Title
Data-Driven Methods for Safe Task Transfer Learning with Model Predictive Control

Author
Vallon, Charlott Sophie

Publication Date
2021

Peer reviewed|Thesis/dissertation

Data-Driven Methods for Safe Task Transfer Learning with Model Predictive Control

by

Charlott Sophie Vallon

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Francesco Borrelli, Chair
Assistant Professor Mark Mueller
Associate Professor Benjamin Recht

Fall 2021

Data-Driven Methods for Safe Task Transfer Learning with Model Predictive Control

Abstract

Data-Driven Methods for Safe Task Transfer Learning with Model Predictive Control

by

Charlott Sophie Vallon

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Professor Francesco Borrelli, Chair

The topic of learning in control has garnered much attention in recent years, with many researchers proposing methods for combining data-based learning methods with more traditional control design. For systems repeatedly performing a single task, iterative learning controllers provide a structured, model-based way of using collected data to iteratively improve on a particular task while guaranteeing constraint satisfaction during the learning process. However, it remains difficult to design model-based learning controllers that both perform well and act safely in a variety of changing or unknown environments.

This dissertation considers a particular problem: how to use stored trajectory data from a system solving an initial set of tasks in order to design a controller that performs a related task in a new environment both safely (satisfying all new constraints) and effectively (maximizing a desired objective). We approach this question from a Model Predictive Control (MPC) perspective. Fundamentally, we ask how traditionally model-based terminal sets and cost functions of the MPC may be replaced with data-driven counterparts while maintaining feasibility guarantees that classical MPC theory offers. We consider various instantiations of the changing environment problem, including known or unknown task environments with time-invariant or time-varying constraints. For each scenario, we propose approaches for safe and effective control design. Using tools from MPC theory, optimization, and statistics, we demonstrate the safety (with high probability) for each proposed control scheme. The presented control approaches are validated in simulations and experiments in a variety of applications, including autonomous racing, robotic manipulation, and computer game tasks. The evaluations demonstrate the potential for safely integrating data in model-based control design.

For all my wonderful families.

And especially for Jacqueline and Sean,
my very greatest joys and inspirations,
for their unconditional support.

Contents

List of Figures

List of Tables

Acknowledgments

I am deeply grateful for the generous financial support of the National Institute for Standards and Technology and the GMSE Fellowship throughout my graduate studies. The work presented in this dissertation is also based upon research generously supported by the National Science Foundation under Grant No. 191853 and the Office of Naval Research Grant No. N00014-18-1-2833. Any conclusions herein do not necessarily reflect their views.

No paragraph here could adequately express the immense gratitude I feel for my advisor, Francesco Borrelli, but his feedback, mentorship, and constant encouragement during the last five years have made me an infinitely better researcher and person. I would also like to thank Professor Koushil Sreenath and Professor Laurent El Ghaoui for being on my Qualifying Exam committee, and Professor Mark Mueller and Professor Benjamin Recht for serving on both my Qualifying and Dissertation Committees.

Being a part of the Model Predictive Control lab has been the happiest part of graduate school. I am very thankful to my labmates for their company during late-night paper writing sessions, assistance in deciphering Borrelli's feedback, and the general support of all my endeavors (except vegetarianism). I am so proud to be academically and socially associated with you, and you all are what I will miss most about Berkeley.

They say beauty is in the eyes of the beholder, and never is this more true than for Etcheverry Hall. I would very much like to thank all my Etcheverry friends, who have made the building's monochrome hallways feel like a cozy home. Thank you especially to Monica Li for not letting me stop running before we reached the twelve mile mark (a true story and a metaphor); to Erin Kunz for always being there for me and always knowing what I mean and never laughing at my Peets order; to Tony Zheng for being an all-around inspiring human, co-GSI, mentor, Wordle player, and chef; to Ed Zhu for being the most excellent desk neighbor and a true friend through small (a bug in the code) and large (a bug in the wall) problems; to Brian Reli-Bri Cera for being my project partner every time and rolling with my last-minute changes to the presentation script (also every time); and to Kate (Kathy) Schweidel for being my sounding board, live-sharer, and other half. What a complete joy to know all of you and to have you in my corner.

Thank you so very much to all my family for the phone calls, visits, care packages, love, and support throughout the ups and downs of these years at Berkeley. A special thanks also to the Anderson family for providing a home away from home, and especially to Teresa, who to me represents everything right in this world and who I try hard to emulate.

To my sister Jackie: there is no responsibility I take more seriously than being your older sister, but you make it very easy and extremely fun. I am so proud of you, so grateful to have you as my best friend, and thankful for our sister days. And, finally, thank you to Sean Anderson, for taking care of me and making these last four years my happiest and most adventurous. You remain my greatest luck and best decision. This dissertation is dedicated to the two of you. I don't see how I could have achieved any of this without you, and I'm so thankful.

Chapter 1

Introduction

1.1 Background

Control design for systems repeatedly performing a single task has been studied extensively. Such problems arise frequently in practical applications [20, 130], and examples range from autonomous cars racing around a track [65, 22, 101] to robotic system manipulators performing industrial warehouse tasks [53, 3, 126, 129]. In these cases, the system repeatedly performs the same task under a constant set of environment constraints.

Iterative Learning Controllers (ILCs) aim to autonomously improve a system's closed-loop reference tracking performance at each iteration of a repeated task while rejecting periodic disturbances [20, 108]. In classical ILC, the controller uses tracking error data from previous task iterations to better track a provided reference trajectory during the current iteration. ILC methods been used in a variety of applications, ranging from quadrotor control [35] to additive manufacturing [4], robotics [52], and ventricular assist devices [58]. Recent work has also explored reference-free ILC for applications whose goals are better defined through a performance metric, rather than a reference trajectory. Examples include autonomous racing tasks (e.g. "minimize lap time") [101, 56], or optimizing flight paths for tethered energy-harvesting systems (e.g. "maximize average power generation") [29]. In these cases, the controller again uses previous iteration data to improve closed-loop performance, but with respect to the chosen performance metric rather than reference tracking error.

Because ILC policies are trained to attenuate task-specific disturbances or avoid task-specific obstacles, the learned ILC policy is generally not cost-effective, or even feasible, if the task environment or reference trajectory changes [101, 123]. Several approaches have been proposed for improving ILC performance in these situations, including representing tasks in terms of basis functions [140, 51], enforcing sparsity in the ILC policy to prevent overfitting to a particular task [84], or training neural networks to predict the task-specific ILC input given an environment observation [136, 88]. Generally, if the task goal or environment changes, a new reference trajectory with which to initialize the ILC needs to be designed to match the

new task.

Many methods exist to use data collected from a task to efficiently find trajectories for variations of that task. Such a trajectory could then, for example, be used to initialize a new ILC or to solve the task directly. In the simplest case, two tasks differ only in their environment constraints (e.g. different track shape in a racing task), while task goals (e.g. "reach the finish line") and performance objectives (e.g. "minimize lap time") remain the same. These approaches can loosely be considered in three groups: model-free transfer learning, trajectory libraries, and model-based generalizable policies.

AI/Transfer learning

In the Artificial Intelligence and Reinforcement Learning communities, the ability to generate a control policy which performs well under different environment conditions is a common challenge, often referred to as "generalization" or "task transfer learning" [119, 133, 138]. The challenge results because policies learned from repeated tasks are trained to avoid specific environmental obstacles or disturbances; if the task environment (such as the location of the obstacles) changes, purely data-driven policies are generally no longer guaranteed to be feasible. Because these approaches are typically model-free, they lack a structured way of adapting to new changes in the environment. As a result, model-free approaches typically focus on minimizing a policy's performance loss between environments [27, 80], rather than guaranteeing feasibility of the policy in a new environment.

Similar strategies are proposed in [45, 117]. The authors in [63] propose learning a feature-based value function approximation in the space of shared task features. The learned function then provides an initial guess for the new task's true value function, which can be used to initialize a standard RL method. These mappings must be learned and evaluated for each saved state, scaling poorly to long horizon tasks. Furthermore, these methods offer no guarantees for safety in the new task. Because the safety of the policy will be of high importance in this dissertation, we will not include further details here on model-free approaches; however, any of the cited works are excellent references for the interested reader.

Trajectory libraries

A variety of model-based methods have been suggested for finding feasible trajectories or policies for new tasks using stored data from related tasks. One popular approach relies on building trajectory libraries [10, 113, 72, 116], and adapting the stored trajectories online to the changed constraints of the new tasks.

For example, the authors of [82] design new walking gaits for a bipedal robot navigating across stepping stones by linearly interpolating trajectories from a library of previously recorded asymptotically stable periodic walking gaits. A trajectory library built using differential dynamic programming is used in [72] to design a controller for balance control in a humanoid robot. In this approach, a trajectory is selected from the library at each time step based on current task parameter estimates and a k-nearest neighbor selection scheme.

A similar method is explored in [116], where differential dynamic programming is combined with receding horizon control. The authors of [135] consider a set of actions and corresponding motion primitives for iterative teleoperative tasks. Given a new user-provided input, probabilistic inferences are made over the respective set of locally feasible trajectories. Similar approaches considering probabilistic distributions over trajectory libraries are proposed for robotic manipulators and autonomous vehicles in [87] and [137], respectively.

In [113], manually chosen environment features are used to divide a task and create a library of local trajectories in relative state space frames. These trajectories are then pieced back together in real-time according to the features of the new task environment. The authors in [11] propose a two-step approach: a desired path planning method is run in parallel with a retrieve-and-repair algorithm that selects an appropriate trajectory from a trajectory library, and adapts the trajectory to the constraints of a new task. Retrieve and repair was shown to decrease overall planning time, but requires checking for constraint violations at each point along a retrieved trajectory.

The authors in [5] propose piecing together stored trajectories corresponding to discrete system dynamics only at states of dynamics transition. Thus, this method can avoid verifying constraint satisfaction at each point of the trajectory. However, this method only applies to discontinuities in system dynamics, and does not generalize to other task variations.

While trajectory library methods decrease planning time, they do require maintaining trajectory libraries and verifying or interpolating the saved trajectories at each new time step, which may be unneccessary and inefficient, and cannot typically a priori guarantee constraint satisfaction in the new task.

Generalizable policies

A third type of approach learns model-based, generalizable policies from stored task data. These methods can sometimes guarantee feasibility (at least with high likelihood) by incorporating system models in the policy design [14, 50, 114, 90], and here, too, approaches have been proposed for applications ranging from autonomous vehicles to robotic manipulation [87, 42, 123].

In [90], a fixed map is learned between a given reference trajectory and the input sequence required for a linear time-invariant system to track that reference trajectory. Once learned, this map can be used to determine an input sequence that lets the linear system track a new reference trajectory. Thus this approach defines a policy given a new reference trajectory, but does not directly provide the trajectory itself. No adaptations for nonlinear systems are provided.

The authors of [31] consider linear hybrid systems. Data is collected in individual modes, and a polynomial optimization problem is formulated to find a stabilizing controller for arbitrary switching sequences. The authors of [43] learn a mapping between a robot gripper pose performing the same task with different tools, based on online human corrections. While this method was effective in demonstrations, it required human supervision and intervention, and provided no safety guarantees.

MPC-based approach

In this dissertation, we approach the task transfer learning problem from an MPC perspective. Our goal will be to use stored trajectory data from previous tasks, collected using an ILC or other controller, in order to find MPC policies for tasks with different environmental constraints. We specifically desire that these MPC policies are:

1. safe (satisfy all constraints), with constraint satisfaction guarantees before beginning the new control task,
2. effective (optimize a desired objective), and
3. efficient to store and evaluate.

To the best of our knowledge, this is the first work that considers such a systematic MPC-focused approach. We formalize our problem statement next, and conclude this section with a dissertation outline.

1.2 Problem Formulation

We consider a discrete-time system with dynamical model

$$x_{k+1} = f(x_k, u_k), \tag{1.1}$$

subject to system state and input constraints

$$x_k \in \mathcal{X},\ u_k \in \mathcal{U}. \tag{1.2}$$

The vectors $x_k \in \mathbb{R}^{n_x}$ and $u_k \in \mathbb{R}^{n_u}$ collect the states and inputs at time k.

Tasks

The system (1.1) solves a series of n different control tasks $\{\mathcal{T}^1, \ldots, \mathcal{T}^n\}$. Each control task $\mathcal{T}^i$ is defined by the tuple

$$\mathcal{T}^i = \{\mathcal{X}, \mathcal{U}, \mathcal{P}^i, \mathbf{\Theta}^i\}, \tag{1.3}$$

where $\mathcal{X}$ and $\mathcal{U}$ are the system state and input constraints (1.2). $\mathcal{P}^i \subset \mathcal{X}$ denotes the task target set the system (1.1) needs to reach in order to complete task $\mathcal{T}^i$; thus, the task ends when $x_k \in \mathcal{P}^i$. Lastly, $\mathbf{\Theta}^i$ is an environment descriptor function, formalized below. Note that two tasks can differ either in their goal and/or their environment descriptor function.

(a) The environment descriptor function $\boldsymbol{\Theta}$ gives a local description of the task environment, such as a camera image of the track at a particular location.

(b) From this description, local environment constraints such as lane boundaries the vehicle must observe, can be determined.

Figure 1.1: Example of an environment descriptor function for a racing task.

Task Environments

The n solved control tasks $\{\mathcal{T}^1, \ldots, \mathcal{T}^n\}$ take place in various task environments, parameterized by different environment descriptor functions $\{\boldsymbol{\Theta}^1, \ldots, \boldsymbol{\Theta}^n\}$. In each control task the system model (1.1) and constraints (1.2) are identical. However, the environment descriptor function generates additional task-specific environmental state constraints.

For each task $\mathcal{T}^i$, the environment descriptor function $\boldsymbol{\Theta}^i$ maps the state x_k at time k to a description of the local task environment, denoted $\boldsymbol{\Theta}^i(\cdot)$. The set of states satisfying the environmental constraints imposed by this local task environment are denoted by $E(\boldsymbol{\Theta}^i(\cdot))$. Future chapters will make specific assumptions on the arguments of $\boldsymbol{\Theta}^i$ (e.g. state-dependence, time-invariance), which is why we use only general notation here.

Note that whenever $\boldsymbol{\Theta}^i$ is time-varying, the combined constraints may also be state- and time-dependent. We write these joint system and environment state constraints as

$$x_k \in \mathcal{X}(\boldsymbol{\Theta}^i(\cdot)) = E(\boldsymbol{\Theta}^i(\cdot)) \cap \mathcal{X}. \tag{1.4}$$

For notational simplicity, wherever it is obvious we will drop the argument dependence and refer to the combined system and environment constraints (1.4) simply as $\mathcal{X}(\boldsymbol{\Theta}^i)$.

Racing Example

Examples of local task environment descriptions $\boldsymbol{\Theta}^i(\cdot)$ include camera images, the coefficients of a polynomial describing a race track's lane boundaries, or simple waypoints for tracking. Consider, for example, an autonomous racing task where a car has to drive around a track

as quickly as possible, with the task target set $\mathcal{P}$ defined as the finish line. We will refer back to this example repeatedly throughout this work.

Here, the vehicle-specific state and input constraints (1.2) may include acceleration and steering limits imposed by the construction of the vehicle itself. The environment descriptor function $\boldsymbol{\Theta}^i$ may map a state along the track to a frontview camera image of the track at that point in space and time (Fig. 1.1a). From this camera environment description, we can then extract the additional environmental constraints $E(\boldsymbol{\Theta}^i(\cdot))$, such as lane boundaries the vehicle needs to remain in, which are dictated by the environment rather than the vehicle (Fig. 1.1b).

Task Executions

A feasible execution of the task $\mathcal{T}^i$ in an environment parameterized by $\boldsymbol{\Theta}^i$ is defined as a pair of state and input trajectories

$$\begin{aligned} \mathrm{Ex}(\mathcal{T}^i, \boldsymbol{\Theta}^i) &= [\boldsymbol{x}^i, \boldsymbol{u}^i] \\ \boldsymbol{x}^i &= [x_0^i, x_1^i, ..., x_{D^i}^i],\ x_k^i \in \mathcal{X}(\boldsymbol{\Theta}^i)\ \ \forall k \in [0, D^i],\quad x_{D^i}^i \in \mathcal{P}^i \\ \boldsymbol{u}^i &= [u_0^i, u_1^i, ..., u_{D^i}^i],\ u_k^i \in \mathcal{U}\ \ \forall k \in [0, D^i], \end{aligned} \tag{1.5}$$

where $\boldsymbol{u}^i$ collects the inputs applied to the system and $\boldsymbol{x}^i$ is the resulting state evolution. D^i is the duration of the execution of task $\mathcal{T}^i$. The final state of a feasible task execution, $x_{D^i}^i$, is in the task's target set $\mathcal{P}^i \subset \mathcal{X}(\boldsymbol{\Theta}^i)$. Certain techniques proposed in this work will make additional assumptions about the control objectives when the executions were created, but these will be introduced in the appropriate sections.

In the racing task, feasible executions would be vehicle state and input trajectories obeying the lane boundaries described by the environment descriptor function.

Problem Definition

Given a dynamical model (1.1) with state and input constraints (1.2), (1.4), and a collection of feasible executions (1.5) that solve a series of n control tasks, $\{\mathrm{Ex}(\mathcal{T}^1, \boldsymbol{\Theta}^1), ..., \mathrm{Ex}(\mathcal{T}^n, \boldsymbol{\Theta}^n)\}$, our aim is to find a data-driven MPC policy $u = \pi(x, \cdot)$ that results in a feasible and high-performance execution of a new task in a new environment: $\mathrm{Ex}(\mathcal{T}^{n+1}, \boldsymbol{\Theta}^{n+1})$. Such a policy

will be of the form:

$$\mathbf{u}^\star = \underset{u_{k|k},\ldots,u_{k+N-1|k}}{\arg\min} \quad \sum_{t=0}^{N-1} p_k(x_{k+t|k}, u_{k+t|k}, \cdot) + q(x_{k+N|k}, \cdot) \tag{1.6}$$

$$\text{s.t.} \quad x_{k+t+1|k} = f(x_{k+t|k}, u_{k+t|k}) \tag{1.7}$$

$$u_{k+t|k} \in \mathcal{U},\ \forall t \in \{0, \ldots, N-1\} \tag{1.8}$$

$$x_{k+t|k} \in \mathcal{X}(\boldsymbol{\Theta}^{n+1}),\ \forall t \in \{0, \ldots, N\} \tag{1.9}$$

$$x_{k|k} = x_k \tag{1.10}$$

$$x_{k+N|k} \in \mathcal{X}_N \tag{1.11}$$

$$u_k = \pi(x_k, \cdot) = u^\star_{k|k}. \tag{1.12}$$

The policy (1.6-1.12) searches for the input sequence $\boldsymbol{u}^\star$ of length N that minimizes a cost (1.6) over the planning horizon N while satisfying the system dynamics (1.7) and constraints (1.8-1.9). The predicted state sequence is initialized with the current state x_k (1.10), and the last predicted state, $x_{k+N|k}$, is constrained to be in a chosen terminal set (1.11). At time k, the N-step input sequence that minimizes the chosen objective function (1.6) is calculated and the first input $u^\star_{0|k}$ is applied to the system. A new N-step input sequence is then calculated again at time $k+1$, potentially using updated information about the system or environment. This iterative planning and executing approach is known as receding horizon control.

