 Overview

We tested the RC-LB-NMPC algorithm on a 50 kg Clearpath Husky and a 900 kg Clearpath Grizzly in three different experiments with many different surface materials and topographies. This resulted in over 5 km of path tracking by the RC-LB-NMPC algorithm. The first and second experiments (Sections 7.4.3 and 7.4.4) compared unconstrained LB-NMPC, Constrained LB-NMPC (C-LB-NMPC), and the proposed RC-LB-NMPC algorithm. The three variants solved the same optimization problem (7.6) with the following two exceptions. First, state constraints (7.4) were disabled for the LB-NMPC algorithm. Second, the C-LB-NMPC algorithm included only non-robust state constraints

(i.e., $\alpha = 0$ when computing (7.4)). The third experiment (Section 7.4.5) thoroughly tested the RC-LB-NMPC algorithm over the course of five trials on an outdoor 900-m-long path involving pavement, dirt, sand, grass, inclines, and side slopes (Figure 7.5). The three experiments demonstrate the presented algorithm's ability to guarantee constraint satisfaction while improving performance through learning.

In all experiments, the controller described in Section 7.2 was implemented and run in addition to the VT&R software (Chapter 2) on a Lenovo W530 laptop with an Intel 2.6 Ghz Core i7 processor with 16 GB of RAM. The camera in all tests was a Point Grey Bumblebee XB3 stereo camera. The resulting real-time localization and control signals were generated at approximately 10 Hz. Since GPS was not available, the improvements due to the RC-LB-NMPC algorithm were quantified by the localization provided by the VT&R algorithm.

7.4.2 Tuning Parameters

The performance of the system was adjusted using the NMPC weighting matrices $\mathbf{Q}$ and $\mathbf{R}$, the confidence level, l, and the experience management parameters. The weighting matrices were selected in advance with a 25:1:10 ratios weighting path-tracking errors, angular control inputs, and linear control inputs for the 50 kg Husky and a 25:5:10 ratio for the 900 kg Grizzly robot. The increased weighting on the Grizzly inputs was selected to ensure controller stability at higher speeds. The selected confidence level was $l = 0.997$, resulting in $\alpha = 3$ (i.e., three standard deviations). As in previous chapters, local GP models were generated based on a sliding window of size, $c_{\text{vertex}} = 5$ and $c_{\text{velocity}} = 1$, where velocities were discretized by $v_{\text{bin}} = 0.25\,\text{m/s}$. The maximum number of experiences per bin, c_{bin}, was set to 4 resulting in local models based on up to 180 experiences.

7.4.3 Experiment 1: Indoor Algorithm Comparison

The first experiment compared three algorithms (LB-NMPC, C-LB-NMPC, and RC-LB-NMPC) over three trials using a 50 kg Clearpath Husky robot on an indoor, flat, concrete surface (Figure 7.6). Since the concrete did not develop tire ruts and the lighting did not change over the course of the experiment, these results provide a clear comparison of the algorithms.

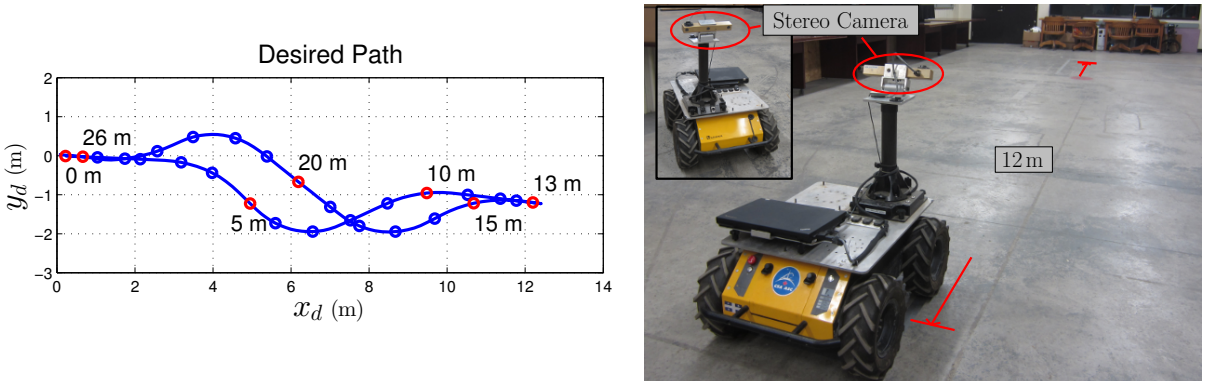


Figure 7.6: (Left) The desired path and (Right) the 50 kg Clearpath Husky at the start of the experiment 1 path. The smooth concrete floor and constant lighting providing identical path conditions for all trials.

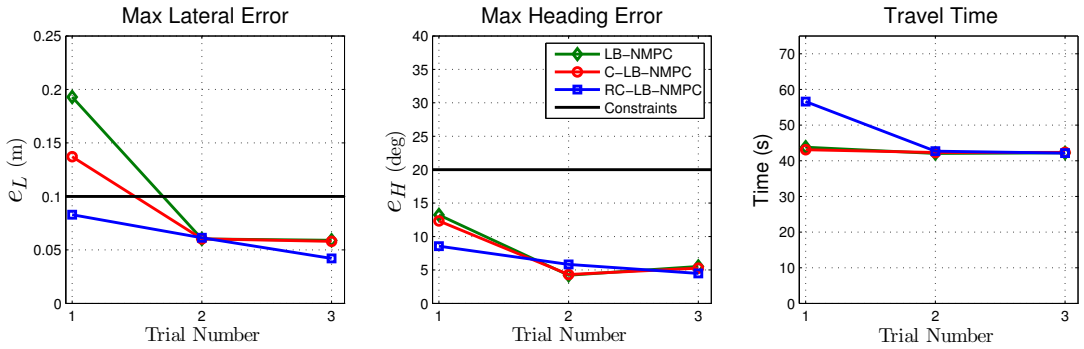


Figure 7.7: The maximum tracking errors and travel times vs. trial for the three compared algorithms in experiment 1. The RC-LB-NMPC algorithm results in constraint satisfaction at the cost an increased travel time (i.e., slower average speed).

We show the maximum tracking errors and travel times vs. trial in Figure 7.7. Without state constraints, the LB-NMPC algorithm incurs the largest tracking errors in the first trial, decreasing in later trials through learning. By adding constraints, the C-LB-NMPC algorithm reduced the errors during the first trial, but was overconfident and failed to satisfy the lateral constraints. On the other hand, the RC-LB-NMPC algorithm resulted in increased travel time relative to the other algorithms when the model was uncertain (i.e., during the first trial), but provided constraint satisfaction throughout all trials. Figure 7.8 shows tracking errors and control inputs vs. distance along the path for all trials. The LB-NMPC and C-LB-NMPC algorithms showed lateral constraint violations at 7, 16, and 24 m along the path (i.e., the turns) in the first trial. The RC-LB-NMPC algorithm avoided such constraint violations partly by lowering the speed to approximately 0.5 m/s throughout the path. This speed represents roughly half of the

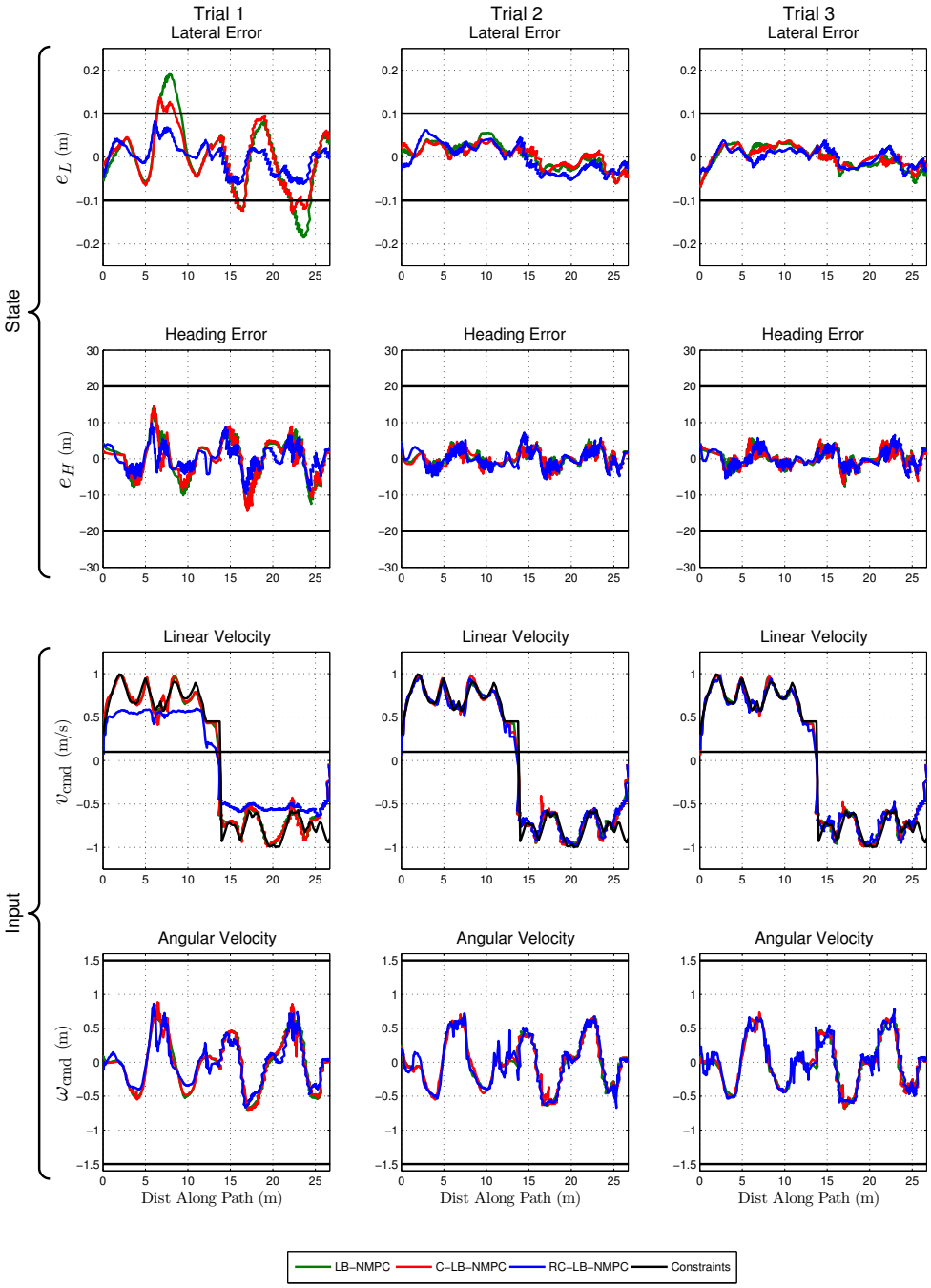


Figure 7.8: Tracking errors and commanded inputs vs. distance during the first, second, and third trials of experiment 1. The RC-LB-NMPC algorithm automatically reduces speed in order to meet lateral and heading constraints during the first trial. As model uncertainty is reduced through learning, the RC-LB-NMPC algorithm naturally increases speed and thus performance.

maximum speed possible by the Husky robot. The ability of the RC-LB-NMPC algorithm to optimally compute the linear and angular speeds of the robot in real-time while robustly meeting constraints represents one of the advantages of this algorithm. In practice, control algorithms must be capable of reacting robustly to given state constraints in tandem with scheduled speeds. This experiment also shows how the RC-LB-NMPC algorithm produces optimal results similar to the unconstrained algorithm after the learned model has collected experience. In this way, the RC-LB-NMPC algorithm behaves conservatively when the learned model is uncertain, and optimally when the model uncertainty has been reduced through learning.

7.4.4 Experiment 2: Off-road Algorithm Comparison

The second experiment once again compared the three algorithms (LB-NMPC, C-LB-NMPC, and RC-LB-NMPC). However, in this experiment, we tested with a 900 kg Clearpath Grizzly robot on a more challenging and realistic 90-m-long, off-road path (Figure 7.9) in the University of Toronto Institute for Aerospace Studies (UTIAS) Mars-Dome. In this experiment, the Grizzly was limited to 2.0 m/s considering path roughness. Since the path was on loose material and the Grizzly is quite heavy, significant ruts developed over the course of the experiment resulting in evolving disturbances from trial

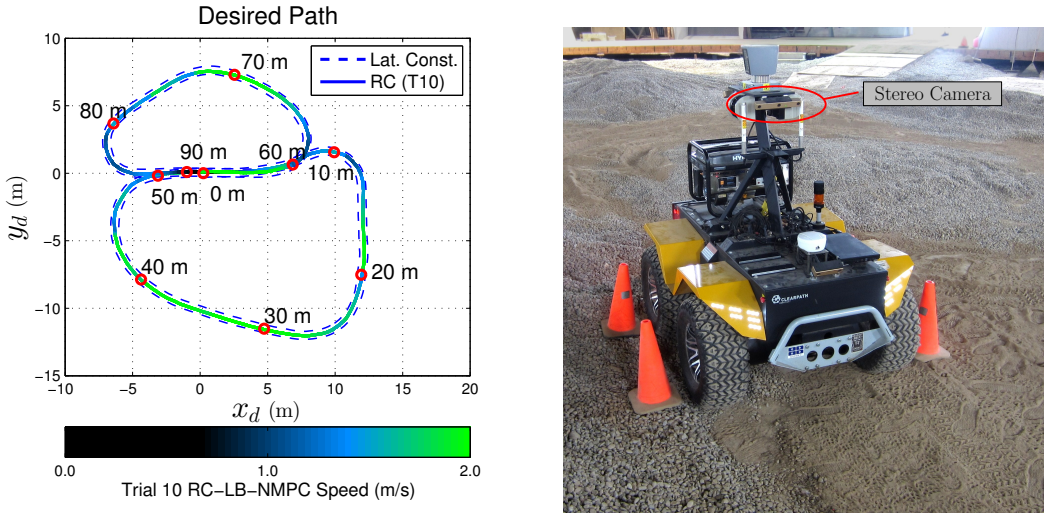


Figure 7.9: (Left) Lateral constraints outlining the experiment 2 path along with the RC-LB-NMPC actual path and speed (trial 10). The RC-LB-NMPC algorithm optimally selects angular and linear speeds in real-time considering the given constraints. (Right) The 900 kg Clearpath Grizzly at the start of the path. The off-road path, higher speeds, and heavy robot results in more challenging conditions for the algorithms.

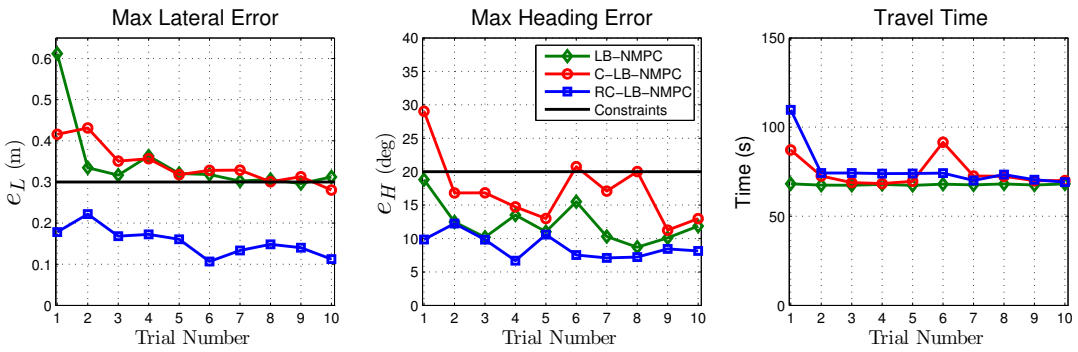


Figure 7.10: The maximum tracking errors and travel times vs. trial from experiment 2. Of the three algorithms, RC-LB-NMPC is the only one to respect constraints in all trials at the cost of higher travel time.

to trial. In this situation, we rely on the uncertainty of the learned model to capture the effect of the evolving disturbances, and only the RC-LB-NMPC algorithm uses the learned uncertainty to guarantee constraint satisfaction. We show the maximum tracking errors and travel times vs. trial in Figure 7.10. Without state constraints, the LB-NMPC algorithm results in the fastest travel time and also the highest lateral errors of the first trial. During later trials, the errors are reduced through learning with no notable changes in travel time. By adding constraints, the C-LB-NMPC algorithm reduced the errors during the first trial but still failed to satisfy the lateral constraints. As with experiment 1, the RC-LB-NMPC algorithm resulted in constraint satisfaction throughout all trials and decreased travel time through learning. This confirmed our goal of providing guaranteed constraint satisfaction while improving performance through learning. Figure 7.11 shows tracking errors and control inputs vs. distance along the path for trials one, two, and ten. Without constraints, the LB-NMPC algorithm resulted in the highest speeds and errors. The constraints led the C-LB-NMPC algorithm to reduce speeds in general but the algorithm was overconfident and exceeded limits during each trial. Specifically, the turn at 65 m is an example where significant ruts developed causing wheel slip. Since the RC-LB-NMPC algorithm robustly incorporated learned model uncertainty, the algorithm successfully passed through this section during each trial. Figure 7.12 shows measured and predicted disturbances used by the RC-LB-NMPC algorithm. The most complex disturbances affected the angular state of the robot, $g_{(3)}$.