In this dissertation, we will present multiple approaches to using stored data from previous tasks (1.5) to design MPC cost functions (1.6) and terminal constraints (1.11). In MPC theory, the cost function and terminal constraints are typically used to demonstrate recursive feasibility of the controller. Recursive feasibility refers to the desired property that if the MPC controller (1.6-1.12) is applied repeatedly to a dynamical system, the optimization problem (1.6-1.11) will always have at least one feasible solution. This ensures that the MPC will always be able to find a constraint-satisfying input (1.12) to apply to the system. Future chapters will analyze this property, and how to design terminal constraints that induce such a property, in more detail.

If feasibility were the only goal, one approach would be to use a robust, conservative controller to solve the new task—for example, tracking the centerline of the road at a very slow longitudinal speed. However, we would also like to solve the new task *well*. Thus, in addition to satisfying the new environment constraints, the execution should try to minimize a desired objective function $J(\boldsymbol{x}^{n+1}, \boldsymbol{u}^{n+1})$. Assumptions about J will be discussed in future sections.

Remark 1. *For notational simplicity, we write that the collected data set contains one execution each from every previously solved task. However, in practice one may of course collect multiple executions of each task $\mathcal{T}^i$. In this case, the executions can simply be stacked, and the procedures proposed in this dissertation can proceed as described.*

Remark 2. *Throughout this dissertation, we assume perfect knowledge of the system dynamics model (1.7). The approaches presented can also be straightforwardly extended to models with bounded uncertainty.*

1.3 Dissertation Outline and Contributions

This dissertation discusses four different methods to approaching the changing environment problem.

In Chapter 2, a task decomposition approach based on iterative learning MPC is introduced. Task decomposition proposes breaking larger tasks into sequences of subtasks, so that any two related tasks differ only in the sequence ordering of the subtasks. Thus, similar to trajectory library methods, the question becomes whether stored subtask trajectories can be pieced together to form a feasible task trajectory through the constraints of the new task. There are various ways trajectories can be stitched together; we propose an efficient algorithm that adapts the original subtask trajectories (and original task policy) only at points of subtask transition, rather than along the entire previous trajectory, via one-step controllability problems. If the proposed algorithm converges, it provides both a trajectory as well as a policy that solve the new task while satisfying all system and environmental constraints.

Chapter 3 describes how to reduce the computational complexity of task decomposition, while increasing the domain of the resulting policy, for piecewise linear systems with piecewise convex state and environment constraints. The required offline calculation time is improved by reducing the number of one-step controllability problems performed to adapt stored subtask trajectories, and reformulating each into a convex optimization problem for efficient calculation.

In Chapter 4, we attempt to generalize the key ideas of task decomposition. Instead of storing and piecing together recorded subtask trajectories, we train Gaussian processes to learn data-driven estimates of the stored trajectories. For such an approach, rather than requiring that tasks are composed of the same set of subtasks, we simply require that each task's environment constraints evolve according to a task-invariant, but potentially time-varying, function. This allows us to design controllers for entirely new kinds of task environments. We demonstrate that for linear time-varying systems with convex constraints, the proposed algorithm results in a control policy that satisfies all (potentially time-varying) constraints of the new task.

Lastly, whereas Chapters 2-4 focus on using stored task trajectories to estimate controllable sets and adapt these to the constraints of the new task, Chapter 5 uses stored data to learn generalizable control strategies. At each time step, based on a local forecast of the new task environment, the learned strategy consists of a target region in the state space and input constraints to guide the system evolution to the target region. These strategies are learned from stored trajectories of previous tasks, and then applied to the new task in real-time. The target regions are used as terminal sets by a low-level model predictive controller. By

working in conjunction with a model-based safety controller, this method is guaranteed to satisfy all constraints and results in a feasible task execution, despite only relying on a local forecast of the new task (i.e. the entire task description is not necessarily known a priori).

Each proposed method is evaluated in a variety of applications. Concluding remarks comparing the effectiveness and efficiencies of the control designs are written in Chapter 6.

1.4 List of Publications

The results presented in this dissertation have appeared in a number of publications authored by the author of this dissertation. In particular:

- Chapter 2 is based on:
  - C. Vallon and F. Borrelli. "Task decomposition for iterative learning model predictive control." In: *2020 IEEE American Control Conference (ACC)*. 2020, pp. 2024-2029.
- Chapter 3 is based on:
  - C. Vallon and F. Borrelli. "Task decomposition for MPC: A computationally efficient approach for linear time-varying systems." In: *2020 IFAC World Congress*. 2020, pp. 4240-4245.
- Chapter 5 is based on:
  - C. Vallon and F. Borrelli. "Data-driven hierarchical predictive learning in unknown environments." In: *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. 2020, pp. 104-109.
  - C. Vallon and F. Borrelli. "Data-driven strategies for hierarchical predictive control in unknown environments." To appear in: *2021 Transactions on Automation Science and Engineering (TASE)*.

1.5 Preliminaries

The methods introduced in this dissertation make use of the following definitions.

Definition 1. *For a given set $\mathcal{R}$, the* N-step controllable set $\mathcal{K}_N(\mathcal{R})$ *of a system* (1.1) *subject to constraints* (1.2)-(1.4) *is defined recursively as:*

$$\begin{aligned}
&\text{Pre}(\mathcal{R}) = \{x : \exists u \in \mathcal{U} : f(x,u) \in \mathcal{R}\} \\
&\mathcal{K}_0(\mathcal{R}) = \mathcal{R} \\
&\mathcal{K}_j(\mathcal{R}) = \text{Pre}(\mathcal{K}_{j-1}(\mathcal{R})) \cap \mathcal{X},\ j \in \{1, ..., N\}\,.
\end{aligned}$$

For all states in the N-step controllable set to $\mathcal{R}$ there exists a feasible input sequence of length N that drives the the system into $\mathcal{R}$ in N steps.

Definition 2. *A system (1.2) is* N-step controllable *from an initial state x_0 to a terminal state x_P if $x_0 \in \mathcal{K}_N(x_P)$.*

Definition 3. *The set $\mathcal{A}$ is* controllable *to a set $\mathcal{R}$ if there exists an $N > 0$ such that $\mathcal{A} \subseteq \mathcal{K}_N(\mathcal{R})$.*

Note that if an MPC can find a state trajectory ending in a terminal set that is controllable to a goal set, this also guarantees the existence of a feasible state trajectory from the current state to the goal set.

Definition 4. *The set $\mathcal{P}$ is a* control-invariant set *for a system* (1.1) *subject to constraints* (1.2)-(1.4) *if*

$$\forall x_k \in \mathcal{P},\ \exists u_k \in \mathcal{U}\ :\ x_{k+1} = f(x_k, u_k) \in \mathcal{P}.$$

Chapter 2

Task Decomposition

2.1 Introduction

Control design for systems repeatedly performing a single task has been studied extensively. Such problems arise frequently in practical applications [20, 130] and examples range from autonomous cars racing around a track [65, 22, 101] to robotic system manipulators [53, 3, 126, 129]. Iterative Learning Controllers (ILCs) aim to autonomously improve a system's closed-loop reference tracking performance at each iteration of a repeated task, while rejecting periodic disturbances [20, 69, 74]. In classical ILC, the controller uses tracking error data from previous task iterations to better track a provided reference trajectory during the current iteration. Recent work has also explored reference-free ILC strategies for tasks whose goals are better defined in terms of an economic metric, rather than a reference trajectory. The controller again uses previous iteration data to improve closed-loop performance with respect to the chosen performance metric. Examples include autonomous racing tasks (e.g. "minimize lap time") [101, 56], or optimizing flight paths for tethered energy-harvesting systems (e.g. "maximize average power generation") [29].

In this chapter, our objective is to find a feasible trajectory to smartly initialize an Iterative Learning Model Predictive Controller (ILMPC) [100] for a new task, using data from previous tasks. ILMPC is a type of reference-free ILC that uses a sampled safe set to design a model predictive control (MPC) policy for an iterative control task. The ILMPC safe set is initialized using a feasible task trajectory, and guides the policy towards improved performance with each subsequent iteration. Details on the ILMPC approach are provided in Sec. 2.2.

We consider a constrained nonlinear dynamical system and assume the availability of a dataset containing states and inputs corresponding to multiple iterations of a task $\mathcal{T}^1$. This dataset can be stored explicitly (e.g. by human demonstrations [28] or an iterative controller [100]) or generated by roll-out of a given policy (e.g. a hand-tuned controller). We introduce a Task Decomposition for ILMPC algorithm (TDMPC), and show how to use the stored $\mathcal{T}^1$ dataset to efficiently construct a non-empty ILMPC safe set for task $\mathcal{T}^2$ (a new

variation of $\mathcal{T}^1$), containing feasible trajectories for $\mathcal{T}^2$. TDMPC reduces the complexity to adapt trajectories from $\mathcal{T}^1$ to a new task $\mathcal{T}^2$ by decomposing task $\mathcal{T}^1$ into different modes of operation, called subtasks. The stored $\mathcal{T}^1$ trajectories are adapted to $\mathcal{T}^2$ only at points of subtask transition, by solving one-step controllability problems.

In this chapter, we:

1. provide a brief introduction to ILMPC as outlined in [100],
2. present how to build the aforementioned $\mathcal{T}^2$ safe set using the TDMPC algorithm, and
3. prove that the resulting safe set based ILMPC policy is feasible for $\mathcal{T}^2$, and the corresponding closed-loop trajectories have lower iteration cost compared to an ILMPC initialized using simple methods.

The results presented in this chapter have also appeared in:

- C. Vallon and F. Borrelli. "Task decomposition for iterative learning model predictive control." In: *2020 IEEE American Control Conference (ACC)*. 2020, pp. 2024-2029.

2.2 Problem Formulation

We begin by providing a brief overview of safe set based ILMPC [100]. This approach provides a method for iteratively improving the closed-loop cost of a system performing a single task repeatedly, beginning with a suboptimal initial trajectory and eventually converging to a local optimum.

Problem Setup

We consider a system

$$x_{k+1} = f(x_k, u_k), \tag{2.1}$$

where $f(x_k, u_k)$ is the dynamical model, subject to the constraints

$$x_k \in \mathcal{X},\ u_k \in \mathcal{U}. \tag{2.2}$$

The vectors $x_k \in \mathbb{R}^{n_x}$ and $u_k \in \mathbb{R}^{n_u}$ collect the states and inputs at time step k. We pick a set $\mathcal{P} \subset \mathcal{X}$ to be the target set for an iterative task $\mathcal{T}$, performed repeatedly by system (4.1) and defined by the tuple

$$\mathcal{T} = \{\mathcal{X}, \mathcal{U}, \mathcal{P}, \mathbf{\Theta}\}. \tag{2.3}$$

$\mathcal{P} \subset \mathcal{X}$ denotes the task target set the system (4.1) needs to reach in order to complete task $\mathcal{T}$, and $\mathbf{\Theta}$ the task's environment descriptor function.

Assumption 1. *$\mathcal{P}$ is a control invariant set (Def. 4, Sec. 1.5), i.e.:*

$$\forall x_k \in \mathcal{P},\ \exists u_k \in \mathcal{U}\ :\ x_{k+1} = f(x_k, u_k) \in \mathcal{P}.$$

Each task execution is referred to as an iteration. The goal of an ILMPC is to solve at each iteration the optimal task completion problem:

$$\begin{aligned} V^{\star}_{0\to D}(x_0) = \min_{D, u_0, \dots, u_{D-1}} & \sum_{k=0}^{D} p(x_k, u_k) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & x_k \in \mathcal{X}(\boldsymbol{\Theta}),\ u_k \in \mathcal{U}\ \ \forall k \geq 0 \\ & x_D \in \mathcal{P}, \end{aligned} \tag{2.4}$$

where $V^{\star}_{0\to D}(x_0)$ is the optimal cost-to-go from the initial state x_0, and $p(x_k, u_k)$ is a chosen stage cost (which may or may not help guide the system into $\mathcal{P}$). Note that (2.4) also finds the optimal task duration, D.

ILMPC Approach

We consider that the task $\mathcal{T}$ is solved repeatedly. At the j-th successful task iteration, the vectors

$$\mathrm{Ex}^j(\mathcal{T}) = [\boldsymbol{x}^j, \boldsymbol{u}^j] \tag{2.5a}$$

$$\boldsymbol{x}^j = [x_0^j, x_1^j, ..., x_{D^j}^j],\ x_k^j \in \mathcal{X}(\boldsymbol{\Theta})\ \ \forall k \in [0, D^j],\quad x_{D^j}^j \in \mathcal{P}$$

$$\boldsymbol{u}^j = [u_0^j, u_1^j, ...u_{D^j}^j],\ u_k^j \in \mathcal{U}\ \ \forall k \in [0, D^j], \tag{2.5b}$$

collect the inputs applied to system (1) and the corresponding state evolution. In (2.5), x_k^j and u_k^j denote the system state and control input at time k of the j-th iteration, and D^j is the duration of the j-th iteration.

After J number of iterations of task $\mathcal{T}$, we define the sampled safe state set and sampled safe input set as:

$$\mathcal{SS}^J = \left\{ \bigcup_{j=1}^{J} \boldsymbol{x}^j \right\},\ \mathcal{SU}^J = \left\{ \bigcup_{j=1}^{J} \boldsymbol{u}^j \right\}, \tag{2.6}$$

where $\mathcal{SS}^J$ contains all states visited by the system during the J previous task iterations, and $\mathcal{SU}^J$ the corresponding inputs applied at each of these states. Hence, for any state in $\mathcal{SS}^J$ there exists a feasible input sequence contained in $\mathcal{SU}^J$ to reach the goal set $\mathcal{P}$ while satisfying state and input constraints (4.2).

Similarly, we define the sampled cost set as:

$$\mathcal{SQ}^J = \left\{ \bigcup_{j=1}^{J} \boldsymbol{q}^j \right\} \tag{2.7a}$$

$$\boldsymbol{q}^j = [V^j(x_0^j), V^j(x_1^j), ..., V^j(x_{D^j}^j)], \tag{2.7b}$$

where $V^j(x_k^j)$ is the realized cost-to-go from state x_k^j at time step k of the j-th task execution:

$$V^j(x_k^j) = \sum_{i=k}^{D^j} p(x_i^j, u_i^j). \tag{2.8}$$

Note that (2.8) is not the best conceivable cost-to-go from x_k^j, but rather the closed-loop cost that was achieved in iteration j using the stored inputs in $\boldsymbol{u}^j$.

The safe set based ILMPC policy tries to solve (2.4) by using state and input data collected during past task iterations, stored in the sampled safe sets. At time k of iteration $J+1$, the ILMPC solves the optimal control problem:

$$\begin{aligned} V^{\text{ILMPC},J+1}(x_k^{J+1}) = \min_{j,u_{k|k},\ldots,u_{k+N-1|k}} \quad & \sum_{t=0}^{N-1} p(x_{k+t|k}, u_{k+t|k}) + V^j(x_{k+N|k}) \\ \text{s.t.} \quad & x_{k+t+1|k} = f(x_{k+t|k}, u_{k+t|k}) \\ & u_{k+t|k} \in \mathcal{U} \ \forall t \in \{0, \ldots, N-1\} \\ & x_{k+t|k} \in \mathcal{X}(\boldsymbol{\Theta}) \ \forall t \in \{0, \ldots, N\} \\ & x_{k|k} = x_k \\ & x_{k+N|k} \in\in \mathcal{SS}^J \cup \mathcal{P} \\ & j \in \{1, J\}, \end{aligned} \tag{2.9}$$

which searches for an input sequence over a chosen planning horizon N that controls the system (4.1) to the state in the sampled safe state set (2.6) or task target set $\mathcal{P}$ with the lowest cost-to-go (2.8) reachable from x_k. We then apply a receding horizon strategy:

$$u(x_k^j) = \pi^{\text{ILMPC}}(x_k^j) = u_{k|k}^\star. \tag{2.10}$$

A system (4.1) in closed-loop with (2.9-2.10) is guaranteed to result in a feasible task execution if $x_0 \in \mathcal{SS}^J$. At each time step, the ILMPC policy searches for the optimal input based on previous task data, leading to gradual performance improvement on the task as the sampled safe sets continue to grow with each subsequent iteration, until convergence to a global or local minimum. The key intuition of ILMPC is that safe sets (2.6) collected across previous iterations contain points which are all controllable to the task goal set $\mathcal{P}$; finding an MPC trajectory ending in $\mathcal{SS}^J$ ensures the recursive feasibility of the ILMPC policy. Note also that the controller (2.9-2.10) is only constrained to lie in the previous data at the last predicted state, $x_{k+N|k}$. This allows the ILMPC to explore new states that optimize closed-loop cost. For details on ILMPC, we refer to [100].

Drawbacks

ILMPC is an appealing method to find an effective policy for tasks that are difficult to solve outright (i.e. when the dynamics model (4.1) is not entirely known [101] or when

computation power must be minimized [103]). However, the sampled safe sets used in the ILMPC policy (2.10) must first be initialized to contain at least one feasible task execution. If the task constraints change even slightly, the trajectories stored in the safe sets will no longer be guaranteed to contain feasible executions to $\mathcal{P}$. A new ILMPC policy must be re-initialized for the reconfigured task, requiring a new initial task execution.

In this chapter, we introduce TDMPC, an approach for using data collected from an initially solved task $\mathcal{T}^1$ in order to efficiently find such an execution for a new, different task $\mathcal{T}^2$. This execution can be used to directly solve the task using (2.10), or to initialize a new ILMPC for $\mathcal{T}^2$. We approach this using subtasks, formalized in Sec. 2.3, and the concept of controllability.

2.3 Task Decomposition for ILMPC

Here we describe the intuition behind TDMPC, and provide an algorithm for the method.