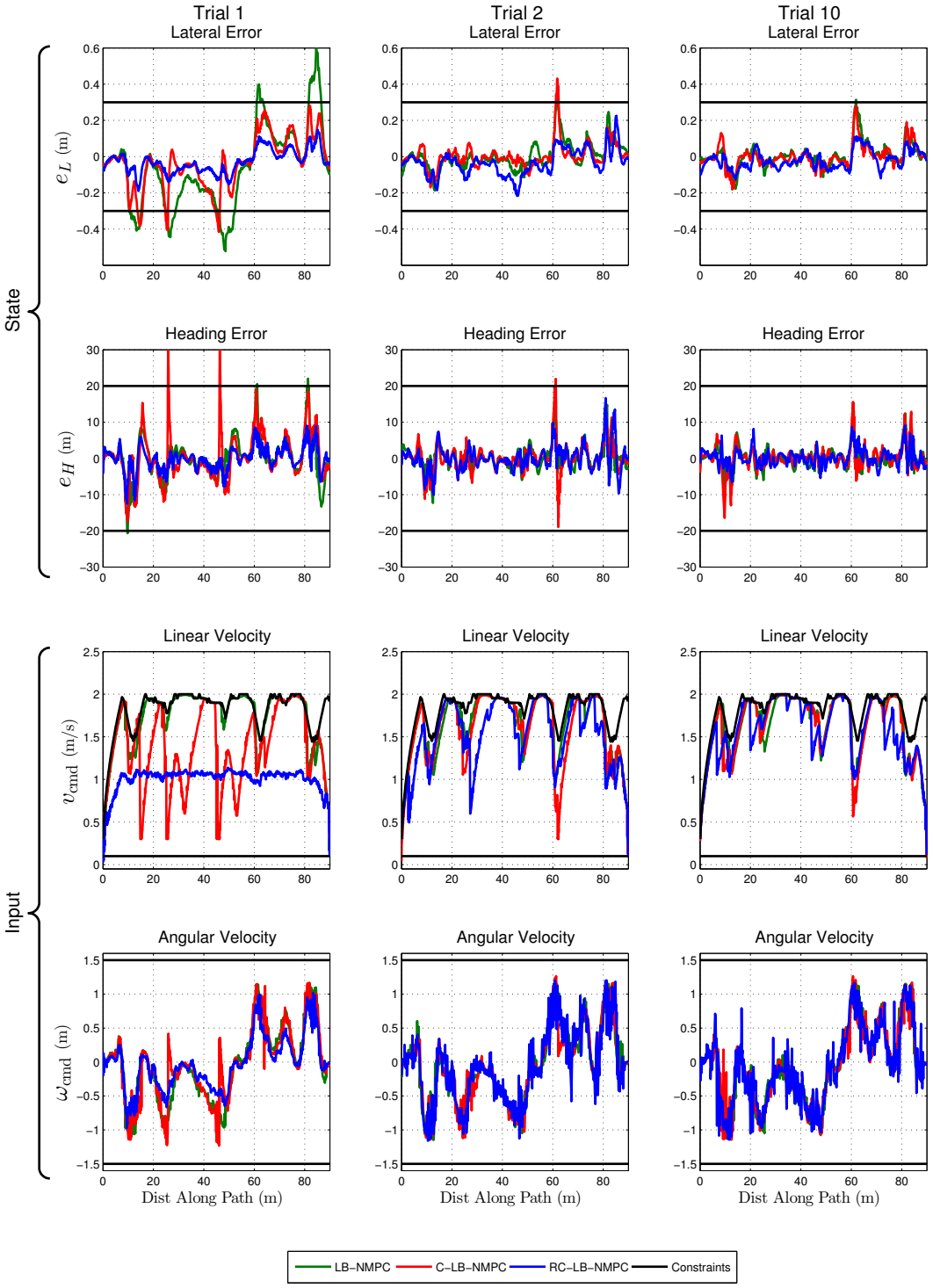


Figure 7.11: Tracking errors and commanded inputs vs. distance during the first, second, and tenth trials of experiment 2. The most challenging turn occurred at 60 m along the path, where significant ruts developed in the loose gravel.

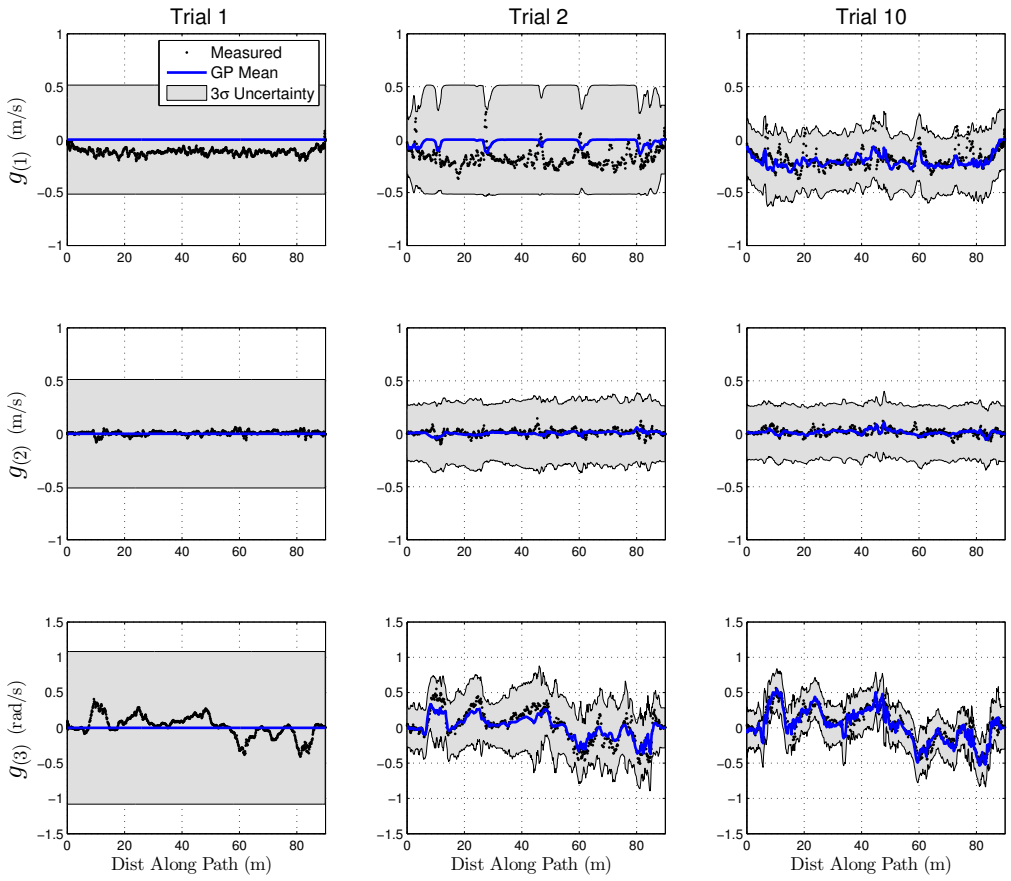


Figure 7.12: Modelled disturbances, $\mathbf{g}(\cdot) = (g_{(1)}(\cdot), g_{(2)}(\cdot), g_{(3)}(\cdot))$, during the first, second, and tenth trials. As the algorithm gathers experience, the model uncertainty decreases and model accuracy increases.

7.4.5 Experiment 3: Field Test

Finally, the third experiment thoroughly tested the RC-LB-NMPC algorithm on a 900-m-long, off-road path repeated five times (Figures 7.5 and 7.13). The third path covered a wide range of surfaces including grass, dirt, pavement, side slopes, and inclines while passing through trees, solid structures, and dense foliage. Once again, we tested with a 900 kg Clearpath Grizzly robot limited to 2.0 m/s considering the path roughness. We show that over the course of the experiment, the RC-LB-NMPC algorithm met constraints and improved performance over sequential trials (Figure 7.14). Moreover, the experiment shows that the algorithm as formulated was able to consistently deliver control input updates at 10 Hz despite the relatively large dataset gathered over the course of the five trials. Specifically, the learned model had accumulated over 25,000 experiences by the fifth trial, relying on the experience management scheme to extract up to 180

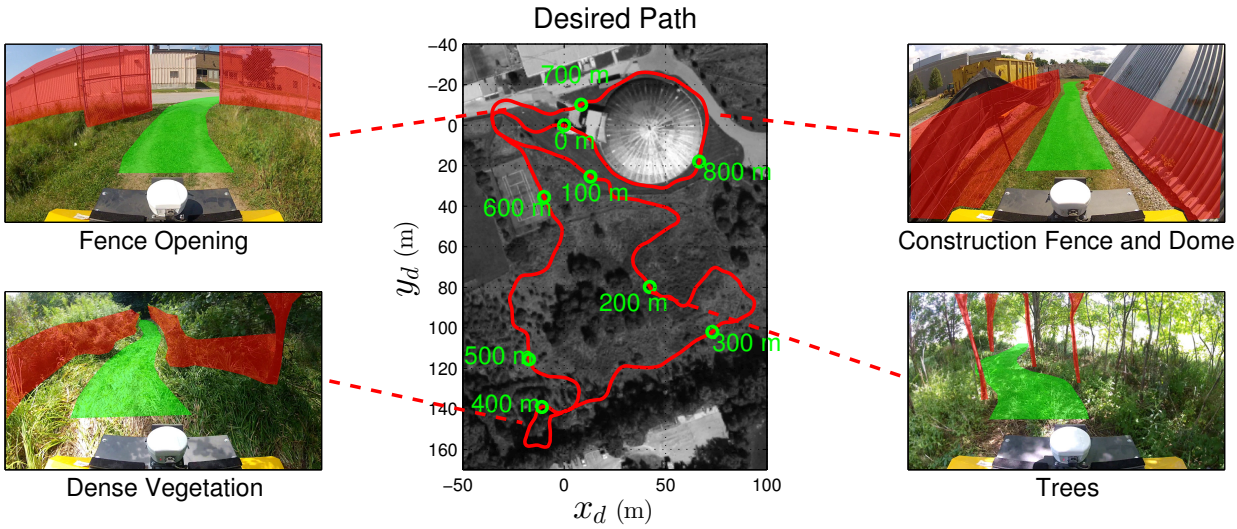


Figure 7.13: (Center) The experiment 3 desired path roughly overlaid on the UTIAS campus. (Left/Right) images showing path sections with constraint-satisfying terrain (green) and unsafe obstacles (red) highlighted. Without an obstacle detection algorithm, the desired path is manually taught with the required clearance and safe/unsafe areas shown are manually highlighted for illustration purposes. (Satellite Imagery: Google)

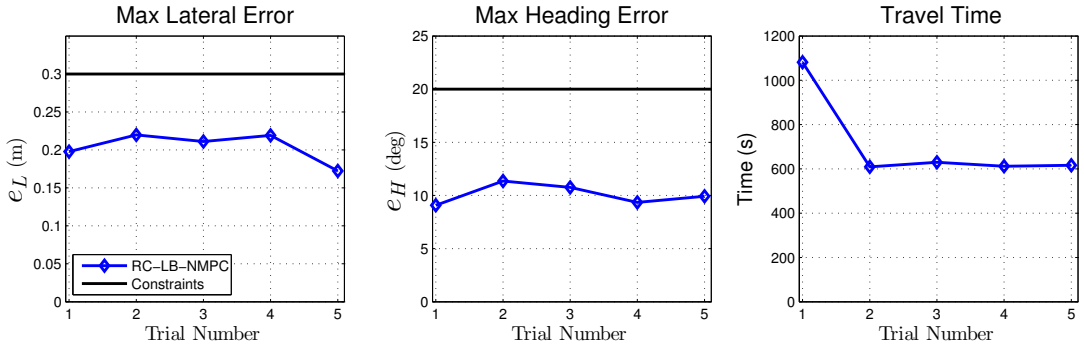


Figure 7.14: The maximum tracking errors and travel times vs. trial for experiment 3. Tracking errors met the given constraints while learning reduced the travel time by almost 50%.

relevant experiences at any given time-step. Tracking error and velocity distributions from trials one and five show that over the course of the experiment, the lateral error distribution widens but continues to fit within the limits (Figure 7.15). We do not expect the tracking errors to go to zero since the RC-LB-NMPC algorithm is an optimal control algorithm, balancing tracking errors and control inputs subject to the constraints. Finally, Figure 7.16 shows tracking errors and control inputs vs. distance along the path for trials one, two, and five. The first trial is representative of a conservative, non-learning RC-NMPC algorithm: without experiences, the mean predicted disturbance is zero but

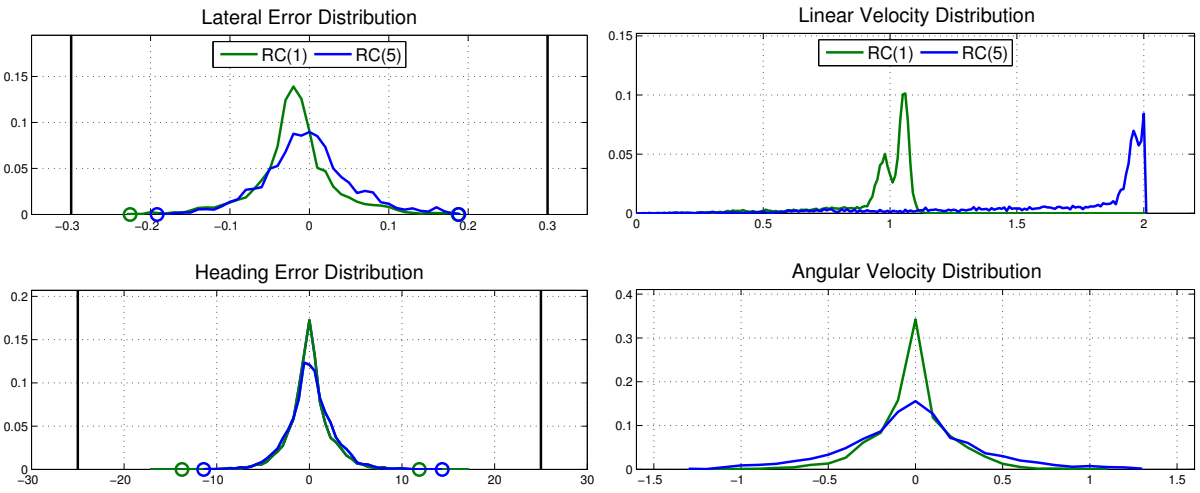


Figure 7.15: Error and velocity distributions for trials one and five of experiment 3. Learning resulted in a relatively wider distribution of lateral errors as the algorithm became more confident.

high uncertainty, capturing the wide range of possible disturbances. After just one trial, the learned model has collected enough experience to significantly reduce uncertainty and increase performance. Modelling disturbances as a GP enables efficient interpolation and extrapolation from data collected during previous trials.

7.5 Discussion

In general, this work combines concepts of RC-MPC and machine learning in a practical and flexible way. With respect to the robust constraints, there are two main opportunities. First, obstacle avoidance (i.e., the ability to optimally deviate slightly from a desired path) using an accurate, learned process model could be of benefit to mobile robots. As previously mentioned, one key benefit of our algorithm is that disturbances and robust constraints are computed in real-time. In practice, the controller could incorporate constraints provided by both the guidance and navigation algorithms to further improve overall system safety and reliability. Second, conservativeness of robust algorithms is an on-going topic for MPC research in general. As can be seen in the results of the RC-LB-NMPC algorithm presented in Chapter 7, the tracking errors remain relatively far from the actual constraints. Reductions in conservativeness will allow for increases in performance of practical robots. Also, one of our key requirements for real-time operation is the ability to rapidly identify relevant experiences for our local disturbance model.