Subtasks

Consider an iterative task $\mathcal{T}$ as in (2.3) and a sequence of M subtasks, where the i-th subtask $\mathcal{S}_i$ is the tuple

$$\mathcal{S}_i = \{\mathcal{X}_i, \mathcal{U}_i, \mathcal{R}_i, \mathbf{\Theta}_i\}. \tag{2.11}$$

We take $\mathcal{X}_i \subseteq \mathcal{X}$ as the subtask workspace, $\mathcal{U}_i \subseteq \mathcal{U}$ the subtask input space (the input constraints while the system is in the subtask), and $\mathcal{R}_i$ the set of transition states from the current subtask $\mathcal{S}_i$ workspace into the subsequent $\mathcal{S}_{i+1}$ workspace:

$$\mathcal{R}_i \subseteq \mathcal{X}_i = \{x \in \mathcal{X}_i : \exists u \in \mathcal{U}_i,\ f(x,u) \in \mathcal{X}_{i+1}\}.$$

Each subtask environment descriptor function $\mathbf{\Theta}_i$ is the restriction of the task environment descriptor function $\mathbf{\Theta}$ on the subtask work space $\mathcal{X}_i$. Here we do not make assumptions about the continuity of $\mathbf{\Theta}$ at points of subtask transition.

Within each subtask $\mathcal{S}_i$, the joint system and environmental state constraints are:

$$\mathcal{X}(\mathbf{\Theta}_i) = \{x \in \mathcal{X}_i \cap E(\mathbf{\Theta}_i(\cdot))\}. \tag{2.12}$$

Similar to our definition in (2.5), a successful subtask execution $\mathrm{Ex}(\mathcal{S}_i)$ is a trajectory of inputs and corresponding states evolving according to (4.1) while respecting state and input constraints (4.2), ending in the transition set:

$$\mathrm{Ex}^j(\mathcal{S}_i) = [\boldsymbol{x}_i^j, \boldsymbol{u}_i^j] \tag{2.13a}$$

$$\boldsymbol{x}_i^j = [x_0^j, x_1^j,\ ..., x_{D_i^j}^j],\ x_k \in \mathcal{X}(\mathbf{\Theta}_i)\ \ \forall k \in [0, D_i^j],\quad x_{D_i^j}^j \in \mathcal{R}_i \tag{2.13b}$$

$$\boldsymbol{u}_i^j = [u_0^j, u_1^j,\ ..., u_{D_i^j}^j],\ u_k \in \mathcal{U}_i\ \ \forall k \in [0, D_i^j],$$

where the vectors $\boldsymbol{u}_i^j$ and $\boldsymbol{x}_i^j$ collect the inputs applied to the system (4.1) and the resulting states, respectively, and x_k^j and u_k^j denote the system state and the control input at time k of subtask execution j. D_i^j is the duration of the j-th execution of subtask i. The final state of each successful subtask execution is in the subtask transition set, from which the system can evolve into new subtasks. For the sake of notational simplicity, we have written all subtask executions as beginning at time step $k = 0$.

We say the task $\mathcal{T}$ is an ordered sequence of the M subtasks (denoted $\mathcal{T} = \{\mathcal{S}_i\}_{i=1}^M$) if any j-th successful task execution (2.5) is the concatenation of successful subtask executions:

$$\begin{aligned}\mathrm{Ex}^j(\mathcal{T}) = [\mathrm{Ex}^j(\mathcal{S}_1), \mathrm{Ex}^j(\mathcal{S}_2), ..., \mathrm{Ex}^j(\mathcal{S}_M)] = [\boldsymbol{x}^j, \boldsymbol{u}^j] \\ \boldsymbol{x}^j = [\boldsymbol{x}_1^j, \boldsymbol{x}_2^j, \ ..., \boldsymbol{x}_M^j] \\ \boldsymbol{u}^j = [\boldsymbol{u}_1^j, \boldsymbol{u}_2^j, \ ..., \boldsymbol{u}_M^j] \\ f\Big(x^j_{D^j_{[1\to i]}}, u_{D^j_{[1\to i]}}\Big) \in \mathcal{X}_{i+1}, \ i \in [1, M-1] \\ x^j_{D^j_{[1\to M]}} \in \mathcal{R}_M,\end{aligned}$$

where $D^j_{[1\to i]}$ is the duration of the first i subtasks during the j-th task iteration. When the state reaches a subtask transition set, the system has completed subtask $\mathcal{S}_i$, and it transitions into the following subtask $\mathcal{S}_{i+1}$. The task is completed when the system reaches the last subtask's transition set, $\mathcal{R}_M$, which we consider as the task's control invariant target set $\mathcal{P}$.

Let $\mathcal{T}^1$ and $\mathcal{T}^2$ be different ordered sequences of the same M subtasks:

$$\mathcal{T}^1 = \{\mathcal{S}_i\}_{i=1}^M, \quad \mathcal{T}^2 = \{\mathcal{S}_{l_i}\}_{i=1}^M, \tag{2.15}$$

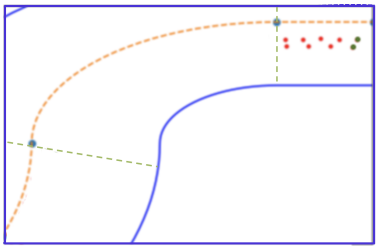
where the sequence $[l_1, l_2, ..., l_M]$ is a reordering of the sequence $[1, 2, ..., M]$. Assume non-empty sampled safe sets $\mathcal{SS}^J_{[1\to M]}$, $\mathcal{SU}^J_{[1\to M]}$, and $\mathcal{SQ}^J_{[1\to M]}$ (2.6, 2.7) containing executions that solve $\mathcal{T}^1$.

The goal of TDMPC is to use the executions stored in the sampled safe sets from $\mathcal{T}^1$ in order to find a feasible execution for $\mathcal{T}^2$, ending in the $\mathcal{T}^2$ goal set $\mathcal{R}_{l_M}$.

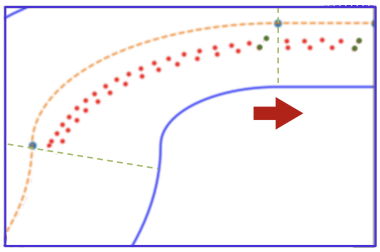
Algorithm

The key intuition of the TDMPC method is that all successful subtask executions from $\mathcal{T}^1$ are also successful subtask executions for $\mathcal{T}^2$, as this definition only depends on properties (2.13) of the subtask itself, not the subtask sequence. We need to determine which subtask executions are also part of task executions for $\mathcal{T}^2$, i.e. controllable to $\mathcal{P} = \mathcal{R}_{l_M}$.

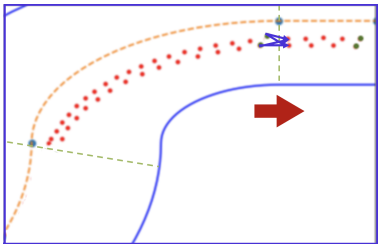
Only stored points at subtask transitions need to be analyzed. Based on this intuition, Algorithm 1 proceeds backwards through the new subtask sequence $[l_1, l_2, ..., l_M]$. The key steps are discussed below, and visualized in Fig. 2.1.



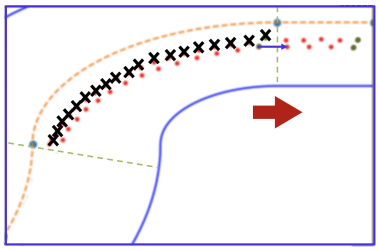
(a) Step 1: transfer stored subtask executions from $\mathcal{S}_{l_M}$.



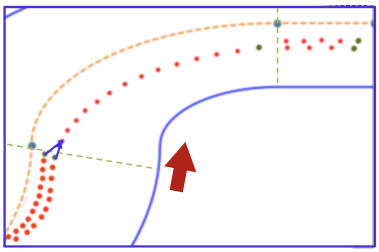
(b) Step 2: populate previous subtask with stored trajectories.



(c) Step 3: check controllability from points at subtask transition.



(d) Step 4: Remove subtask executions that cannot be controlled to the next task.



(e) Step 5: Repeat Steps 2-4 for subsequent subtasks.

Figure 2.1: Depiction of the TDMPC Alg. 1 applied to stored task data from an autonomous racing application. More details can be found in Sec. 2.5.

- Consider the last subtask of $\mathcal{T}^2$, $\mathcal{S}_{l_M}$. All states from $\mathcal{S}_{l_M}$ stored in the $\mathcal{T}^1$ executions are controllable to $\mathcal{R}_{l_M}$ using stored inputs, i.e. there exists a stored input sequence that can be applied to the state such that the system evolves to be in $\mathcal{P} = \mathcal{R}_{l_M}$. Thus, all states from $\mathcal{S}_{l_M}$ in $\mathcal{T}^1$ are also safe for $\mathcal{T}^2$. We next look for stored states from the preceding subtask, $\mathcal{S}_{l_{M-1}}$, which are controllable to $\mathcal{R}_{l_M}$ via $\mathcal{S}_{l_M}$. (Alg. 1, Lines 4-6, Fig. 2.1a-2.1b)

- Define the sampled guard set of $\mathcal{S}_{l_{M-1}}$ as

$$\mathcal{SG}_{l_{M-1}} = \left\{ \bigcup_{j=1}^{J} x^j_{D^j_{[1 \to l_{M-1}]}} \right\}. \tag{2.16}$$

  The set contains those states in $\mathcal{S}_{l_{M-1}}$ from which the system transitioned into another subtask during one of the previous J executions of $\mathcal{T}^1$. Only controllability from the sampled guard set will be important in our approach.

Algorithm 1 TDMPC

1: **input** f, $\mathcal{X}, \mathcal{U}$, $\mathcal{SS}^J_{[1\to M]}$, $\mathcal{SU}^J_{[1\to M]}, \mathcal{SQ}^J_{[1\to M]}$, $[l_1, l_2, ..., l_M]$ (2.6-2.7)
2: **output:** $\mathcal{SS}^0_{[l_1\to l_M]}$, $\mathcal{SU}^0_{[l_1\to l_M]}$, $\mathcal{SQ}^0_{[l_1\to l_M]}$
3:
4: **do** guard set clustering($\mathcal{SS}^J_{[1\to M]}$) (2.16)
5: **initialize empty** $\hat{\mathcal{SS}}, \hat{\mathcal{SU}}$
6: $\hat{\mathcal{SS}}_{l_M} \leftarrow \bigcup_{j=1}^J \boldsymbol{x}^j_{l_M}$
7: $\hat{\boldsymbol{u}}^j_{l_M} \leftarrow \boldsymbol{u}^j_{l_M} \forall j \in [1, J]$, $\hat{\mathcal{SU}}_{l_M} \leftarrow \bigcup_{j=1}^J \hat{\boldsymbol{u}}^j_{l_M}$,
8: $\hat{\mathcal{SQ}}_{l_M} \leftarrow \bigcup_{j=1}^J V^j(\boldsymbol{x}^j_{l_M})$
9: **for** $i \in [l_{M-1} : -1 : l_1]$ **do**
10: $\hat{\mathcal{SS}}_i \leftarrow \bigcup_{j=1}^J \boldsymbol{x}^j_i$
11: $\hat{\boldsymbol{u}}^j_i \leftarrow \boldsymbol{u}^j_i \ \forall j \in [1, J]$
12: **for** $x \in \mathcal{SG}_i$ **do**
13: $k = \{k : x \in \boldsymbol{x}^k_i\}$
14: **initialize empty** $Q^\star$
15: **for** $j : \boldsymbol{x}^j_{i+1} \in \hat{\mathcal{SS}}_{i+1}$ **do**
16: $\hat{\boldsymbol{q}}^j_{i+1} = V^j(\boldsymbol{x}^j_{i+1})$
17: **solve** $(Q^\star_j, u^\star_j) = \text{Ctrb}(x, \boldsymbol{x}^j_{i+1}, \hat{\boldsymbol{q}}^j_{i+1})$ (2.17)
18: **if** $Q^\star$ not empty **then**
19: $j_* = \arg\min_j Q^\star_j$
20: $\hat{\boldsymbol{u}}^k_i[-1] \leftarrow u^\star_{j_*}$
21: **else**
22: $\hat{\mathcal{SS}}_i \leftarrow \hat{\mathcal{SS}}_i \backslash \boldsymbol{x}^k_i$, $\hat{\mathcal{SU}}_i \leftarrow \hat{\mathcal{SU}}_i \backslash \hat{\boldsymbol{u}}^k_i$
23: $\hat{\mathcal{SU}}_i \leftarrow \bigcup_{j=1}^J \hat{\boldsymbol{u}}^j_i$, $\hat{\mathcal{SQ}}_i \leftarrow \bigcup_{j=1}^J V^j(\hat{\boldsymbol{x}}^j_i)$
24:
25: **return** $\mathcal{SS}^0_{[l_1\to l_M]} = \bigcup_{i=l_1}^{l_M} \hat{\mathcal{SS}}_i$, $\mathcal{SU}^0_{[l_1\to l_M]} = \bigcup_{i=l_1}^{l_M} \hat{\mathcal{SU}}_i$, $\mathcal{SQ}^0_{[l_1\to l_M]} = \bigcup_{i=l_1}^{l_M} \hat{\mathcal{SQ}}_i$

- We search for all points in $\mathcal{SG}_{l_{M-1}}$ that are controllable to stored states in $\mathcal{S}_{l_M}$, i.e. points which are in the transition sets of both $\mathcal{T}^1$ and $\mathcal{T}^2$. This problem can be solved using a variety of numerical approaches, including a one-step controllability problem (see Implementation). (Alg. 1, Lines 9-15, Fig. 2.1c)

- For any stored state x in $\mathcal{SG}_{l_{M-1}}$ for which the controllability analysis fails, we remove the stored $\mathcal{SS}^J_{[l_{M-1}]}$ subtask execution ending in x as candidate controllable states for $\mathcal{T}^2$. All remaining stored states in $\mathcal{S}_{l_{M-1}}$ are controllable to stored states in $\mathcal{S}_{l_M}$, and therefore also to $\mathcal{P} = \mathcal{R}_{l_M}$. (Alg. 1, Lines 15-20, Fig. 2.1d)

Algorithm 1 iterates backwards through the remaining subtask sequence (Fig. 2.1e), connecting points in sampled guard sets to previously verified trajectories in the next subtask.

The algorithm terminates when it has iterated through the new subtask order, or when no states in a subtask's sampled guard set can be shown to be controllable to $\mathcal{R}_{l_M}$. The algorithm returns sampled safe sets for $\mathcal{T}^2$ that have been verified through controllability to contain feasible executions of $\mathcal{T}^2$.

Implementation

TDMPC can improve on the computational complexity of existing trajectory transfer methods in two key ways: *(i)* by verifying stored trajectories only at states in the sampled guard set, rather than at each recorded time step, and *(ii)* by solving a data-driven, one-step controllability problem to adapt the trajectories, rather than a multi-step or set-based controllability method.

In the examples presented in this chapter, we implement the search for controllable points (Alg. 1, Line 15) by solving a one-step controllability problem, $(Q^\star, u^\star) = \text{Ctrb}(x, \boldsymbol{z}, \boldsymbol{q})$, where

$$u^\star, \lambda^\star = \underset{u,\lambda}{\arg\min}\ p(x,u) + \lambda^\top \boldsymbol{q} \tag{2.17}$$
$$\text{s.t.}\quad f(x,u) = \lambda^\top \boldsymbol{z}$$
$$\sum \lambda_i = 1,\ \lambda_i \geq 0$$
$$u \in \mathcal{U}_i,$$
$$Q^\star = \lambda^{\star\top} \boldsymbol{q}, \tag{2.18}$$

where $\mathbf{z}$ is a previously verified state trajectory through the next $\mathcal{T}^2$ subtask, and $\mathbf{q}$ the sampled cost vector from $\mathcal{SQ}^J$ (2.7a) associated with the trajectory. (2.17) aims to find an input that connects the sampled guard state x to a state in the convex hull of the trajectory [19, Sec 4.4.2]. If such an input is found, the new cost-to-go (2.8) for the state x is taken to be the convex combination of the stored cost vector (2.18). Solving the controllability analysis to the convex hull is an additional method for reducing computational complexity of TDMPC but is exact only for linear systems with convex constraints. This idea is explored further in Ch. 3.

Note that the number of subtasks and points of subtask transition (i.e. the sets $\mathcal{R}_i$) should be defined as is most useful, given the two tasks. Subtask transition points simply indicate which segments of the stored trajectories are certain to remain feasible in $\mathcal{T}^2$ using the stored policies—but this can change depending on how exactly $\mathcal{T}^2$ differs from $\mathcal{T}^1$. The TDMPC method is therefore not limited in applicability to a predetermined number of reshuffled tasks.

2.4 Properties of TDMPC Policies

We prove feasibility and iteration cost reduction of ILMPC policies (2.10) initialized using TDMPC.

Assumption 2. *$\mathcal{T}^1$ and $\mathcal{T}^2$ are defined as in (3.12), where the subtask workspaces and input spaces are given by $\mathcal{X}_i = \mathcal{X}$, $\mathcal{U}_i = \mathcal{U}$ for all $i \in [1, M]$.*

Theorem 1. *(Feasibility) Let Assumptions 1-2 hold. Assume non-empty sets $\mathcal{SS}^J_{[1\to M]}$, $\mathcal{SU}^J_{[1\to M]}$, $\mathcal{SQ}^J_{[1\to M]}$ containing trajectories of system (4.1) for $\mathcal{T}^1$. Assume Algorithm 1 outputs non-empty sets $\mathcal{SS}^0_{[l_1\to l_M]}$, $\mathcal{SU}^0_{[l_1\to l_M]}$, $\mathcal{SQ}^0_{[l_1\to l_M]}$ for $\mathcal{T}^2$. Then, if $x_0 \in \mathcal{SS}^0_{[l_1\to l_M]}$, the policy $\pi^{\text{ILMPC}}_{[l_1\to l_M]}$, as defined in (2.10), produces a feasible execution of $\mathcal{T}^2$.*

The proof is detailed in Sec. A.1. Theorem 1 implies that the safe sets designed by the TDMPC algorithm induce an ILMPC policy that can be used to successfully complete $\mathcal{T}^2$ while satisfying all input and state constraints.

Assumption 3. *Consider $\mathcal{T}^1$ and $\mathcal{T}^2$ as defined in (3.12). The trajectories stored in $\mathcal{SS}^J_{[1\to M]}$ and $\mathcal{SU}^J_{[1\to M]}$ correspond to executions of $\mathcal{T}^1$ by a nonlinear system (4.1). One stored trajectory corresponds to an execution of (4.1) in closed-loop with a policy $\pi^0(\cdot)$ that is feasible for both $\mathcal{T}^1$ and $\mathcal{T}^2$.*

An example of a control policy $\pi^0(\cdot)$ feasible for two different tasks for the autonomous racing task is a center-lane following controller moving at very slow longitudinal speed. Note that if asm. 3 does not hold for a particular set of tasks $\mathcal{T}^1$ and $\mathcal{T}^2$, it is possible that Alg. 1 fails to find a feasible connection between trajectories of two subtasks. In this case, the TDMPC strategy can be adapted as follows:

1. The controllability problem (2.17) may be transformed from a one-step to an N-step problem, where N can be adjusted until a connection is found.