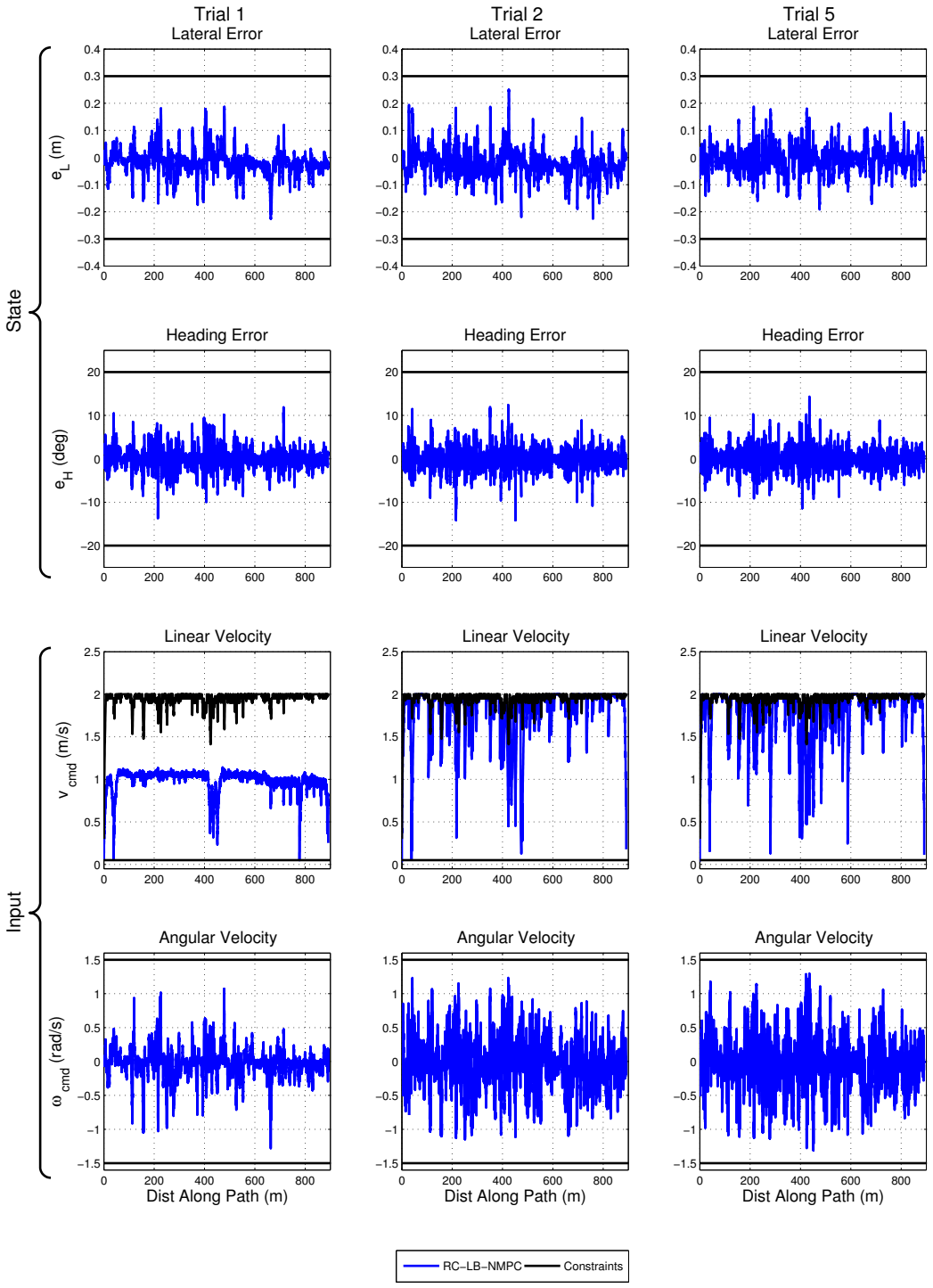


Figure 7.16: Tracking errors and commanded inputs vs. distance during the first, second, and tenth trials of experiment 3. Results in the first trial mimic that of a non-learning RC-NMPC algorithm with high model uncertainty representing all possible disturbances and thus conservative inputs.

7.6 Conclusion

In summary, this chapter presents a Robust Constrained Learning-based Nonlinear Model Predictive Control (RC-LB-NMPC) algorithm for a path-repeating, mobile robot operating in challenging off-road terrain. In Chapter 5, we demonstrated unconstrained LB-NMPC, where tracking errors were reduced using real-world experience instead of pre-programming accurate analytical models. In Chapter 6, we presented MM-LB-NMPC where the mean and uncertainty of state sequences were predicted using a Sigma-Point Transform. The work presented in this chapter represents a major contribution of this thesis, combining the real-time, GP-based disturbance model with efficient state sequence prediction, and state and input constraints. The resulting algorithm computes optimal linear and angular speeds at every time-step such that all plausible predicted sequences satisfy the given constraints considering a learned model whose uncertainty is decreasing over time.

The RC-LB-NMPC algorithm uses a fixed, simple robot model and a learned, non-parametric disturbance model. Disturbances represent measured discrepancies between the *a priori* model and observed system behavior. We use a Sigma-Point Transform to efficiently compute the mean and variance of predicted state sequences given the two-component, learned, stochastic model. Finally, we apply restricted constraints to the mean predicted sequence such that the 3σ confidence region is contained within the desired constraints. We provide extensive experimental results comparing unconstrained LB-NMPC, constrained LB-NMPC, and the RC-LB-NMPC algorithm using two significantly different robots and over 5 km of off-road travel. The results show that during initial trials when model uncertainty is high, the algorithm produces conservative, low-speed inputs. The algorithm then produces safe, high-performance inputs during later trials when model uncertainty is reduced through learning.

Chapter 8

Summary and Future Work

8.1 Summary of Contributions and Publications

In conclusion, we have investigated learning-based control for autonomous mobile robots in this thesis. Practical mobile robots operating in off-road terrain require techniques to mitigate the effects of surface materials, terrain topography, and complex robot dynamics. In practice, finding representative *a priori* models for off-road effects is challenging since (i) the terrain is often not known ahead of time, (ii) robot-terrain interaction models often do not exist, and (iii) even if such models did exist, finding model parameters is cumbersome. In this work, we propose to use experience to improve path tracking.

We began with the goal that the controller be capable of real-time operation over long-distance paths. As a result, we investigated Iterative Learning Control (ILC), a learning-based algorithm where feedforward control inputs are computed offline, prior to operation, based on tracking errors from previous trials. The contributions associated with Chapter 3 include:

1. A learning-based algorithm using vision-based localization enabling high-performance operation in off-road environments without the need for external infrastructure,

2. A learning-based algorithm for mobile robots comprised of ILC in parallel with a feedback controller enabling path completion during the first trial and improved performance thereafter,

3. A novel formulation where feedforward commands are indexed by distance enabling the robot to track the desired path at a safe speed during the first trial, with increased speed during later trials,

4. The first experimental results for ILC on challenging, off-road paths including over 700 m of travel by two significantly different skid steered robots.

The publication associated with this chapter is:

- Ostafew, C., Schoellig, A. P., and Barfoot, T. D. (2013). Visual Teach and Repeat, Repeat, Repeat: Iterative Learning Control to Improve Mobile Robot Path Tracking in Challenging Outdoor Environments. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 176-181.

Based on experimental testing of our work on ILC, we determined that anytime learning and the ability to generalize from learned experiences is necessary in enabling flexible, learning-based behavior for mobile robots. Moreover, we also found an opportunity to use experience to compute time-optimal speed schedules for any given trial. As a result, in Chapters 4 and 5, we presented an automated speed scheduler and Learning-based Nonlinear Model Predictive Control (LB-NMPC). The contributions associated with these chapters include:

1. A novel speed scheduler incorporating path-tracking experience, localization experience, and *a priori* speed and acceleration limits,

2. The first LB-NMPC algorithm for mobile robots where an *a priori* process model is augmented with an experience-based disturbance model. The resulting controller compensates for effects not captured by the fixed process model, such as environmental disturbances and unknown dynamics,

3. A novel approach where disturbances are modelled as a Gaussian Process (GP), and predictions are computed based on a sliding window of training data. This allows for real-time operation over long paths, anytime learning, and the ability to generalize from experience,

4. Extensive experimental results on three significantly different robots with over 7 km of travel on off-road paths.

The publications associated with these chapters are:

- Ostafew, C., Collier, J., Schoellig, A.P., and Barfoot, T. (2014a). Speed Daemon: Experience-Based Mobile Robot Speed Scheduler. In *Proceedings of the Conference on Computer and Robot Vision*, Best Robotics Paper Award, pages 56-62.

- Ostafew, C., Schoellig, A.P., and Barfoot, T. (2014b). Learning-Based Nonlinear Model Predictive Control to Improve Vision-Based Mobile Robot Path-Tracking in Challenging Outdoor Environments. In *Proceedings of the International Conference on Robotics and Automation*, pages 4029-4036.

- Ostafew, C., Collier, J., Schoellig, A. P., and Barfoot, T. D. (2015a). Learning-based Nonlinear Model Predictive Control to Improve Vision-based Mobile Robot Path Tracking. *Journal of Field Robotics*, 33 (1): 133-152.