2. A planning from scratch method can be used to search for a $\mathcal{T}^2$ policy up until the failed subtask connection, after which an ILMPC-based policy (2.10) based on the partial TDMPC safe set can be used for the remainder of the task.

Theorem 2. *(Cost Improvement) Let Assumptions 2-3 hold. Then, Algorithm 1 will return non-empty sets $\mathcal{SS}^0_{[l_1\to l_M]}$, $\mathcal{SU}^0_{[l_1\to l_M]}$, $\mathcal{SQ}^0_{[l_1\to l_M]}$ for $\mathcal{T}^2$. Furthermore, if $x_0 \in \mathcal{SS}^1_{[1\to M]}$, an ILMPC initialized using $\mathcal{SS}^0_{[l_1\to l_M]}$ will incur no higher iteration cost during an execution of $\mathcal{T}^2$ than an ILMPC initialized using a trajectory corresponding to (4.1) in closed-loop with $\pi^0(\cdot)$.*

The proof is found in Sec. A.1. The main idea is that the ILMPC policy will at each time step choose a terminal state with the lowest cost-to-go. If the new safe set $\mathcal{SS}^0_{[l_1\to l_M]}$ contains the trajectory induced by $\pi^0(\cdot)$, the ILMPC will at each time step either match or outperform $\pi^0(\cdot)$.

The proof that the improved closed-loop iteration cost follows from the improved ILMPC cost (2.9) is straightforward and not included here. However, the result shown holds for the examples presented in Sec. 2.5-2.6.

2.5 Application 1: Autonomous Racing

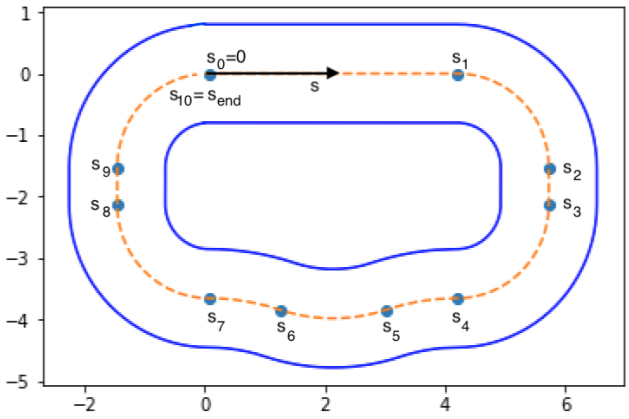


Figure 2.2: Each subtask of the racing task corresponds to a segment of the track with constant curvature. The vehicle state s tracks the distance traveled along the centerline.

Task Formulation

Consider an autonomous racing task, in which a vehicle is controlled to minimize lap time driving around a race track with piecewise constant curvature (Fig. 2.2), while satisfying all system and environmental state and input constraints. We model this task as a series of ten subtasks, where the i-th subtask corresponds to a section of the track with constant radius of curvature c_i. Tasks with different subtask order are tracks consisting of the same road segments in a different order. Each new control task $\mathcal{T}$ corresponds to a new track, described using the environment descriptor function $\mathbf{\Theta}$ which maps the current position along the track to a description of the local track curvature.

The vehicle is modeled as a Euler discretized dynamic bicycle model [101] in the curvilinear abscissa reference frame [94], with states and inputs at time step k given by

$$x_k = [v_{x_k}\ v_{y_k}\ \dot{\psi}_k\ e_{\psi_k}\ s_k\ e_{y_k}]^\top$$
$$u_k = [a_k\ \delta_k]^\top,$$

where v_{x_k}, v_{y_k}, and $\dot{\psi}_k$ are the vehicle's longitudinal velocity, lateral velocity, and yaw rate, respectively, at time step k, s_k is the distance travelled along the centerline of the road, and e_{ψ_k} and e_{y_k} are the heading angle and lateral distance error between the vehicle and the path. The inputs are longitudinal acceleration a_k and steering angle δ_k.

The vehicle is subject to system-imposed state and input constraints given by

$$\mathcal{X} = \left\{ x : \begin{bmatrix} 0 \\ -10 \text{ m/s} \\ -\frac{\pi}{2} \text{ rad} \\ -\frac{\pi}{3} \text{ rad} \\ 0 \\ -\frac{l}{2} \text{ m} \end{bmatrix} \leq \begin{bmatrix} v_x \\ v_y \\ w_z \\ e_\psi \\ s \\ e_y \end{bmatrix} \leq \begin{bmatrix} 10 \text{ m/s} \\ 10 \text{ m/s} \\ \frac{\pi}{2} \text{ rad} \\ \frac{\pi}{3} \text{ rad} \\ L \text{ m} \\ \frac{l}{2} \text{ m} \end{bmatrix} \right\} \tag{2.20}$$

$$\mathcal{U} = \left\{ u : \begin{bmatrix} -1 \text{ m/s}^2 \\ -0.5 \text{ rad/s}^2 \end{bmatrix} \leq \begin{bmatrix} a \\ \delta \end{bmatrix} \leq \begin{bmatrix} 1 \text{ m/s}^2 \\ 0.5 \text{ rad/s}^2 \end{bmatrix} \right\},$$

where $l = 0.8$ is the track's lane width and $L = 19.2$ the track length, both in meters. These bounds indicate that the vehicle can only drive forwards on the track, up to a maximum velocity, and must stay within the lane.

The task target set is the race track's finish line,

$$\mathcal{P} = \mathcal{R}_M = \{x : s \geq s_{\text{end}}\}.$$

The task goal is to complete a lap and reach the target set $\mathcal{P}$ as quickly as possible. Therefore we define the stage cost in (2.9) as:

$$p(x_k, u_k) = \begin{cases} 0, & x_k \in \mathcal{P} \\ 1, & \text{otherwise.} \end{cases}$$

We formulate each subtask according to (2.11), with:

Subtask Workspace $\mathcal{X}_i$

$$\mathcal{X}_i = \{x \in \mathcal{X} : s_{i-1} \leq s \leq s_i\},$$

where s_{i-1} and s_i mark the distances along the centerline to the start and end of the subtask. This means $s_{10} = s_{\text{end}}$ is the total length of the track.

Subtask Input Space $\mathcal{U}_i$

$$\mathcal{U}_i = \mathcal{U}$$

The input limits are a function of the vehicle, not the environment, and do not change between subtasks.

Subtask Transition Set, $\mathcal{R}_i$

We define the subtask transition set to be the states along the subtask border where the track's radius of curvature changes:

$$\mathcal{R}_i = \{x \in \mathcal{X}_i : \exists u \in \mathcal{U}_i, \text{ s.t. } f(x, u) \in \mathcal{X}_{i+1}\}.$$

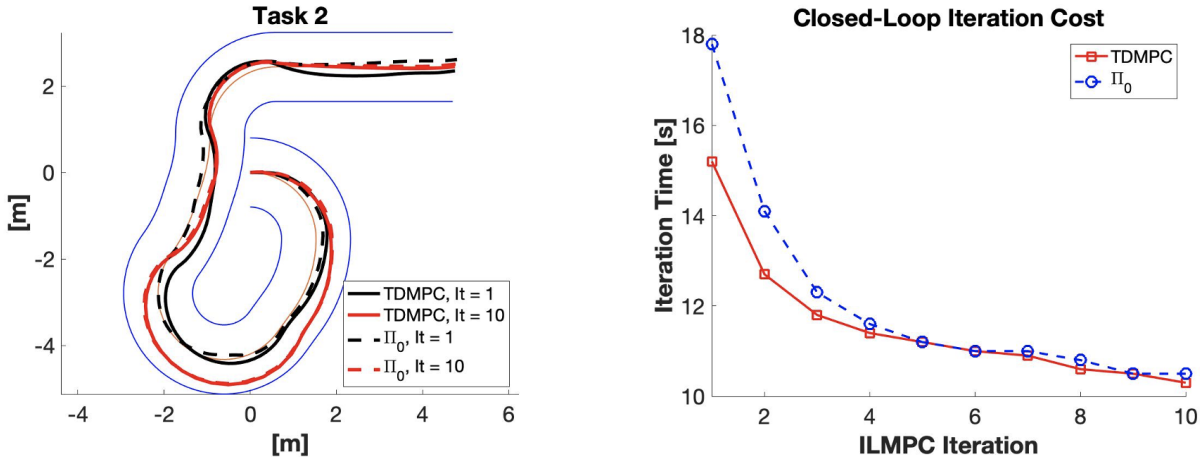


Figure 2.3: The TDMPC-initialized ILMPC converges to a locally optimal trajectory faster than the PID-initialized one.

Subtask Environment Descriptor Function, $\mathbf{\Theta}_i$

The task's environment descriptor function $\mathbf{\Theta}$ maps the current position along the track to a description of the upcoming track curvature. Since the curvature is assumed constant in each subtask, $\mathbf{\Theta}_i$ maps any $x \in \mathcal{X}_i$ to c_i.

Simulation Setup

An ILMPC (2.10) is used to complete $J = 5$ executions of $\mathcal{T}^1$, the track depicted in Fig. 2.2. The vehicle begins each task iteration at standstill on the centerline at the start of the track. The J executions and their costs are stored in $\mathcal{SS}_{[1\to M]}$, $\mathcal{SU}_{[1\to M]}$, and $\mathcal{SQ}_{[1\to M]}$. An initial trajectory for the ILMPC safe sets is executed using a centerline-tracking, low-velocity PID controller, π_0.

TDMPC then uses these sampled safe sets to design initial policies for a new track composed of the same track segments. Two ILMPCs are designed for the reconfigured track: one initialized with TDMPC, and another initialized with π_0. Each ILMPC completes $J = 10$ laps around the new tracks. In this example, the reconfigured track is not continuous, and should be considered to be a segments of larger, continuous track.

Simulation Results

Fig. 2.3 compares the first and tenth trajectories around the track of the two ILMPCs, plotted as black and red lines. The π_0-initialized ILMPC (in dashed black) initially stays close to the centerline, taking nearly 18 seconds to traverse the new track. The TDMPC-

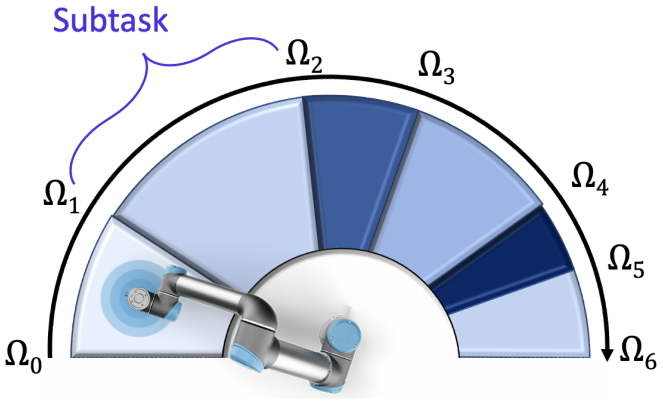


Figure 2.4: Topview of the robotic path planning task. Each subtask corresponds to an obstacle in the environment with constant height.

initialized ILMPC, however, traverses the new track more efficiently starting with the first lap. The first lap completed using the TDMPC-initialized ILMPC (in solid black) begins closer to the final locally optimal policies (in red) that both ILMPCs eventually converge to. Here, the TDMPC method is able to leverage experience on another track in order to complete sections of the new track in a locally optimal way, even on the first iteration of a new task.

The lap time of each of the ten ILMPC iterations is plotted in the bottom of Fig. 2.3. As expected, the TDMPC-initialized ILMPC completes the first several laps faster than the π_0-initialized ILMPC. The TDMPC-initialized ILMPC requires fewer task iterations and less time per iteration to reach a locally optimal trajectory.

2.6 Application 2: Robotic Path Planning

TDMPC can also be used to combine knowledge gained from solving a variety of previous tasks. For example, if n ILMPCs as in (2.10) complete J iterations of n different tasks, all composed of the same subtasks, TDMPC can be used to design a policy for a task $\mathcal{T}^{n+1}$. The algorithm draws on subtask executions collected over n different tasks in order to build safe sets for $\mathcal{T}^{n+1}$. We evaluate this approach in a robotic path planning example.

Task Formulation

Consider a task in which a UR5e[1] robotic arm needs to move an object to a target without colliding with obstacles (Fig. 2.4). The obstacles are modeled as extruded disks of varying heights above and below the robot, leaving a workspace space between $o_{\min,i}$ and $o_{\max,i}$. Here,

[1]https://www.universal-robots.com/products/ur5-robot/

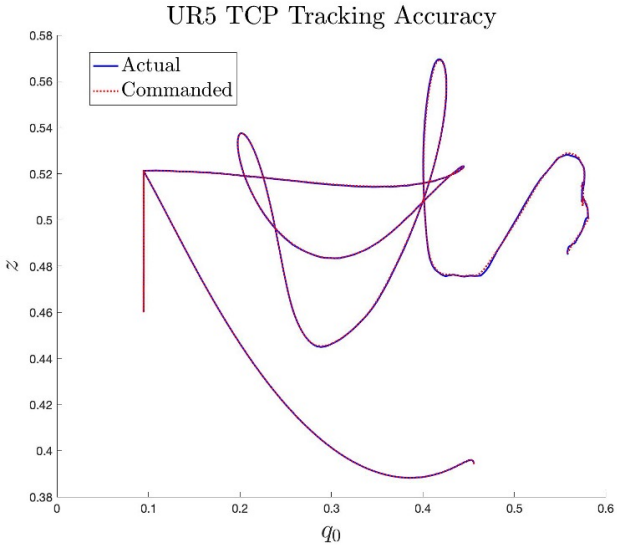


Figure 2.5: The UR5e manipulator has very high tracking accuracy, allowing us to model the end effector as an integrator system in place of a more complex dynamic model.

each subtask corresponds to the workspace above a particular obstacle. Different subtask orderings correspond to a rearranging of the obstacle locations. As in the autonomous racing example, here the environment descriptor function $\boldsymbol{\Theta}$ maps an end-effector position to the corresponding workspace heights at that location.

In a certain subset of the state and input space, characterized experimentally, the UR5e has very high end-effector reference tracking accuracy (see Fig. 2.5). This allows us to use a simplified end-effector model in place of a discretized second-order model as in [110]. We solve the task in the reduced state space:

$$x_k = [q_{0_k} \ \dot{q}_{0_k} \ z_k \ \dot{z}_k]^\top$$
$$u_k = [\ddot{q}_{0_k} \ \ddot{z}_k]^\top,$$

where q_{0_k} is the angle of the robot's base joint and z_k is the height of the robot end-effector at time step k, calculated from the six joint angles via forward kinematics. $\dot{q}_{0_k}$ and $\dot{z}_k$ are the corresponding velocities. We control $\ddot{q}_{0_k}$ and $\ddot{z}_k$, the accelerations of q_0 and z, respectively. We model the simplified system as a quadruple integrator:

$$x_{k+1} = \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 & 0 \\ dt & 0 \\ 0 & 0 \\ 0 & dt \end{bmatrix} u_k, \tag{2.22}$$

where $dt = 0.01$ seconds is the sampling time.

The system state and input constraints are

$$\mathcal{X} = \{x : -\pi \text{ rad/s} \leq \dot{q}_{0_k} \leq \pi \text{ rad/s}\}$$
$$\mathcal{U} = \left\{u : -\pi \text{ rad/s}^2 \leq \ddot{q}_{0_k} \leq \pi \text{ rad/s}^2\right\}.$$

The task target set is the end of the obstacle course,

$$\mathcal{P} = \mathcal{R}_M = \{x : q_0 = \Omega_M,\ o_{\min,M} \leq z \leq o_{\max,M}\},$$

and the task goal is to reach the target set as quickly as possible:

$$p(x_k, u_k) = \begin{cases} 0, & x_k \in \mathcal{P} \\ 1, & \text{otherwise.} \end{cases}$$

We formulate each subtask according to (2.11).

Subtask Constraints $\mathcal{X}_i$

$$\mathcal{X}_i = \left\{ x \in \mathcal{X} : \begin{bmatrix} \Omega_{i-1} \text{ rad} \\ o_{\min,i} \text{ m} \\ \dot{z}_{\min,i} \text{ m/s} \end{bmatrix} \leq \begin{bmatrix} q_0 \\ z_k \\ \dot{z}_k \end{bmatrix} \leq \begin{bmatrix} \Omega_i \text{ rad} \\ o_{\max,i} \text{ m} \\ \dot{z}_{\max,i} \text{ m/s} \end{bmatrix} \right\}$$

where Ω_{i-1} and Ω_i mark the cumulative angle to the beginning and end of the i-th obstacle, as in Fig. 3.4. The robot end-effector is constrained to remain in the space between the upper and lower obstacles, bounded by $o_{\min,i}$ and $o_{\max,i}$. The base's rotational velocity $\dot{q}_{0k}$ and $\dot{z}_k$ are constrained to lie in the experimentally determined region of high end-effector tracking accuracy. Specifically, we take

$$\dot{z}_{\max,i} = C_1 \sin\left(\arccos\left(\frac{o_{\min,i}}{d_1}\right)\right)$$
$$\dot{z}_{\min,i} = -\dot{z}_{\max,i},$$

where the constants C_1 and d_1 depend on setup parameters and joint limits provided by the manufacturer.

Subtask Input Space $\mathcal{U}_i$

$$\mathcal{U}_i = \left\{u \in \mathcal{U}_i \ :\ \ddot{z}_{\min,i} \text{ m/s}^2 \leq \ddot{z}_k \leq \ddot{z}_{\max,i} \text{ m/s}^2\right\},$$

where $\ddot{q}_{0k}$ and $\ddot{z}_k$ are constrained to lie in the experimentally determined region of high end-effector tracking accuracy. Specifically,

$$\ddot{z}_{\max,i} = C_2 \sin\left(\arccos\left(\frac{o_{\min,i}}{d_2}\right) + \frac{o_{\min,i}}{d_3}\right)$$
$$\ddot{z}_{\min,i} = -\ddot{z}_{\max,i},$$

where C_2, d_2 and d_3 depend on setup parameters and joint limits provided by the manufacturer.

Subtask Transition Set, $\mathcal{R}_i$

We define the subtask transition set to be the states along the subtask border where the next obstacle begins:

$$\mathcal{R}_i = \{x \in \mathcal{X}_i : \exists u \in \mathcal{U}_i, \text{ s.t. } q_0^+ \geq \Omega_i \},$$

where $x^+ = A_i x + B_i u$ (3.1). The task target set is the end of the last mode:

$$\mathcal{R}_6 = \{x : q_0 = \Omega_6,\ o_{\min,6} \leq z \leq o_{\max,6}\}.$$

The task goal is to reach the target set as quickly as possible:

$$p(x_k, u_k) = \begin{cases} 0, & x_k \in \mathcal{R}_6 \\ 1, & \text{otherwise.} \end{cases}$$

Experimental Setup

An ILMPC (2.10) was used to complete $J = 10$ executions of five different training tasks, where each training task corresponded to a reordering of the obstacles. In each task, the ILMPC tries to reach the target set as quickly as possible while avoiding the obstacles. Each ILMPC was initialized with a trajectory resulting from executing a policy π_0 that tracks the center height of each mode with the end-effector, while the robot rotated at a low constant joint velocity $\dot{q}_0$. TDMPC was then applied to the combined sampled safe sets of the five training tasks, and used to design an initial policy for a new ILMPC on an unseen ordering of obstacles, shown in Fig. 2.6. The white space corresponds to environment obstacles, so that the ILMPC task is to reach the end of the last mode as quickly as possible while controlling the end-effector to remain within the safe (green) part of the state space. A second ILMPC was initialized with the center-height tracking π_0, for comparison. After initialization, the two ILCMPs completed $J = 20$ iterations of the new task. These iterations were executed in simulation using the simplified model (5.29), and the first and last trajectories of each ILMPC were then tracked by a real UR5e robot using end-effector tracking.

Experimental Results

The measured robot trajectories are plotted in Fig. 2.6. The π_0-initialized ILMPC follows the center-height of each mode closely during the first task iteration (plotted in dashed black). After ten iterations of the task, the resulting trajectory (plotted in dashed red) has only diverged from the center-height trajectory slightly. Correspondingly, after ten iterations the π_0-initialized ILMPC still requires more than four seconds to complete the task.

The TDMPC-initialized ILMPC, however, draws on knowledge gathered over many previous tasks in order to solve the task efficiently right away. Already on the first trajectory (plotted in solid black), the TDMPC-initialized ILMPC solves the task in under three seconds. This is a 30% improvement over the π_0-initialized ILMPC. As in the autonomous

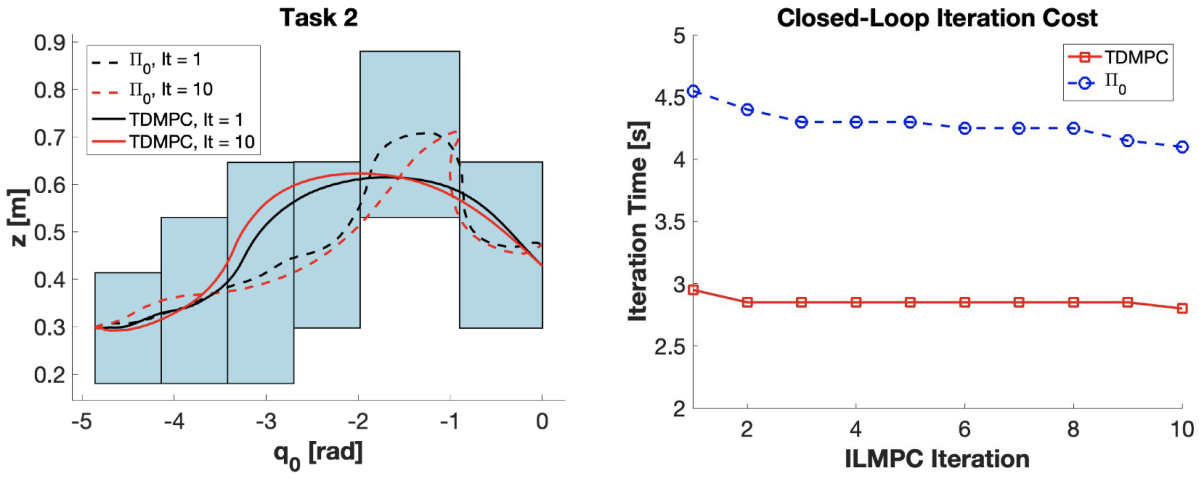


Figure 2.6: The TDMPC-initialized ILMPC solves $\mathcal{T}^2$ much faster than the ILMPC initialized with a center-height tracking policy π_0.

driving task, the first trajectory completed by the TDMPC-initialized ILMPC is very close to the ultimate locally optimal trajectory.

Because of the nonconvex obstacles, this task is nonconvex, and there are many locally optimal trajectories. At various iterations of the task, both the TDMPC-initialized and the π_0-initialized ILMPCs get stuck at such local minima, so that the ILMPCs performance metric remained constant over several iterations before improving again (Fig. 2.6). At these performance plateaus, the realized trajectories continue to change. We believe that the variability in the mixed integer solver used in the ILMPC led the ILMPC to follow different trajectories with the same iteration cost, as if encouraging exploration. Some of these different trajectories then allowed for performance improvement in the next iteration.

2.7 Discussion

The TDMPC algorithm provides a systematic way of adapting stored task trajectories to the changed environmental constraints of a new task. Here we analyze two perspectives of TDMPC.

A Hybrid Systems Perspective

The TDMPC algorithm performs backwards reachability between points in different subtasks. If each subtask is viewed as a mode of operation, TDMPC can be analyzed from a hybrid systems reachability perspective.

Hybrid systems refer to a class of dynamical systems that switch among several discrete operating modes, with each mode governed by its own dynamics [18]. Hybrid systems reachability considers whether a feasible trajectory exists between a set of initial states and a set of goal states in a potentially different mode. The extensive literature on hybrid systems reachability mainly focuses on two approaches: set-based methods and simulation [107]. Set-based methods are exhaustive methods that use reach set computation to verify feasibility of entire sets of initial conditions and bounded inputs, and many algorithms have recently been proposed [62, 96, 106, 75]. Though effective, set-based methods suffer from the "curse of dimensionality", and do not scale well with state dimension. These methods often approximate complex sets as polyhedral or ellipsoidal, which affects solution accuracy. To combat this, the authors of [7] propose splitting the system state into independent substates, but this is not guaranteed to work for complex systems.

Sampling-based simulation methods verify feasibility of a trajectory from an initial condition under sampled input sequences. These methods are less limited to low-dimensional systems, but are not an exhaustive search and can miss subtle phenomena that complex dynamics may generate [115, 36, 6, 70].

In contrast, TDMPC only solves reachability problems between discrete points in the sampled guard set. This ensures the algorithm scales well with state dimension and number of subtasks without requiring drastic approximations. Even if set-based methods are computationally feasible for a particular system, in contrast to TDMPC they only provide sets of reachable states, without a complete policy. Additionally, traditional sampling-based hybrid systems reachability methods propagate sampled trajectories with random inputs, without any check on whether the trajectory is promising, or will inevitably lead to eventual infeasibility. TDMPC explicitly only checks for reachability to feasible points. Lastly, TDMPC views the transitions between subtasks as particular to the task instance, rather than permanent. We are aware of no previously published work in which the transitions of a hybrid system change.

A Dynamic Programming Perspective

Dynamic Programming (DP) methods provide exact solutions to constrained optimal control problems. However, DP can incur tremendous cost and is therefore not implementable for high-dimensional systems. Recent work [12] proposes forming aggregate (or representative) features out of system states in order to reduce the problem dimension. These reduced-dimension problems can provide approximate solutions to the original task. In spatio-temporal aggregation, coarse space and time states are chosen as aggregate features. Space-time barriers serve as transition sets between these aggregate features, and the shortest path problem is solved only between points immediately adjacent to the barriers. This is an analog to TDMPC performing reachability only at points in the sampled guard sets. While, unlike TDMPC, spatio-temporal aggregation does not explicitly consider a notion of reordering, it provides an additional perspective on the utility of task segmentation for computationally effective policy instantiating.

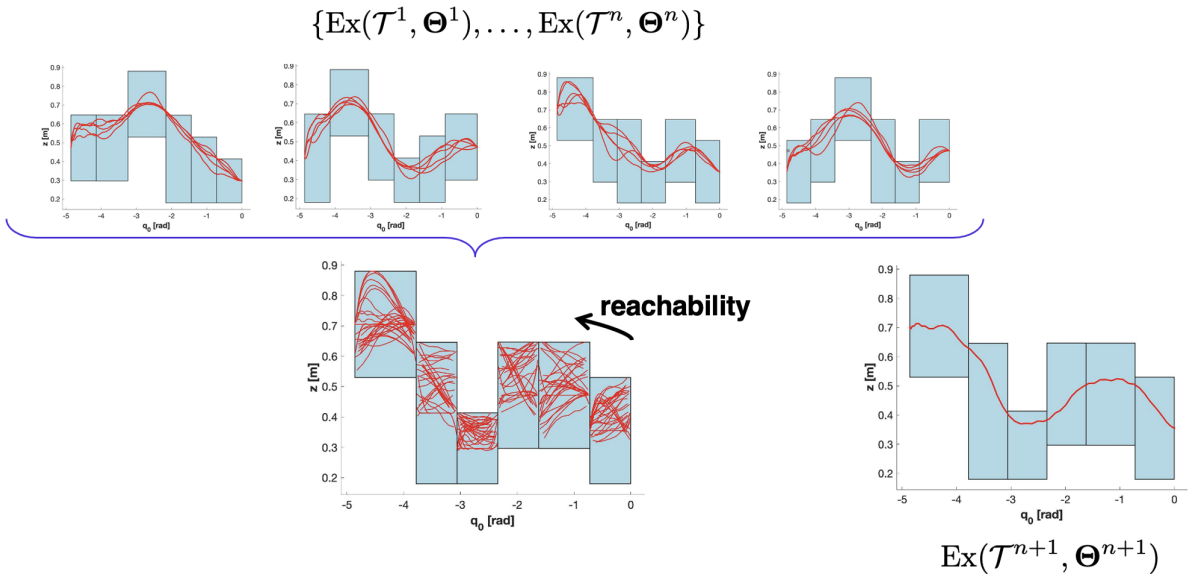


Figure 2.7: An overview of the TDMPC approach for using stored data to efficiently find an initial trajectory for a new ILMPC.

A Controllability Perspective

Something about quickly constructing controllable sets by searching in the span of the data? Here we can discuss that this is what's happening in ILMPC (we use controllable sets), and also generally for safe MPC. I'm not sure where to put this.

2.8 Conclusion

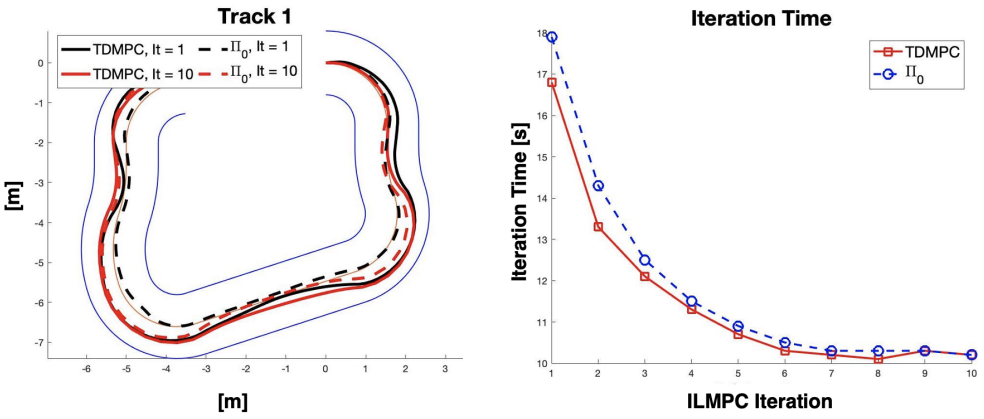
In this chapter, we introduced the first approach for using stored state and input trajectories from executions of a task to efficiently designs policies for executing variations of that task. We began by considering a simplification of the changing environment problem, and assuming that different tasks could be split into a shared set of subtasks. The proposed TDMPC algorithm takes inspiration from ILMPC, an iterative learning control method that uses stored trajectory data to design safe terminal sets for an MPC controller that result in performance improvement with each task iteration. TDMPC breaks the stored task trajectories into subtasks and performs controllability analysis at sampled safe states between subtasks. This one-step controllability analysis allows us to quickly learn whether any previously recorded subtask executions (which were controllable to the previous task target set) are also controllable to the new task's target set. These verified trajectories can then be used to construct an ILMPC safe set, and used in an ILMPC controller for the new task.

The effectiveness of the proposed algorithm was evaluated on autonomous racing and

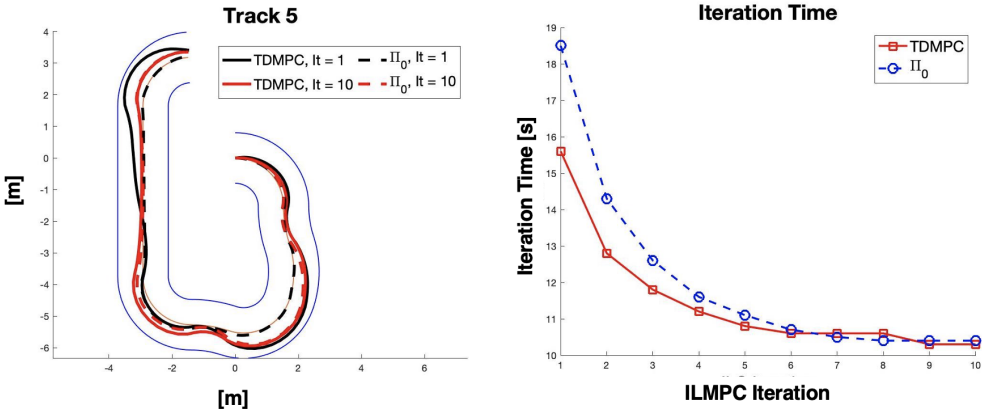
robotic manipulation tasks, both in simulation and experiments. Our results confirm that TDMPC allows an ILMPC to converge to a locally-optimal minimum-time trajectory faster than using simple initialization methods, by providing a smarter initial trajectory constructed from efficient previous subtask executions.

The TDMPC algorithm can improve upon other trajectory library methods by only needing to verify and adapt the original task policy at points of subtask transition, rather than along the entire trajectory, and it provides a fast and straightforward method for finding approximations of controllable sets in a new task. In Chapter 3, we increase this efficiency further by considering algorithm adaptations for piecewise linear systems with piecewise convex constraints.

2.9 Additional Results



(a)



(b)

Figure 2.8: Additional examples from the autonomous racing application.

Chapter 3

Task Decomposition for Piecewise Linear Systems

3.1 Introduction

Classical Iterative Learning Controllers (ILCs) aim to improve a system's closed-loop reference tracking performance at each iteration of a repeated task ([20, 69]). In both classical and reference-free ILC, the controller uses data from previous iterations to improve future closed-loop performance with respect to the appropriate performance metric. At the very first iteration, these methods require either a reference trajectory to track or a feasible trajectory with which to initialize the control algorithm. If the task changes, a new trajectory must be designed, which can be difficult for complex tasks.

In this chapter, we again consider the problem of finding a feasible trajectory to smartly initialize an Iterative Learning Model Predictive Controller (ILMPC) ([98]) for a new task. ILMPC is a reference-free ILC that uses a safe set to design an MPC policy for an iterative control task. This safe set is initialized using a feasible task trajectory, and collects states from which the task can be completed. In Chapter 2, a Task Decomposition algorithm for ILMPC (TDMPC) was introduced for nonlinear, constrained dynamical systems. TDMPC is data-efficient, requires no human supervision, and, if the algorithm converges, produces trajectories that are guaranteed to satisfy all constraints for the new task. TDMPC decomposes an initial task $\mathcal{T}^1$ into different modes of operation, called subtasks, and adapts stored $\mathcal{T}^1$ trajectories to a new task $\mathcal{T}^2$ only at points of subtask transition, by solving one-step controllability problems. In this chapter, we:

1. present a reformulation of the TDMPC Alg. 1 in Chapter 2 for searching for a task trajectory in the space of previously stored subtask trajectories. We introduce a new formulation for piecewise linear systems with piecewise convex state and input constraints. The new formulation further reduces the computational burden of finding feasible trajectories for a new task $\mathcal{T}^2$ by formulating the controllability check as a

convex optimization problem, and simultaneously increases the size of the resulting $\mathcal{T}^2$ safe set.

2. prove that the induced safe set based MPC policy is feasible for $\mathcal{T}^2$. This policy can be used to initialize an iterative learning control algorithm, or to directly obtain a suboptimal execution of $\mathcal{T}^2$.

The results presented in this chapter have also appeared in:

- C. Vallon and F. Borrelli. "Task decomposition for MPC: A computationally efficient approach for linear time-varying systems." In: *2020 IFAC World Congress*. 2020, pp. 4240-4245.

3.2 Problem Formulation

As in Chapter 2, we consider a system (4.1) solving tasks $\mathcal{T}$ that can be decomposed into an ordered sequence of subtasks. However, here we restrict our focus to systems with piecewise linear dynamics and piecewise convex system and environmental constraint sets. Specifically, within the i-th subtask $\mathcal{S}_i$, the system is subject to linear dynamics

$$\forall x_k \in \mathcal{X}_i,\ x_{k+1} = A_i x_k + B_i u_k, \tag{3.1}$$

and convex system state and input workspaces

$$x_k \in \mathcal{X}_i,\ u_k \in \mathcal{U}_i, \tag{3.2}$$

where $\mathcal{X}_i \subseteq \mathcal{X}$ and $\mathcal{U}_i \subseteq \mathcal{U}$ are convex sets. $\mathcal{R}_i$ is the set of transition states from $\mathcal{S}_i$ into the next subtask $\mathcal{S}_{i+1}$:

$$\mathcal{R}_i \subseteq \mathcal{X}_i = \{x \in \mathcal{X}_i : \exists u \in \mathcal{U}_i,\ A_i x + B_i u \in \mathcal{X}_{i+1}\}. \tag{3.3}$$

Additionally, the environmental constraints in each subtask are also assumed to be convex:

Assumption 4. *The environmental state constraints in each subtask, $E(\boldsymbol{\Theta}_i(\cdot))$, are convex in each subtask.*

Note that the requirement of piecewise linearity of the system dynamics (3.1) and piecewise convexity of the constraints are key novelties compared to Chapter 2.

3.3 Safe Set Based ILMPC for Piecewise Linear Systems

In Chapter 2, an overview of safe set based ILMPC [100] for general nonlinear systems was given. We then presented the TDMPC algorithm, a method for using stored executions from

previous tasks in order to construct a trajectory for a new task. Such a trajectory can be used to initialize an ILMPC for the new task.

Here we want to take advantage of the additional linearity and convexity requirements on the subtask dynamics and constraints in order to improve the efficiency of TDMPC Alg. 1. In [98], an adaptation of the ILMPC for linear time-invariant systems with convex constraints was introduced. We begin by proposing an adaptation of this ILMPC formulation for piecewise linear systems with piecewise convex constraints.