Finally, based on our work with LB-NMPC, we determined that in addition to real-time control over long-distance paths, anytime learning, and the ability to generalize from experience, the learning-based algorithm should also provide robust control inputs. Specifically, the learning algorithm begins with an uncertain, nominal process model and learns an accurate, low-uncertainty model based on experience. As a result, we presented Min-Max LB-NMPC and Robust Constrained LB-NMPC in Chapters 6 and 7 in order to guarantee stability throughout the learning process in spite of model uncertainty. The contributions associated with these chapters include:

1. The first example of efficient prediction of the mean and uncertainty of state sequences considering a learned model using a Sigma-Point Transform,

2. A novel Min-Max LB-NMPC (MM-LB-NMPC) algorithm combining the GP-based learned model, efficient prediction of state sequences using a Sigma-Point Transform, and an alteration to the NMPC performance function to optimize for the worst-case scenario,

3. A novel Robust Constrained LB-NMPC (RC-LB-NMPC) algorithm combining the GP-based learned model, efficient prediction of state sequences using a Sigma-Point Transform, and state and input constraints.

The publications associated with these chapters are:

- Ostafew, C., Schoellig, A., and Barfoot, T. (2015b). Conservative to Confident: Treating Uncertainty Robustly Within Learning-based Control. In *Proceedings of the International Conference on Robotics and Automation*, pages 421-427.

- Ostafew, C., Schoellig, A. P., and Barfoot, T. D. (2016). Robust Constrained Learning-based NMPC Enabling Reliable Mobile Robot Path Tracking. *International Journal of Robotics Research*, To Appear.

Finally, efforts towards experimental validation of Guidance, Navigation, and Control frameworks in general also led to a number of publications. These contributions include the application of the framework for a human-guided exploration task, and lighting-resistant VT&R. These were published as follows:

- Berczi, L.-P., Ostafew, C., Stenning, B., Barfoot, T., Jones, E., Tornabene, L., Osinski, G., and Daly, M. (2014). Place Revisiting and Teleoperation for a Sample-Return Mission Control Architecture. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, Montreal, Canada.

- Paton, M., MacTavish, K., Ostafew, C., and Barfoot, T. D. (2015). It's Not Easy Seeing Green: Lighting-resistant Stereo Visual Teach & Repeat using Color-Constant Images. In *Proceedings of the International Conference on Robotics and Automation*, pages 1519-1526.

8.2 Future Work

In this thesis, we focused on developing the primary building blocks enabling real-time, flexible, learning-based control for mobile robots. The work culminated in the RC-LB-NMPC algorithm capable of learning from experience while guaranteeing stability throughout the learning process. As discussed in Chapter 5, two opportunities for future work include the analysis of convergence rates and the exploration vs. exploitation problem as discussed in Chapter 5. Moreover, we presented the opportunities for obstacle avoidance and reductions in the conservativeness of the RC-LB-NMPC algorithm in Chapter 7. In addition to the insights provided throughout the thesis, there are other opportunities for future work stemming from both improvements to the individual components and to the overall algorithm.

In order to enable real-time control of the GP-based algorithms, we employed an experience-management scheme. The approach created local models at every time-step based on relevant experiences and discarded older experiences in order to keep the total number of experiences manageable. Improvement of the learned model represents a significant opportunity for future work. For example, further research into the selection of support points is likely to improve modelling accuracy and reduce computation time by requiring fewer experiences in each local model. Moreover, the approach of discarding

experiences precludes the possibility of long-term evolution of robot-terrain interactions due to changes in weather or season. Finally, one could also investigate the ability to manage mislearning. Mislearning occurs when the model incorporates an outlier disturbance estimate and could perhaps be identified under an experience management scheme.

With respect to the overall concept of learning-based control, Jordan and Mitchell (2015) present a survey of recent trends and prospects for machine learning. Specifically, they highlight the opportunity and need for further research into team-based learning. The goal is to identify effective methods of transferring data between robots such that any improvements found by one robot can be shared by all robots in a team. Team-based learning is a growing topic studying various methods of transferring data between robots such that any improvements found by one robot can be shared by all. As opposed to our experiments demonstrating a single robot tracking a single path, one could imagine a large network of paths with many robots improving performance collectively and safely. Finally, there is an opportunity for further work on generalization from experiences. In this work, we assume disturbances are place-specific and generalize locally (i.e., from experiences collected at nearby locations and similar speeds). However, future work could be focused on removing the place-specific assumption and generalizing from path to path based on speed or path qualities, like curvature.

These possibilities for future work will further enable robust, safe, life-long learning for practical mobile robots.

Bibliography

Ahn, H.-S., Chen, Y., and Moore, K. L. (2007). Iterative Learning Control: Brief Survey and Categorization. *IEEE Transactions on Systems, Man, and Cybernetics: Applications and Reviews*, 37(6):1099–1121. (ref. pages 18 and 69)

Aicardi, M., Casalino, G., Bicchi, A., and Balestrino, A. (1995). Closed Loop Steering of Unicycle like Vehicles via Lyapunov Techniques. *IEEE Robotics & Automation Magazine*, 2(1):27–35. (ref. page 3)

Arimoto, S., Kawamura, S., and Miyazaki, F. (1984). Bettering Operation of Robots by Learning. *Journal of Robotic Systems*, 1(2):123–140. (ref. pages 5, 7, 18, and 19)

Aswani, A., Gonzalez, H., Sastry, S. S., and Tomlin, C. (2013). Provably Safe and Robust Learning-based Model Predictive Control. *Automatica*, 49:1216–1226. (ref. pages 45 and 88)

Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Speeded-up Robust Features (SURF). *Computer vision and image understanding*, 110(3):346–359. (ref. page 12)

Bemporad, A., Borrelli, F., and Morari, M. (2003). Min-Max Control of Constrained Uncertain Discrete-time Linear Systems. *IEEE Transactions on Automatic Control*, 48(9):1600–1606. (ref. page 72)

Berczi, L.-P., Ostafew, C., Stenning, B., Barfoot, T., Jones, E., Tornabene, L., Osinski, G., and Daly, M. (2014). Place Revisiting and Teleoperation for a Sample-Return Mission Control Architecture. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space*.

Bianco, C. (2009). Kinematically Constrained Smooth Real-time Velocity Planning for Robotics Applications. In *Proceedings of the IEEE International Conference on Control and Automation*, pages 373–378. (ref. page 32)

Blackmore, L. (2006). A Probabilistic Particle Control Approach to Optimal, Robust Predictive Control. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*. (ref. page 72)

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge university press. (ref. page 92)

Bristow, D. A., Tharayil, M., and Alleyne, A. G. (2006). A Survey of Iterative Learning Control. *IEEE Control Systems*, 26(3):96–114. (ref. pages 18, 23, and 69)

Brockett, R. W. (1983). *Asymptotic Stability and Feedback Stabilization*. Birkhauser, Boston, MA. (ref. page 3)

Brooks, C. A. and Iagnemma, K. (2005). Vibration-based Terrain Classification for Planetary Exploration Rovers. *IEEE Transactions on Robotics*, 21(6):1185–1191. (ref. page 33)

Calafiore, G. C. and Fagiano, L. (2013). Robust Model Predictive Control via Scenario Optimization. *IEEE Transactions on Automatic Control*, 58(1):219–224. (ref. page 73)

Campo, P. J. and Morari, M. (1987). Robust Model Predictive Control. In *Proceedings of the American Control Conference*, pages 1021–1026. (ref. page 72)

Cariou, C., Lenain, R., Thuilot, B., and Berducat, M. (2009). Automatic Guidance of a Four-Wheel-Steering Mobile Robot for Accurate Field Operations. *Journal of Field Robotics*, 26(6-7):504–518. (ref. page 4)

Chen, Y. and Moore, K. (2002a). A Practical Iterative Learning Path-Following Control of an Omni-Directional Vehicle. *Asian Journal of Control*, 4(1):90–98. (ref. page 19)

Chen, Y. and Moore, K. L. (2002b). An Optimal Design of PD-type Iterative Learning Control with Monotonic Convergence. In *Proceedings of the IEEE International Symposium on Intelligent Control*, pages 55–60. (ref. page 19)

Chisci, L., Rossiter, J. A., and Zappa, G. (2001). Systems with Persistent Disturbances: Predictive Control with Restricted Constraints. *Automatica*, 37(7):1019–1028. (ref. page 87)

De Luca, A. and Di Benedetto, M. D. (1993). Control of Nonholonomic Systems via Dynamic Compensation. *Kybernetika*, 29(6):593–608. (ref. page 3)

De Roover, D. and Bosgra, O. H. (2000). Synthesis of Robust Multivariable Iterative Learning Controllers with Application to a Wafer Stage Motion System. *International Journal of Control*, 73(10):968–979. (ref. page 19)