As in standard ILMPC [100], we begin by sorting stored executions into a safe set. After J number of task iterations, we define the time-indexed sampled subtask safe set of subtask $\mathcal{S}_i$ as:

$$\mathcal{KS}_{i,k} = \left\{ \bigcup_{j=1}^{J} x^j_{D_i^j - k} \right\}, \; k \in [0, \max_j D_i^j - 1]. \tag{3.4}$$

For given k and i, $x^j_{D_i^j-k}$ is the k-to-last state visited in $\mathcal{S}_i$ during the j-th task iteration. Thus each set (3.4) is the collection of states from which the system reaches the subtask transition set $\mathcal{R}_i$ in exactly k steps during a previously recorded task iteration, while satisfying subtask constraints (3.2). Notice that these sets are more restrictive than $\mathcal{SS}^J$ (2.6). We similarly define a time-indexed sampled subtask input set of subtask $\mathcal{S}_i$ as:

$$\mathcal{KU}_{i,k} = \left\{ \bigcup_{j=1}^{J} u^j_{D_i^j - k} \right\}, \; k \in [0, \max_j D_i^j - 1].$$

Lastly, we define convex subtask safe sets and convex subtask input sets as:

$$\begin{aligned} \mathcal{CK}_{i,k} &= \left\{ \sum_{p=1}^{|\mathcal{KS}_{i,k}|} \lambda_p z_p : \lambda_p \geq 0, \; \sum_{p=1}^{|\mathcal{KS}_{i,k}|} \lambda_p = 1, z_p \in \mathcal{KS}_{i,k} \right\} \\ \mathcal{CU}_{i,k} &= \left\{ \sum_{p=1}^{|\mathcal{KU}_{i,k}|} \lambda_p w_p : \lambda_p \geq 0, \; \sum_{p=1}^{|\mathcal{KU}_{i,k}|} \lambda_p = 1, w_p \in \mathcal{KU}_{i,k} \right\}, \end{aligned} \tag{3.5}$$

where $|\mathcal{KS}_{i,k}|$ is the cardinality of $\mathcal{KS}_{i,k}$. Note that because $\mathcal{X}_i$, $\mathcal{U}_i$, and $E(\mathbf{\Theta}_i(\cdot))$ are all convex, there also exists a feasible k-step input sequence to $\mathcal{R}_i$ for each element in $\mathcal{CK}^J_{i,k}$ (see [98] or Thm. 3 for a proof). This fact will be key to adapting the TDMPC Alg. 1 for piecewise linear systems.

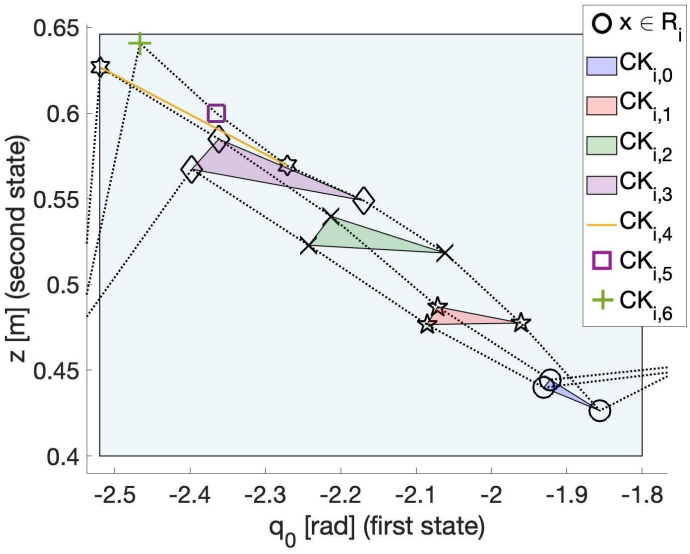


Figure 3.1: Convex subtask safe sets contain states from which the transition set can be reached in a certain number of steps. Data reproduced from robotic path-planning application (see Sec. 3.6).

Lastly, we define a barycentric cost-to-go over the convex subtask safe sets:

$$v(x) = \min_{\lambda_p \geq 0,\ I,\ K} \sum_{p=0}^{|\mathcal{KS}_{I,K}|} \lambda_p V(z_p) \tag{3.6}$$
$$\text{s.t.} \sum_{p=0}^{|\mathcal{KS}_{I,K}|} \lambda_p = 1$$
$$\sum_{p=0}^{|\mathcal{KS}_{I,K}|} \lambda_p z_p = x, \quad z_p \in \mathcal{KS}_{I,K},$$

where $V(z_p)$ was the realized cost-to-go (2.8) from state z_p during a past execution. This allows us to approximate the cost-to-go from new states in the time-indexed convex hull of stored points as the convex combination of nearby states visited in previous task iterations.

Fig. 3.1 depicts these sets for three trajectories through a subtask from a robotic path-planning task detailed in Sec. 3.6. Using these sets, we can approximate the optimal control

problem (2.4) at time step k of iteration $J+1$ by solving:

$$V^{\text{ILMPC},J+1}(x_k^{J+1}) = \min_{u_{k|k},\ldots,u_{k+N-1|k},I,K} \quad \sum_{t=k}^{k+N-1} p(x_{k+t|k}, u_{k+t|k}) + v(x_{k+N|k}) \tag{3.7}$$
$$\text{s.t.} \quad x_{k+t+1|k} = f(x_{k+t|k}, u_{k+t|k})$$
$$u_{k+t|k} \in \mathcal{U}_i,\ \forall t \in \{0,\ldots,N-1\}$$
$$x_{k+t|k} \in \mathcal{X}(\boldsymbol{\Theta}),\ \forall t \in \{0,\ldots,N\}$$
$$x_{k|k} = x_k^j$$
$$x_{k+N|k} \in \mathcal{CK}_{I,K} \cup \mathcal{P},$$

which searches for an input sequence over a horizon N that controls the system (3.1) to the state in a convex subtask safe state set or task target set $\mathcal{P}$ with the lowest cost-to-go (3.6). We use a receding horizon strategy:

$$u(x_k^j) = \pi^{\text{ILMPC}}(x_k^j) = u_{k|k}^\star. \tag{3.8}$$

In Sec. 3.5, we prove that a system (3.1) in closed-loop with (3.8) leads to a feasible task execution for $\mathcal{T}$.

At the first iteration of a new task, the ILMPC (3.7) requires non-empty sets $\mathcal{CK}_{\cdot,\cdot}$ containing at least one feasible execution of the task. In Sec. 3.4, we present a computationally efficient approach for creating such sets using data from executions of different tasks.

Controllability Properties

First, we state the following propositions, which motivate the approach of storing task executions in time-indexed convex subtask safe sets (3.5).

Proposition 1. *In general, not all states belonging to the convex hull of stored subtask executions are controllable to the subtask transition set (3.3).*

Proof. We know that all states in a feasible subtask execution (2.13) are controllable to a point in the subtask transition set (2.13b). A sufficient condition for all states in the convex hull of feasible subtask executions to also be controllable to the convex hull of corresponding points in the subtask transition set is for latter to be a control invariant set.

By definition of a feasible task execution (2.5) and the proof of Thm. 3, any point in the convex hull of stored states in the transition set is also controllable to the task's control invariant goal set. For linear time-invariant systems, states that are controllable to an invariant set are also invariant ([19]). However, for linear time-varying systems these states are only stabilizable to the invariant. Therefore the convex hull of the points in the transition set is not inherently an invariant.

We recognize that the convex hull of states in the transition set *is* a control invariant set if its backward controllable sets grow to contain each other, i.e. if for all scalar values

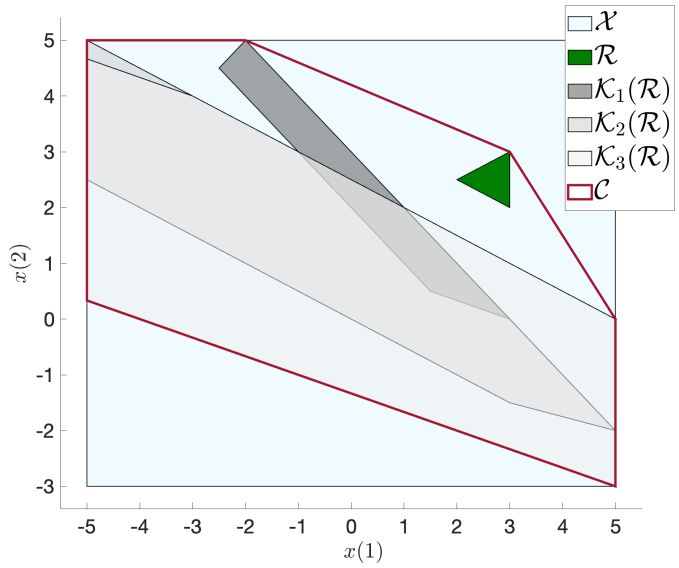


Figure 3.2: For the double integrator system (3.9 - 3.10), the chosen target set (3.11) is not an invariant set, as the N-step controllable sets are not subsets of the $(N+1)$-step controllable sets.

N, $\mathcal{K}_{N-1}(\mathcal{R}_i) \subseteq \mathcal{K}_N(\mathcal{R}_i)$. Unfortunately, this property does not generally hold for linear systems. Consider for example the double integrator system

$$x_{k+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k, \tag{3.9}$$

subject to the convex state and input constraints

$$x_k \in \mathcal{X}, \quad u_k \in [-2, 2]. \tag{3.10}$$

Define a target set, $\mathcal{R}$, to be the convex hull of three points:

$$\mathcal{R} = \text{convhull}\left(\begin{bmatrix} 3 \\ 2 \end{bmatrix}, \begin{bmatrix} 2 \\ 2.5 \end{bmatrix}, \begin{bmatrix} 3 \\ 3 \end{bmatrix} \right). \tag{3.11}$$

The 1-step, 2-step, and 3-step controllable sets to $\mathcal{R}$ are plotted in Fig. 3.2. We also plot $\mathcal{C}$, the convex hull of $\mathcal{R}$ and the controllable sets. It is clear from the plot that $\mathcal{R}$ is not an invariant set. This means it is possible that states in the convex hull of trajectories leading into $\mathcal{R}$ are not N-step controllable to $\mathcal{R}$ for any value of N. In Fig. 3.2, this corresponds to states who are in $\mathcal{C}$, but not in any $\mathcal{K}_N(\mathcal{R})$.

Thus, we have shown that the convex hull of stored subtask executions is not generally controllable to the subtask transition set. □

Proposition 2. *All states in the time-indexed convex subtask safe sets (3.5) are controllable to the subtask transition set (3.3).*

The proof follows from Thm. 3 in Sec. 3.5.

3.4 Task Decomposition for Piecewise Linear ILMPC

Let Task 1 and Task 2 be different ordered sequences of the same M subtasks:

$$\mathcal{T}^1 = \{\mathcal{S}_i\}_{i=1}^M, \quad \mathcal{T}^2 = \{\mathcal{S}_{l_i}\}_{i=1}^M, \tag{3.12}$$

where the sequence $[l_1, l_2, ..., l_M]$ is a reordering of the sequence $[1, 2, ..., M]$. Assume non-empty subtask safe sets $\mathcal{KS}_{[1\to M]}$ (3.4) containing task data from $\mathcal{T}^1$.

Our goal is to use stored subtask safe sets (3.4) from $\mathcal{T}^1$ in order to find convex subtask safe sets (3.5) for $\mathcal{T}^2$. These sets can then be used to initalize a controller for the new task. The key intuition of TDMPC is that all successful subtask executions (2.13) from $\mathcal{T}^1$ are also successful subtask executions for $\mathcal{T}^2$, as this definition only depends on properties of each individual subtask, not the subtask sequence. With this notion, Alg. 2 proceeds backwards through the new subtask sequence.

TDMPC Algorithm

The notation $(l_M, \cdot)$ or $(i, \cdot)$ indicates that the described action is undertaken for all appropriate second arguments.

- Consider the last subtask, $\mathcal{S}_{l_M}$. By definition, for any state in $\mathcal{CK}_{l_M,\cdot}$ there exists a stored input sequence in $\mathcal{CU}_{l_M,\cdot}$ that can be applied such that the system evolves into $\mathcal{R}_{l_M}$. Thus, all states from $\mathcal{S}_{l_M}$ in $\mathcal{T}^1$ are also safe for $\mathcal{T}^2$. We next look for stored states from the preceding subtask, $\mathcal{S}_{l_{M-1}}$, which are controllable to $\mathcal{R}_{l_M}$ via $\mathcal{S}_{l_M}$. (Alg. 2, Lines 2-5).

- Form the sampled guard set of $\mathcal{S}_{l_{M-1}}$, defined as:

$$\mathcal{SG}_{l_{M-1}} = \mathcal{KS}_{l_{M-1},0}. \tag{3.13}$$

  The sampled guard set for subtask l_{M-1} contains the states in $\mathcal{S}_{l_{M-1}}$ from which the system transitioned into another subtask during a past execution of $\mathcal{T}^1$ (Alg. 2, Line 10).

- Determine which points in $\mathcal{SG}_{l_{M-1}}$ are one-step controllable to $\mathcal{CK}_{l_M,k}$ for some time index k. This problem can be solved using a variety of numerical approaches. In the results presented in this paper, we check controllability for each k, and choose the

Algorithm 2 TDMPC algorithm

1: **input** $\mathcal{KS}_{[1\to M]}$, $\mathcal{KU}_{[1\to M]}$, $[l_1, l_2, ..., l_M]$
2: **output:** $\mathcal{CK}_{[l_1\to l_M]}$, $\mathcal{CU}_{[l_1\to l_M]}$
3:
4: **do** $\mathcal{CK}_{[1\to M]} = \text{convexify}(\mathcal{KS}_{[1\to M]})$ (3.5)
5: **do** $\mathcal{SG}_i = \text{guard set clustering}(\mathcal{KS}_{[1\to M]})$ (3.13)
6: **initialize empty** $\hat{\mathcal{KS}}$, $\hat{\mathcal{KU}}$, $\hat{\mathcal{CK}}$, $\hat{\mathcal{CU}}$
7: $\hat{\mathcal{CK}}_{l_M,\cdot} \leftarrow \mathcal{CK}_{l_M,\cdot}$, $\hat{\mathcal{CU}}_{l_M,\cdot} \leftarrow \mathcal{CU}_{l_M,\cdot}$
8: **for** $i \in [l_{M-1} : -1 : l_1]$ **do**
9: $\quad\hat{\mathcal{KS}}_{i,\cdot} \leftarrow \mathcal{KS}_{i,\cdot}$
10: $\quad\hat{\mathcal{KU}}_{i,\cdot} \leftarrow \mathcal{KU}_{i,\cdot}$
11: $\quad$**initialize empty** $\hat{\mathcal{KU}}_{i,0}$
12: $\quad$**for** $x \in \mathcal{SG}_i$ **do**
13: $\quad\quad$**check** $(q^\star, u^\star) = \text{Ctrb}(x, \hat{\mathcal{CK}}_{i+1,\cdot})$ (3.14)
14: $\quad\quad$**if infeasible then**
15: $\quad\quad\quad\hat{\mathcal{KS}}_{i,\cdot} \leftarrow \hat{\mathcal{KS}}_{i,\cdot} \backslash \text{trajectory}(x)$
16: $\quad\quad\quad\hat{\mathcal{KU}}_{i,\cdot} \leftarrow \hat{\mathcal{KU}}_{i,\cdot} \backslash \text{trajectory}(u^\star)$
17: $\quad\quad$**else**
18: $\quad\quad\quad\hat{\mathcal{KU}}_{i,0} \leftarrow u^\star$
19: $\quad\hat{\mathcal{CK}}_{i,\cdot} \leftarrow \text{convexify}(\hat{\mathcal{KS}}_{i,\cdot})$ (3.5)
20: $\quad\hat{\mathcal{CU}}_{i,\cdot} \leftarrow \text{convexify}(\hat{\mathcal{KU}}_{i,\cdot})$ (3.5)
21:
22: **return** $\mathcal{CK}_{[l_1\to l_M]} \leftarrow \hat{\mathcal{CK}}$, $\mathcal{CU}_{[l_1\to l_M]} \leftarrow \hat{\mathcal{CU}}$

input u to minimize the cost of the resulting state according to (3.6). Specifically, for each point $x \in \mathcal{SG}_{l_{M-1}}$ and index k, we solve $(q^\star, u^\star) = \text{Ctrb}(x, \hat{\mathcal{CK}}_{l_M,k})$, where:

$$\begin{aligned} u^\star = \ \arg\min_{u} \quad & v(z) && (3.14)\\ \text{s.t.} \quad & z = A_{l_{M-1}}x + B_{l_{M-1}}u \\ & u \in \mathcal{U}_{l_{M-1}} \\ & z \in \hat{\mathcal{CK}}_{l_M,k}, \\ q^\star = v(A_{l_{M-1}}x + B_{l_{M-1}}u^\star). && (3.15) \end{aligned}$$

If (3.14) is feasible, the previously stored $\mathcal{T}^1$ cost-to-go (3.6) from the state x is replaced by $q^\star$, the cost to reach the goal set of $\mathcal{T}^2$. (Alg. 2, Line 11)

- For all states x in $\mathcal{SG}_{l_{M-1}}$ not controllable to any convex safe set in $\mathcal{S}_{l_M}$, we remove the stored subtask execution ending in x out of the set of subtask safe sets for $\mathcal{S}_{l_M}$. (Alg. 2, Lines 12-16)

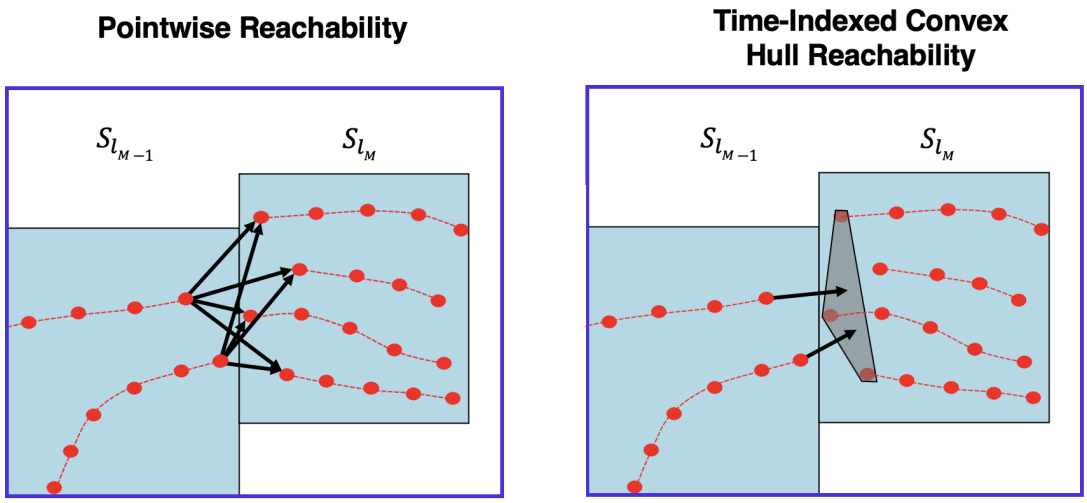


Figure 3.3: For systems with piecewise-linear dynamics and piecewise-convex constraints, the multiple pointwise controllability checks can be replaced with a single convex controllability check.