Diehl, M., Ferreau, H. J., and Haverbeke, N. (2009). Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation. In *Lecture notes in control and information sciences, volume 384, Nonlinear Model Predictive Control*, pages 391–417. Springer. (ref. page 48)

Dong, W. and Kuhnert, K.-D. (2005). Robust Adaptive Control of Nonholonomic Mobile Robot with Parameter and Nonparameter Uncertainties. *IEEE Transactions on Robotics*, 21(2):261–266. (ref. page 4)

Farrokhsiar, M., Pavlik, G., and Najjaran, H. (2013). An Integrated Robust Probing Motion Planning and Control Scheme: A Tube-based MPC Approach. *Robotics and Autonomous Systems*, 61(12):1379–1391. (ref. page 87)

Fatouhi, R., Szyszkowski, W., and Nikiforuk, P. (2002). Trajectory Planning and Speed Control for a Two-link Rigid Manipulator. *Journal of Mechanical Design*, 124(3):585–589. (ref. page 32)

Fierro, R. and Lewis, F. L. (1998). Control of a Nonholonomic Mobile Robot using Neural Networks. *IEEE Transactions on Neural Networks*, 9(4):589–600. (ref. page 4)

Foglia, M. M. and Reina, G. (2006). Agricultural Robot for Radicchio Harvesting. *Journal of Field Robotics*, 23(6-7):363–377. (ref. page 1)

Freeman, C., Cai, Z., Rogers, E., and Lewin, P. (2011). Iterative Learning Control for Multiple Point-to-Point Tracking Application. *IEEE Transactions on Control Systems Technology*, 19(3):590–600. (ref. page 25)

Fukao, T., Nakagawa, H., and Adachi, N. (2000). Adaptive Tracking Control of a Nonholonomic Mobile Robot. *IEEE Transactions on Robotics and Automation*, 16(5):609–615. (ref. page 4)

Furgale, P. (2011). *Extensions to the Visual Odometry Pipeline for the Exploration of Planetary Surfaces.* PhD thesis, University of Toronto Institute for Aerospace Studies. (ref. pages 11, 13, and 15)

Furgale, P. and Barfoot, T. (2010). Visual Teach and Repeat for Long-Range Rover Autonomy. *Journal of Field Robotics*, 27(5):534–560. (ref. pages 11, 22, 67, and 89)

Ghosh, J. and Paden, B. (2002). A Pseudoinverse-based Iterative Learning Control. *IEEE Transactions on Automatic Control*, 47(5):831–837. (ref. page 19)

González, R., Fiacchini, M., Guzmán, J. L., Álamo, T., and Rodríguez, F. (2011). Robust Tube-based Predictive Control for Mobile Robots in Off-road Conditions. *Robotics and Autonomous Systems*, 59(10):711–726. (ref. page 87)

Guillet, A., Lenain, R., and Thuilot, B. (2013). Off-Road Path Tracking of a Fleet of WMR with Adaptive and Predictive Control. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 2855–2861. (ref. page 4)

Gunnarsson, S. and Norrlöf, M. (2001). On the Design of ILC Algorithms using Optimization. *Automatica*, 37(12):2011–2016. (ref. page 19)

Han, K.-L. and Lee, J. (2011). Iterative Path Tracking of an Omni-Directional Mobile Robot. *Advanced Robotics*, 25:1817–1838. (ref. page 19)

Hehn, M. and D'Andrea, R. (2014). A Frequency Domain Iterative Learning Algorithm for High-performance, Periodic Quadrocopter Maneuvers. *Mechatronics*, 24(8):954 – 965. (ref. page 18)

Hellström, T., Johansson, T., and Ringdahl, O. (2006). Development of an Autonomous Forest Machine for Path Tracking. In *Proceedings of the 5th Annual Conference on Field and Service Robotics*, pages 603–614. Springer. (ref. pages 1 and 2)

Howard, T., Green, C., and Kelly, A. (2009). Receding Horizon Model-Predictive Control for Mobile Robot Navigation of Intricate Paths. In *Proceedings of the 7th Annual Conference on Field and Service Robotics*, pages 69–78. (ref. page 45)

Johnson, A. E., Goldberg, S. B., Cheng, Y., and Matthies, L. H. (2008). Robust and Efficient Stereo Feature Tracking for Visual Odometry. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 39–46. (ref. pages 1 and 11)

Jordan, M. and Mitchell, T. (2015). Machine Learning: Trends, Perspectives, and Prospects. *Science*, 349(6245):255–260. (ref. page 111)

Julier, S. J. and Uhlmann, J. K. (2004). Unscented Filtering and Nonlinear Estimation. *Proceedings of the IEEE*, 92(3):401–422. (ref. pages 73 and 78)

Kanayama, Y., Kimura, Y., Miyazaki, F., and Noguchi, T. (1990). A Stable Tracking Control Method for an Autonomous Mobile Robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 384–389. (ref. page 3)

Kang, M., Lee, J., and Han, K. (2005). Kinematic Path-Tracking of Mobile Robot using Iterative Learning Control. *Journal of Robotic Systems*, 22(2):111–121. (ref. page 19)

Kant, K. and Zucker, S. W. (1986). Toward Efficient Trajectory Planning: The Path-Velocity Decomposition. *The International Journal of Robotics Research*, 5(3):72–89. (ref. page 33)

Kelly, A. and Stentz, A. (1998a). Rough Terrain Autonomous Mobility (Part 1): A Theoretical Analysis of Requirements. *Autonomous Robots*, 5(2):129–161. (ref. page 33)

Kelly, A. and Stentz, A. (1998b). Rough Terrain Autonomous Mobility (Part 2): An Active Vision, Predictive Control Approach. *Autonomous Robots*, 5(2):163–198. (ref. page 33)

Kerrigan, E. C. and Maciejowski, J. M. (2004). Feedback Min-Max Model Predictive Control using a Single Linear Program: Robust Stability and the Explicit Solution. *International Journal of Robust and Nonlinear Control*, 14(4):395–413. (ref. page 72)

Klančar, G. and Škrjanc, I. (2007). Tracking-Error Model-Based Predictive Control for Mobile Robots in Real Time. *Robotics and Autonomous Systems*, 55(6):460–469. (ref. pages 5 and 44)

Kocijan, J., Murray-Smith, R., Rasmussen, C., and Girard, A. (2004). Gaussian Process Model Based Predictive Control. In *Proceedings of the American Control Conference*, volume 3, pages 2214–2219. (ref. page 45)

Kolmanovsky, I. and McClamroch, N. H. (1995). Developments in Nonholonomic Control Problems. *IEEE Control Systems*, 15(6):20–36. (ref. page 3)

Kühne, F., Lages, W. F., and Silva, J. (2005). Mobile Robot Trajectory Tracking using Model Predictive Control. In *Proceedings of the Latin-American Robotics Symposium*, pages 1–7. (ref. pages 5 and 44)

Langson, W., Chryssochoos, I., Raković, S., and Mayne, D. Q. (2004). Robust Model Predictive Control using Tubes. *Automatica*, 40(1):125–133. (ref. page 87)

Maimone, M., Cheng, Y., and Matthies, L. (2007). Two Years of Visual Odometry on the Mars Exploration Rovers. *Journal of Field Robotics*, 24(3):169–186. (ref. pages 1 and 11)

Marruedo, D. L., Alamo, T., and Camacho, E. F. (2002). Input-to-state Stable MPC for Constrained Discrete-time Nonlinear Systems with Bounded Additive Uncertainties. In *Proceedings of the IEEE Conference on Decision and Control*, volume 4, pages 4619–4624. (ref. pages 86 and 87)

Marshall, J., Barfoot, T., and Larsson, J. (2008). Autonomous Underground Tramming for Center-articulated Vehicles. *Journal of Field Robotics*, 25(6-7):400–421. (ref. pages 1 and 2)

Matsuko, J. and Borrelli, F. (2012). Scenario-based Approach to Stochastic Linear Predictive Control. In *Proceedings of the Conference on Decision and Control*, pages 5194–5199. (ref. page 72)

Matthies, L. (1989). *Dynamic Stereo Vision*. PhD thesis, Carnegie-Mellon University. (ref. page 11)

Matthies, L. and Shafer, S. (1987). Error Modeling in Stereo Navigation. *IEEE Journal of Robotics and Automation*, 3(3). (ref. page 11)

Mayne, D. Q. (2014). Model Predictive Control: Recent Developments and Future Promise. *Automatica*, 50(12):2967–2986. (ref. pages 4, 9, and 86)