- After checking the entire sampled guard set, all remaining convex subtask safe sets for $\mathcal{S}_{l_{M-1}}$ are controllable to convex subtask safe sets in $\mathcal{S}_{l_M}$, and therefore also to $\mathcal{R}_{l_M}$. (Alg. 2, Lines 17-19)

Alg. 2 iterates backwards through the remaining subtasks, verifying the controllability of points in sampled guard sets to a convex subtask safe set in the following subtask. The algorithm returns convex subtask safe sets for $\mathcal{T}^2$ that can be used to initialize an ILMPC (3.7 - 3.8) for $\mathcal{T}^2$.

Note the reformulation of the stored $\mathcal{T}^1$ executions into convex sets (3.5). This allows us to replace the point-to-point controllability verification from Alg. 1 in Chapter 2 with point-to-set controllability in Alg. 2; Fig. 3.3 depicts this comparison. This allows for three major improvements to the procedure:

1. The reformulated controllability problem (3.14) is a convex optimization problem, which is, in general, much faster to solve than the non-convex point-to-point controllability.

2. By using the convex hull of stored states (3.5) as a target set in (3.14), rather than individual states, more points in the sampled guard sets (3.13) can potentially be demonstrated to lead to feasible $\mathcal{T}^2$ executions.

3. We increase the number of points for which we know a feasible $\mathcal{T}^2$ policy, since such a policy is known for all points in the time-indexed convex hulls of $\mathcal{T}^1$ trajectories (3.5), rather than only the discrete points of the $\mathcal{T}^1$ trajectories.

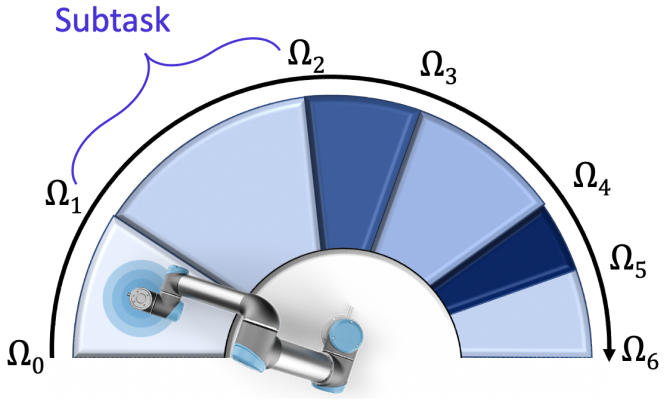


Figure 3.4: Topview of the robot path planning task. Each subtask corresponds to a pair of upper and lower obstacles.

Next, we show that the resulting time-indexed convex sets for $\mathcal{T}^2$ induce feasible ILMPC policies.

3.5 Properties of the PWL-TDMPC Policy

We prove the feasibility of ILMPC policies (3.8) initialized using Alg. 2. Note that here, $\mathcal{CK}^j$ denotes the convex subtask safe sets containing trajectories from the first j executions of a task.

Assumption 5. *$\mathcal{T}^1$ and $\mathcal{T}^2$ are defined as in (3.12), with each subtask defined by linear dynamics (3.1) and convex constraints (3.2).*

Theorem 3. *Let Assumptions 4-5 hold. Assume Alg. 2 outputs non-empty sets $\mathcal{CK}^0_{[l_1 \to l_M]}$ for $\mathcal{T}^2$. Then, if $x_0 \in \mathcal{CK}^0_{[l_1 \to l_M]}$, the policy $\pi^{\text{ILMPC}}_{[l_1 \to l_M]}$, as defined in (3.8), produces a feasible execution of $\mathcal{T}^2$.*

The proof is detailed in Sec. A.2. It follows from the same arguments that the ILMPC policy (3.7-3.8) will eventually bring any $x_0 \in \mathcal{CK}^J$ to the task target set $\mathcal{R}_{l_M}$.

3.6 Application: Robot Path Planning

Task Formulation

We demonstrate the effectiveness of Alg. 2 in the robotic path planning example introduced in Chapter 2. Consider a UR5e[1] robotic arm tasked with maneuvering through six sets of

[1]https://www.universal-robots.com/products/ur5-robot/

obstacles modeled as extruded disks of varying heights above and below the robot. Each set of upper and lower obstacles leaves a workspace between the disks for the robot to move between. Here, each subtask $\mathcal{S}_i$ corresponds to the workspace between a pair of lower and upper obstacles. Different subtask orderings correspond to a rearranging of the obstacle locations, indicated by Ω_i in Fig. 3.4.

In a certain subset of the state and input space, characterized experimentally, the UR5e has very high end-effector reference tracking accuracy (see Fig. 2.5). This allows us to use a simplified end-effector model in place of a discretized second-order model as in [110]. We solve the task in the reduced state space:

$$x_k = [q_{0_k}\ \dot{q}_{0_k}\ z_k\ \dot{z}_k]^\top$$
$$u_k = [\ddot{q}_{0_k}\ \ddot{z}_k]^\top,$$

where q_{0_k} is the angle of the robot's base joint along the Ω direction and z_k is the height of the robot end-effector at time step k, calculated from the six joint angles via forward kinematics. $\dot{q}_{0_k}$ and $\dot{z}_k$ are the corresponding velocities. We control $\ddot{q}_{0_k}$ and $\ddot{z}_k$, the accelerations of q_0 and z, respectively. We model the base-and-end-effector system as a quadruple integrator:

$$x_k \in \mathcal{X}_i \implies x_{k+1} = A_i x_k + B_i u_k \tag{3.17a}$$

$$A_i = \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix},\ B_i = \begin{bmatrix} 0 & 0 \\ dt & 0 \\ 0 & 0 \\ 0 & dt \end{bmatrix},\ \forall i \in [1, 6] \tag{3.17b}$$

where $dt = 0.01$ seconds is the sampling time. This simplified model holds as long as we operate within the region of high end-effector reference tracking accuracy, characterized in previous experiments.

This reduced state space allows us to formulate the task as a concatenation of $M = 6$ subtasks with piecewise affine dynamics and convex constraints, according to (2.11); thus we can use Alg. 2 to find a policy for a new task. For additional task description details, we refer to Sec. 2.6 in Chapter 2.

Experimental Results

We evaluate the efficiency of Alg. 2 by comparing its run-time with the the run-time of the point-to-point controllability analysis for task decomposition introduced in [122] for nonlinear systems.

First, an ILMPC (3.7)-(3.8) is used to complete five executions of five different training tasks, where each training task is a different reordering of the six obstacles. Each ILMPC is initialized with a suboptimal state and input trajectory that tracks the center-height of each subtask while the robot arm rotates at a low base velocity $\dot{q}_0$. In each task, the ILMPC tries to reach the target set as quickly as possible while avoiding the obstacles. The new

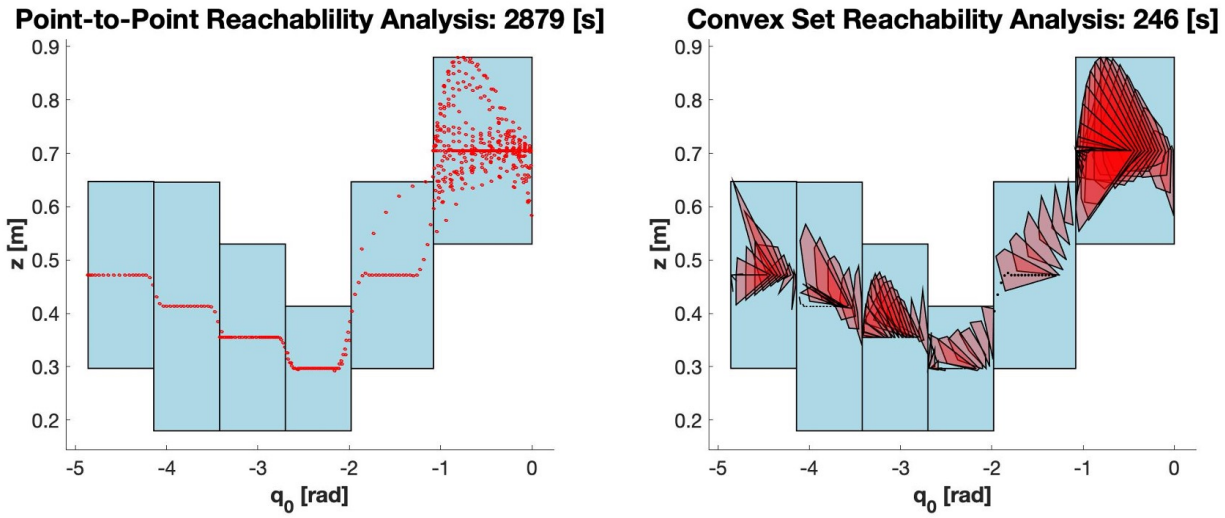


Figure 3.5: Alg. 2 produces a significantly larger set of feasible states for $\mathcal{T}^2$ in 10% of the time as the algorithm in [122]. The sampled guard sets for each subtask are plotted in black.

task $\mathcal{T}^2$ is configured from another new reshuffling of the obstacles. Fig. 3.5 depicts the $\mathcal{T}^2$ workspace in light blue, along with the $\mathcal{T}^2$ safe sets returned by the two versions of the TDMPC algorithm. The left image plots in red the feasible safe states for $\mathcal{T}^2$ output from the point-to-point controllability method from [122]. The right image shows in red the feasible safe sets output by the efficient point-to-convex-set controllability method from Alg. 2.

For each state in a subtask's sampled guard set, the point-to-point controllability method solves a mixed-integer program to try to find an input that controls the system to the last state of a subsequent subtask trajectory. Therefore the complexity of both controllability methods depends on the state dimension and number of trajectories through the subsequent subtask (as this provides an upper bound for the size of the subtask safe set). The efficiency improvement results from replacing the mixed-integer constraint in point-to-point controllability with a convex constraint of equal complexity, which is typically easier to compute.

In the example shown, our improved method was an order of magnitude faster at finding safe states for $\mathcal{T}^2$ than the point-to-point method, requiring 246 seconds of processing instead of 2879 seconds, using a 2017 Mac Book Pro with 2.8 GHz Quad-Core Intel Core i7. Indeed, the efficient reformulation of Alg. 2 for linear systems resulted in an average eleven fold speed-up for five different trials of the described setup and testing procedure. In Tab. 3.1, each trial corresponds to a newly shuffled $\mathcal{T}^2$.

As is clear from Fig. 3.5, the convex set controllability analysis outputs a significantly larger set of feasible states for $\mathcal{T}^2$ than the pointwise method. This results from two main phenomena. First, more states from the $\mathcal{T}^1$ sampled guard sets remain in $\mathcal{T}^2$ when using the convex set controllability than point-to-point controllability. This follows since the new

Trial	Convex Set Analysis	Pointwise Analysis
1	246 s	2879 s
2	312 s	5561 s
3	219 s	1618 s
4	212 s	1810 s
5	264 s	2806 s

Table 3.1: Controllability Analysis Run-Time

controllability analysis (3.14) considers a larger one-step target set than the pointwise controllablity analysis (i.e. a convex hull of states in the next subtask rather than individual states). As a result, more points in the sampled guard set can be shown to lead to feasible executions of $\mathcal{T}^2$. Second, while the point-to-point controllability analysis only checks for $\mathcal{T}^2$ feasibility of the actual subtask trajectories from $\mathcal{T}^1$, the new convex set controllability analysis automatically also provides a policy for the convex subtask safe sets induced by the trajectories. All safe states found using the point-to-point controllability method are thus also found using the convex set controllability method. Accordingly, an ILMPC (3.7-3.8) initialized with safe sets returned from Alg. 2 may also lead to a faster first execution of $\mathcal{T}^2$.

3.7 Conclusion

This chapter presented an extension to the Task Decomposition algorithm from Chapter 2. The new algorithm is designed for piecewise linear systems with piecewise convex constraint sets, and is shown to reduce the computational burden associated with the TDMPC algorithm and demonstrably increase the domain of the induced ILMPC policy. We prove that the resulting policies are guaranteed to lead to feasible executions of the new task. Finally, we evaluate the effectiveness of the proposed algorithm in a robotic path planning task, and demonstrate the reduced computational burden compared with the TDMPC algorithm for nonlinear systems.

Chapter 4

Probabilistically Safe Controllable Sets

4.1 Introduction

We again consider a constrained, discrete-time dynamical system solving a series of tasks. In each considered task $\mathcal{T}^i$, the system state and input constraints are identical, and the task environment imposes additional task-specific environmental constraints. Our goal is to use stored trajectories from previous tasks to find a control policy that solves a new task safely and effectively. The previous two chapters described efficient approaches for adapting stored task trajectories to the changed constraints of the new task; however, these methods required a priori knowledge of the entire new task environment in order to perform the backwards reachability analysis required to calculate controllable sets to the task goal.

In this chapter, we consider a situation in which the task-specific environmental constraints are at each time step k parameterized by a scenario parameter z_k which evolves according to a time-varying but task-invariant scenario dynamics function $z_{k+1} = \phi(z_k, k)$. (Consider, for example, time-varying hyperrectangular constraints with fixed sizes whose centers at any time step k are given by z_k.) We specifically consider the case where ϕ is unknown. Given a collection of state-input-parameter trajectories that solve a collection of n previous tasks (i.e. satisfying all system and respective task-dependent constraints), our goal is to develop an MPC control policy that results in a feasible trajectory for a new task $\mathcal{T}^{n+1}$ with a new sequence of time-varying scenario parameters.

MPC for systems with time-invariant constraints has been studied extensively, and it is well-understood how to design the policy to guarantee stability and feasibility. Significantly fewer studies exist that examine MPC with time-varying constraints; some are proposed in [73, 71, 76, 128, 134, 55, 83]. The authors in [73, 71, 76, 128] design controllers robust to possible changes in state constraints, but make assumptions on what those changes can be and require pre-calculating appropriate invariant sets. An approach using Control Lyapunov Functions and Control Barrier Functions for systems evolving on manifolds is presented in

[134]. Such functions can be difficult to design for arbitrary systems and requires an explicit model of how the constraints will vary. The authors of [55] and [83] formulate MPC problems with time-varying constraints specific to autonomous driving and inverted pendulum robots, respectively, but these works focus on how to recalculate time-varying constraints efficiently, rather than ensuring feasibility.

Here, we will utilize controllable set theory to find a controller satisfying all time-varying constraints in the new task. Controllable sets play an important role in safe MPC control design, including for autonomous systems [120, 57]. They represent areas of the state space from which the system can be safely controlled into a goal set $\mathcal{P}$, while satisfying all (possibly time-varying) state and input constraints. Thus, if the MPC can find a short-horizon trajectory reaching a controllable set to the task goal, there must also exist a longer trajectory ending in the task goal itself. Using controllable sets as terminal sets in MPC is the most common way to guarantee desirable properties such as stability and recursive feasibility [77, 15]. Our approach is therefore to find a controllable set at each time step of solving the new task $\mathcal{T}^{n+1}$, and using it as the terminal set in the MPC.

For linear systems with convex constraints, exact controllable sets can be calculated using iterative methods, most famously via the Matlab MPT toolbox [66]. While these iterative methods provide exact characterizations of controllable sets, the computational requirement scales quickly with system dimension, becoming unreasonable for real-time calculation for dimensions greater than four [2].

For constrained nonlinear systems, calculating controllable sets is generally considered to be impossible [93, 41], but many methods for calculating approximate controllable sets have been presented. One common approach is to linearize the system and constraints and find robust controllable sets that take into account induced linearization errors [48, 40].

Hamilton-Jacobi based backward reachability [79] or dynamic programming [13] methods can provide the most accurate approximations. An overview of these methods is given in [8]. However, these approaches famously scale very poorly with system dimension. Lyapunov-based methods can also provide good approximations [125, 92]. However, these methods require finding an appropriate Lyapunov function for the system, which can be very difficult as there are no clear guidelines for constructing Lyapunov functions for arbitrary systems [118, 16].

Because finding exact solutions is so difficult, recent work has considered using data-driven and sample-based approaches to find estimates of controllable sets [91, 24, 131, 33, 26, 64]. These approaches can be especially useful if the system dynamics are unknown; in these situations, trajectories of the system can be sampled and analyzed to estimate controllable sets. Most purely data-driven approaches cannot provide guarantees that the resulting set is a controllable set for the true system, though some offer probabilistic guarantees.

The extensive computational overhead required for the approximations described thus far mean that these calculations must happen offline. To ease this burden, new research has explored the possibility of analytically deriving controllable sets for certain classes of systems. Such analytic methods are desirable because they can provide simple expressions that need only be evaluated at a new state or constraint configuration, rather than entirely recalculated.

This could allow on-board computers to find controllable sets during a control task. The authors of [2] proposed a method for linear discrete-time systems, and analytic expressions for controllable sets of integrator systems are derived in [34]; these have known solutions for system dimension up to four. Extensive research has also led to analytic solutions of controllable sets in variations of the pursuit-evasion game [109, 17, 37]. These solutions are specific to the presented dynamics, and cannot be adapted to other scenarios.

In this chapter, we describe an approach that uses machine learning to quickly find approximate controllable sets for new, time-varying environmental constraints. In contrast with the previously introduced task decomposition approach, here we propose training and storing a learned strategy function, rather than storing the previous trajectories themselves. This strategy function is trained offline on stored trajectories, and implemented during the new control task, where its output is used to efficiently construct approximations of the controllable set to the new task's goal set. This controllable set estimate is then integrated into an MPC policy. Note that in contrast with [123], we do not require the tasks to be composed of the same subtasks—the new task's scenario parameter sequence can be entirely new.

Our approach consists of two main steps. Offline, before beginning the new task, stored trajectories are used to find ellipsoidal inner-approximations of controllable sets. Note that each of these sets is controllable with respect to the environment constraints imposed by the specific scenario parameter's evolution. A "strategy function" is then constructed that maps a scenario parameter to the center of such an ellipsoidal controllable set approximation. Online, while solving the new control task, the strategy function is evaluated and a new controllable set estimate is formed. Critically, this set is constructed so as to be with high probability contained in the true controllable set to the task goal.

In this chapter, we:

1. propose a method for estimating controllable sets from data for systems with time-varying constraints,

2. demonstrate how to use stored data to quickly find approximate controllable sets for new constraints, such that the approximation is with high probability a subset of the true controllable set,

3. demonstrate how to incorporate these sets into an MPC framework, and

4. provide probabilistic guarantees of feasible task completion for linear systems with convex time-varying constraints under the resulting MPC policy.