Mayne, D. Q., Kerrigan, E. C., Van Wyk, E., and Falugi, P. (2011). Tube-based Robust Nonlinear Model Predictive Control. *International Journal of Robust and Nonlinear Control*, 21(11):1341–1353. (ref. page 87)

Meier, F., Hennig, P., and Schaal, S. (2014). Efficient Bayesian Local Model Learning for Control. In *Proceedings of the IEEE International Conference on Intelligent Robotics Systems*, pages 2244–2249. (ref. pages 45 and 55)

Moravec, H. (1980). *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*. PhD thesis, Stanford University, Stanford, CA. (ref. page 11)

Munoz, V., Ollero, A., Prado, M., and Simon, A. (1994). Mobile Robot Trajectory Planning with Dynamic and Kinematic Constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2802–2807. (ref. page 32)

Nagatani, K., Noyori, T., and Yoshida, K. (2013). Development of Multi-D.O.F. Tracked Vehicle to Traverse Weak Slope and Climb up Rough Slope. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 2849–2854. (ref. page 35)

Nguyen-Tuong, D. and Peters, J. (2011). Model Learning for Robot Control: A Survey. *Cognitive processing*, 12(4):319–340. (ref. pages 5 and 69)

Nguyen-Tuong, D., Peters, J., and Seeger, M. (2009). Local Gaussian Process Regression for Real Time Online Model Learning. *Advances in Neural Information Processing Systems*, 22:1193–1200. (ref. pages 45 and 55)

Nistér, D. (2005). Preemptive RANSAC for Live Structure and Motion Estimation. *Machine Vision and Applications*, 16(5):321–329. (ref. page 14)

Nocedal, J. and Wright, S. (1999). *Numerical Optimization*, volume 2. Springer New York. (ref. pages 48 and 76)

Oriolo, G. (2014). Wheeled Robots. In *Encyclopedia of Systems and Control*, pages 1–9. Springer. (ref. page 3)

Oriolo, G., De Luca, A., and Vendittelli, M. (2002). WMR Control via Dynamic Feedback Linearization: Design, Implementation, and Experimental Validation. *IEEE Transactions on Control Systems Technology*, 10(6):835–852. (ref. page 4)

Oriolo, G., Panzieri, S., and Ulivi, G. (1998). An Iterative Learning Controller for Nonholonomic Mobile Robots. *International Journal of Robotics Research*, 17(9):954–970. (ref. page 19)

Ostafew, C., Collier, J., Schoellig, A. P., and Barfoot, T. D. (2014a). Speed Daemon: Experience-Based Mobile Robot Speed Scheduler. In *Proceedings of the Conference on Computer and Robot Vision*, pages 56–62.

Ostafew, C., Collier, J., Schoellig, A. P., and Barfoot, T. D. (2015a). Learning-based Nonlinear Model Predictive Control to Improve Vision-based Mobile Robot Path Tracking. *Journal of Field Robotics*, 33(1):133–152.

Ostafew, C., Schoellig, A. P., and Barfoot, T. D. (2013). Visual Teach and Repeat, Repeat, Repeat: Iterative Learning Control to Improve Mobile Robot Path Tracking in Challenging Outdoor Environments. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pages 176–181.

Ostafew, C., Schoellig, A. P., and Barfoot, T. D. (2014b). Learning-based Nonlinear Model Predictive Control to Improve Vision-Based Mobile Robot Path-Tracking in Challenging Outdoor Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4029–4036.

Ostafew, C., Schoellig, A. P., and Barfoot, T. D. (2015b). Conservative to Confident: Treating Uncertainty Robustly Within Learning-based Control. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 421–427.

Ostafew, C., Schoellig, A. P., and Barfoot, T. D. (2016). Robust Constrained Learning-based NMPC Enabling Reliable Mobile Robot Path Tracking. *International Journal of Robotics Research*. To Appear: IJR-15-2464.

Paton, M., MacTavish, K., Ostafew, C., and Barfoot, T. D. (2015). It's Not Easy Seeing Green: Lighting-resistant Stereo Visual Teach & Repeat using Color-Constant Images. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1519–1526.

Peters, S. and Iagnemma, K. (2008). Mobile Robot Path Tracking of Aggressive Maneuvers on Sloped Terrain. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 242–247. (ref. pages 5 and 45)

Pin, G., Raimondo, D., Magni, L., and Parisini, T. (2009). Robust Model Predictive Control of Nonlinear Systems with Bounded and State-dependent Uncertainties. *IEEE Transactions on Automatic Control*, 54(7):1681–1687. (ref. page 87)

Prado, M., Simón, A., Carabias, E., Perez, A., and Ezquerro, F. (2003). Optimal Velocity Planning of Wheeled Mobile Robots on Specific Paths in Static and Dynamic Environments. *Journal of Robotic Systems*, 20(12):737–754. (ref. pages 32 and 33)

Prado, M., Simón, A., and Ezquerro, F. (2002). Velocity, Acceleration and Deceleration Bounds for a Time-optimal Planner of a Wheeled Mobile Robot. *Robotica*, 20(2):181–193. (ref. page 32)

Purwin, O. and D'Andrea, R. (2006). Trajectory Generation and Control for Four Wheeled Omnidirectional Vehicles. *Robotics and Autonomous Systems*, 54(1):13–22. (ref. pages 32 and 33)

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A Unifying View of Sparse Approximate Gaussian Process Regression. *The Journal of Machine Learning Research*, 6:1939–1959. (ref. page 45)

Raimondo, D. M., Limon, D., Lazar, M., Magni, L., and Camacho, E. F. (2009). Min-Max Model Predictive Control of Nonlinear Systems: A Unifying Overview on Stability. *European Journal of Control*, 15(1):5 – 21. (ref. page 72)

Rasmussen, C. E. (2006). *Gaussian Processes for Machine Learning*. The MIT Press. (ref. pages 45, 51, 52, and 73)

Rasmussen, C. E. and Ghahramani, Z. (2002). Infinite Mixtures of Gaussian Process Experts. *Advances in Neural Information Processing Systems*, 2:881–888. (ref. page 45)

Rawlings, J. B. and Mayne, D. Q. (2009). *Model Predictive Control: Theory and Design*. Nob Hill Publishing. (ref. pages 4, 8, 44, 46, and 86)

Rossmann, J., Schluse, M., Schlette, C., Buecken, A., Krahwinkler, P., and Emde, M. (2009). Realization of a Highly Accurate Mobile Robot System for Multi Purpose Precision Forestry Applications. In *Proceedings of the IEEE International Conference on Advanced Robotics*, pages 1–6. (ref. page 1)

Samson, C. and Ait-Abderrahim, K. (1991). Feedback control of a nonholonomic wheeled cart in cartesian space. *In Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, 2:1136–1141. (ref. pages 3 and 21)

Schaal, S. and Atkeson, C. (2010). Learning Control in Robotics. *IEEE Robotics & Automation Magazine*, 17(2):20–29. (ref. page 5)

Schildbach, G., Fagiano, L., Frei, C., and Morari, M. (2014). The Scenario Approach for Stochastic Model Predictive Control with Bounds on Closed-loop Constraint Violations. *Automatica*, 50(12):3009–3018. (ref. page 72)

Schoellig, A. P., Mueller, F., and D'Andrea, R. (2012). Optimization-Based Iterative Learning for Precise Quadrocopter Trajectory Tracking. *Autonomous Robots*, 33:103–127. (ref. pages 18 and 19)

Scokaert, P. O. M. and Mayne, D. Q. (1998). Min-Max Feedback Model Predictive Control for Constrained Linear Systems. *IEEE Transactions on Automatic Control*, 43(8):1136–1142. (ref. page 72)

Snelson, E. and Ghahramani, Z. (2007). Local and Global Sparse Gaussian Process Approximations. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 524–531. (ref. page 45)

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., et al. (2006). Stanley: The Robot that Won the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9):661–692. (ref. pages 1, 2, and 33)

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Galatali, T., Geyer, C., et al. (2008). Autonomous Driving in Urban Environments: Boss and the Urban Challenge. *Journal of Field Robotics*, 25(8):425–466. (ref. page 1)

Waheed, I. and Fotouhi, R. (2009). Trajectory and Temporal Planning of a Wheeled Mobile Robot on an Uneven Surface. *Robotica*, 27(4):481–498. (ref. page 32)

Witsenhausen, H. S. (1968). A Minimax Control Problem for Sampled Linear Systems. *IEEE Transactions on Automatic Control*, 13(1):5–21. (ref. page 72)

Xie, F. and Fierro, R. (2008). First-State Contractive Model Predictive Control of Nonholonomic Mobile Robots. In *Proceedings of the American Control Conference*, pages 3494–3499. (ref. page 45)