We emphasize that while machine learning has been used to find the boundaries of controllable sets for systems with time-invariant constraints, the novelty of the approach in this chapter is that we consider using machine learning to quickly construct controllable sets as a function of a time-varying constraint parameter. Our aim is to provide an alternative method based on stored trajectory data for representing analytical expressions for controllable sets that can be evaluated efficiently and can provide accuracy guarantees in some cases.

4.2 Problem Formulation

Problem Setup

We consider a discrete-time system with dynamical model

$$x_{k+1} = f(x_k, u_k), \tag{4.1}$$

subject to system state and input constraints

$$x_k \in \mathcal{X},\ u_k \in \mathcal{U}. \tag{4.2}$$

The vectors $x_k \in \mathbb{R}^{n_x}$ and $u_k \in \mathbb{R}^{n_u}$ collect the states and inputs at time k.

The system (4.1) solves a series of n finite-horizon control tasks, $\{\mathcal{T}^1, \ldots, \mathcal{T}^n\}$, each with fixed duration $T+1$. Each control task $\mathcal{T}^i$ is defined by the tuple

$$\mathcal{T}^i = \{\mathcal{X}, \mathcal{U}, \mathcal{P}^i, \mathbf{\Theta}^i\},$$

where $\mathcal{X}$ and $\mathcal{U}$ are the system state and input constraints (4.2). In this chapter, we consider environment parameter functions $\mathbf{\Theta}^i$ of the form

$$\mathbf{\Theta}^i(x_k, k) = z_k^i,$$

where $z_k^i \in \mathbb{R}^{n_z}$ denotes a scenario parameter specific to the i-th task at time k. The parameters evolve according to an unknown task-invariant, but time-varying, scenario dynamics function ϕ:

$$\begin{aligned}\mathbf{\Theta}^i(x_k, k) &= z_k^i \\ &= \phi(z_{k-1}^i, k-1) \\ &= \phi(\mathbf{\Theta}^i(x_{k-1}, k-1), k-1).\end{aligned}$$

We denote the resulting task-specific sequence of k environment scenario parameters as

$$\mathcal{S}^i = \{z_0^i, \ldots, z_T^i \mid z_{k+1}^i = \phi(z_k^i, k)\}. \tag{4.3}$$

In each control task the system model (4.1) and constraints (4.2) are identical. However, the task-specific scenario parameters $\mathcal{S}^i$ generate additional time-varying state constraints specific to each task's environment, denoted as

$$\mathcal{X}(\mathbf{\Theta}^i(x_k, k)) = \mathcal{X}(z_k^i) = \{x \mid h(x, z_k^i) \leq 0\},\ k = 0, \ldots, T. \tag{4.4}$$

As in our previous definitions of tasks, $\mathcal{P}^i$ represents the task's goal set, or the set of states the system must reach in order to complete the task. Here we specifically define

$$\mathcal{P}^i = \mathcal{X}_T,\ \forall i \in \mathbb{Z}^+.$$

Assumption 6. *In this chapter we consider tasks and environment scenarios that all share a goal set:*

$$\mathcal{X}(z_T^i) \subseteq \mathcal{X}_T, \ \forall i \in \mathbb{Z}^+.$$

As before, a feasible execution of the task $\mathcal{T}^i$ in environment scenario $\mathcal{S}^i$ is defined as a pair of state and input trajectories

$$\text{Ex}(\mathcal{T}^i, \mathcal{S}^i) = (\boldsymbol{x}^i, \boldsymbol{u}^i), \tag{4.5}$$

where

$$\begin{aligned} \boldsymbol{x}^i &= [x_0^i, x_1^i, \dots, x_T^i], \ x_k^i \in \mathcal{X} \cap \mathcal{X}(z_k^i), \ x_T^i \in \mathcal{P}^i \\ \boldsymbol{u}^i &= [u_0^i, u_1^i, \dots, u_{T-1}^i], \ u_k^i \in \mathcal{U}. \end{aligned}$$

Note that each closed-loop state x_k^i satisfies both the system state constraints and the task-specific environmental state constraints specified by the parameter z_k^i at time k.

Problem Statement

Consider a dynamical model (4.1) with state and input constraints (4.2), (4.4), and a collection $\mathcal{D}$ of feasible executions (4.5) that solve a series of n control tasks,

$$\mathcal{D} = \{\text{Ex}(\mathcal{T}^1, \mathcal{S}^1), ..., \text{Ex}(\mathcal{T}^n, \mathcal{S}^n)\}.$$

Our aim is to utilize $\mathcal{D}$ to find a data-driven state feedback control policy π such that for any new scenario $\mathcal{S}^{n+1}$ satisfying Asm. 6 and $x_0^{n+1} \in \mathcal{X} \cap \mathcal{X}(z_0^{n+1})$,

$$f(x_k, \pi(x_k)) \in \mathcal{X} \cap \mathcal{X}(z_{k+1}^{n+1}), \ \forall k = 0, \dots, T.$$

By definition, such a policy results in a feasible execution $\text{Ex}(\mathcal{T}^{n+1}, \mathcal{S}^{n+1})$.

Critically, we consider that at the start of the new task we do not have access to the entire scenario $\mathcal{S}^{n+1}$ nor the scenario dynamics function ϕ. Instead, at any time k of solving the new task we know only the current state x_k and a limited N-step scenario parameter forecast,

$$\mathbf{z}_{k:k+N}^{n+1} = [z_k^{n+1}, ..., z_{k+N}^{n+1}]. \tag{4.6}$$

Assumption 7. *In this chapter we assume that we have access to the true scenario parameter forecast, i.e. that there is no uncertainty in* $\mathbf{z}_{k:k+N}^{n+1}$.

Assumption 8. *We assume that there exists a feasible execution of* $\mathcal{T}^{n+1}$ *from initial state* x_0^{n+1} *which satisfies the environment constraints specified by the scenario parameters* $\mathcal{S}^{n+1}$.

Definitions and Notation

This chapter makes use of the following controllability definitions, reproduced here from Ch. 1.

Definition 5. *For a given set $\mathcal{R}$, the N-*step controllable set *$\mathcal{K}_N(\mathcal{R})$ of a system* (4.1) *subject to constraints* (4.2)-(5.3) *is defined recursively as:*

$$\begin{aligned}&\text{Pre}(\mathcal{R}) = \{x : \exists u \in \mathcal{U} : f(x,u) \in \mathcal{R}\}\\&\mathcal{K}_0(\mathcal{R}) = \mathcal{R}\\&\mathcal{K}_j(\mathcal{R}) = \text{Pre}(\mathcal{K}_{j-1}(\mathcal{R})) \cap \mathcal{X}(\boldsymbol{\Theta}),\ j \in \{1,...,N\}\,.\end{aligned}$$

For all states in the N-step controllable set to $\mathcal{R}$ there exists a feasible input sequence of length N that drives the system into $\mathcal{R}$ in N steps.

Definition 6. *The set $\mathcal{A}$ is* controllable *to a set $\mathcal{R}$ if there exists an $N > 0$ such that $\mathcal{A} \subseteq \mathcal{K}_N(\mathcal{R})$.*

Note that if an MPC can find a state trajectory ending in a terminal set that is controllable to the goal set while satisfying the task-specific time-varying environmental constraints, this guarantees the existence of a feasible state trajectory from the current state to the goal set.

In addition to these definitions, this chapter utilizes ellipsoidal sets. Here we write an ellipsoid as the image of the unit ball under an affine transformation

$$\text{Ell}(c,P) = \{x \mid (x-c)^\top P^{-1}(x-c) \leq 1\}, \tag{4.7}$$

where c is the center and P the shape matrix of the ellipsoid.

4.3 Probabilistically Safe Controllable Sets

As in the two previous chapters, our goal will be to use stored trajectory data to design terminal sets for an MPC policy of the form (1.11). Specifically, we will design a terminal set that is controllable to the task goal. If an iteration of the MPC can find a state and input sequence ending in a terminal set that is controllable to the task goal, by Def. 3 there is also a (longer) state and input sequence ending in the task goal.

Common approaches to finding controllable sets typically make one of two assumptions:

1. *Knowledge of the entire scenario $\mathcal{S}^{n+1}$*: If the entire task scenario were known, controllable sets to the goal $\mathcal{P}^{n+1}$ could be computed offline using Def. 1 and used as terminal sets in an MPC, resulting in closed-loop control with guaranteed recursive feasibility.

2. *Knowledge of ϕ*: If the scenario dynamics ϕ can be estimated from the limited scenario data (4.6) available at time k, controllable sets based on robust predictions of the future scenario parameters beyond time step $k+N$ could be computed. However, such robust reachability analysis is difficult for high-dimensional and nonlinear systems, and too time-consuming for online control.

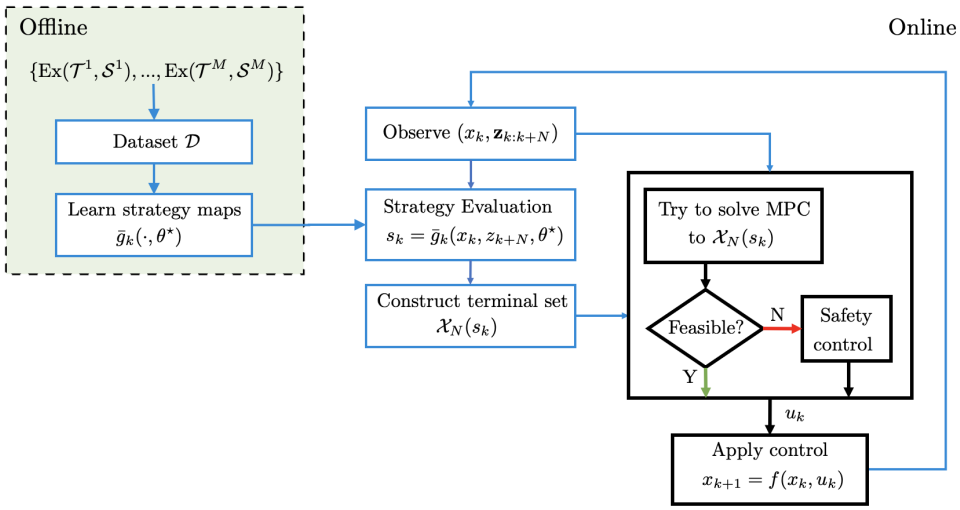


Figure 4.1: Offline, we learn a strategy map $\bar{g}$ from a dataset $\mathcal{D}$ containing stored executions from previous tasks. Online, at each time k, an N-step local environment forecast $\boldsymbol{z}_{k:k+N}$ is used to determine if a new high-level control strategy s_k is available. Strategies provide instructions how to construct a terminal set $\mathcal{X}_N(s_k)$ towards which to steer the system.

In our considered context, the limited environment look-ahead (4.6) poses a challenge. Greedy MPC control using only constraint information available at the current time step is likely to become infeasible at future time steps, especially if scenario parameters change quickly with respect to the control horizon.

In this chapter, we propose learning data-driven, strategy-parameterized sets that mimic the effect of an MPC terminal set that is controllable to a goal set. Before starting the new task $\mathcal{T}^{n+1}$, we use the stored executions in $\mathcal{D}$ to learn time-varying high-level strategy functions $\bar{g}_k$ for $k = 0, \ldots, T$ which map a state x_k and the furthest available environment forecast from (4.6) at time k to a strategy state s_k:

$$s_k = \bar{g}_k(x_k, z_{k+N}, \theta^\star), \tag{4.8}$$

where $\theta^\star$ are learned parameters. Online, at each time k of solving the new task $\mathcal{T}^{n+1}$, we evaluate the appropriate mapping (4.8) at the current state and environment forecast, and use the resulting strategy state to construct an MPC terminal set $\mathcal{X}_N(s_k)$. The construction approach of these sets will be presented in Sec. 4.6. Lastly, an MPC controller searches for

a feasible input sequence to steer the system into $\mathcal{X}_N(s_k)$:

$$\begin{aligned}
\mathbf{u}^\star(x_k, \mathbf{z}_{k:k+N}) = \underset{u_{k|k},\ldots,u_{k+N-1|k}}{\arg\min} \quad & \sum_{t=0}^{N-1} l_k(x_{k+t|k}, u_{k+t|k}) + l_N(x_{k+N|k})\cdot) \\
\text{s.t.} \quad & x_{k+t+1|k} = f(x_{k+t|k}, u_{k+t|k}) \\
& u_{k+t|k} \in \mathcal{U},\ \forall t \in \{0,\ldots,N-1\} \\
& x_{k+t|k} \in \mathcal{X} \cap \mathcal{X}(z_{k+t}),\ \forall t \in \{0,\ldots,N\} \\
& x_{k|k} = x_k \\
& x_{k+N|k} \in \mathcal{X}_N(s_k) \\
u_k = u^\star_{k|k}.
\end{aligned}$$

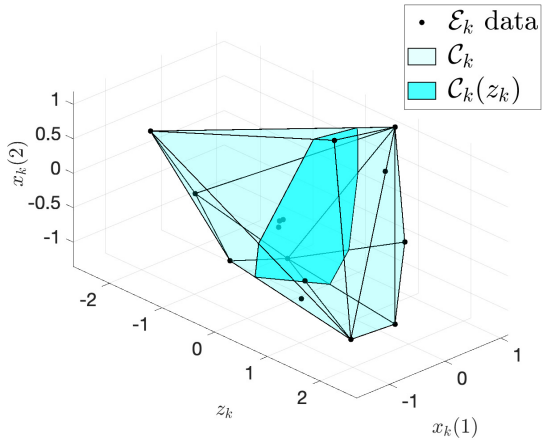
Our goal is to learn appropriate strategy mappings $\bar{g}_k$ in 4.8 so that using the resulting terminal sets $\mathcal{X}_N(s_k)$ in a low-level MPC leads to a feasible execution for the new task despite the limited forecast of the new scenario parameter sequence $\mathcal{S}^{n+1}$. An overview of our proposed control architecture is shown in Fig. 4.1. Our approach is split into three parts:

1. using stored data $\mathcal{D}$ to find parameterized approximations of controllable sets,
2. learning mappings $\bar{g}_k$ (4.8) from a current system state and environment forecast to a strategy state s_k that parameterizes an appropriate controllable set, and
3. using the strategy state s_k to construct an inner approximation $\mathcal{X}_N(s_k)$ of the $(T - (k+N))$-step controllable set to the task goal $\mathcal{P}$.

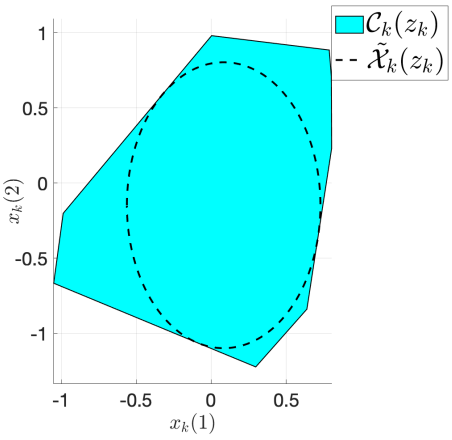
Each of these components is addressed in detail in the following sections. In Sec. 4.8 we also provide feasibility guarantees for the resulting closed-loop control for certain classes of systems.

A Note About Performance

In this chapter we specifically aim to use strategies to learn data-driven estimates of controllable sets to the task goal. Here the strategy state s_k only influences the low-level controller by changing the terminal constraint. The control objective function (e.g. minimize control effort) can be freely chosen by the control designer without affecting the recursive feasibility guarantees provided by our proposed terminal constraint. Critically, this also means that no restrictions are placed on the control objectives that guided the executions of previous tasks making up our stored data set $\mathcal{D}$. In contrast with [121], we do not make the implicit assumption that the objectives used to construct $\mathcal{D}$ are identical to each other or to the control objective for the new task $\mathcal{T}^{n+1}$.



(a) Steps II - IV: forming time-indexed discrete sets $\mathcal{E}_k$, taking the convex hull $\mathcal{C}_k$, and slicing at a specific scenario parameter z_k, resulting in $\mathcal{C}_k(z_k)$. Here, $x_k(1)$ and $x_k(2)$ are the first and second dimensions of the system state x at task time k.



(b) Step V: fitting an ellipse $\mathcal{X}_k(z_k)$ to the polytopic controllable set approximation $\mathcal{C}_k(z_k)$.

Figure 4.2: We use data from previous tasks to find ellipsoidal approximations of controllable sets for different scenario conditions.

4.4 Approximating Controllable Sets

Our first aim is to find estimates of controllable sets to the task goal $\mathcal{P}^{n+1} \supseteq \mathcal{X}(z_T^{n+1})$ using stored trajectories from previous tasks. These sets are then used as terminal sets in an MPC.

Exact Approaches

As stated in Def. 3, finding controllable sets to a goal set requires recursively intersecting the backward reachable set of the system (4.1) with the constraint sets applicable at each recursion step (4.2)-(4.4). Efficient methods for finding controllable sets exist for systems with linear dynamics and convex constraints [44, 67, 104, 95, 105], though these scale poorly with system dimension [39]. However, this is a difficult and computationally intensive problem in general, and is typically not well-suited to online computation [32, 38, 1].

Data-Driven Approximation

In order to avoid complex calculations to find exact controllable sets, we use data from previous tasks stored in $\mathcal{D}$ to find approximations of controllable sets. Note that because the scenario parameters impose additional constraints that must be satisfied at different

time steps, these controllable sets are specific to a particular task time step k and scenario parameter z_k. Our approach is outlined here, and visualized in Fig. 4.2.

Step 1: Form time-indexed sets of augmented states

We first form augmented states q which stack the recorded system state and environment parameter at a given time:

$$q_k = [x_k^\top, z_k^\top]^\top \in \mathbb{R}^{n_x+n_z}. \tag{4.9}$$

The augmented states from different tasks are collected according to the time at which they were recorded. At each time $k = 0, \ldots, T$, this set is defined as

$$\mathcal{E}_k = \bigcup_{i=1}^{n} q_k^i,$$

and contains the collection of state and scenario parameter pairs from which the system previously solved a particular control task at time k. Each augmented state in $\mathcal{E}_k$ concatenates a state x_k that is controllable to the goal state under environment constraints described by the corresponding scenario parameter z_k and its deterministic evolution 4.3 until the end of the task. Therefore any set $\mathcal{E}_k$ is a subset of the $(T-k)$-step controllable set to the task goal set under a specific scenario $\mathcal{S}^i$.

Step 2: Take the convex hull of time-indexed sets

To convert these collections of discrete points into polytopic sets, we take the convex hull of each set:

$$\begin{aligned} \mathcal{C}_k &= \left\{ \sum_{i=1}^{n} \lambda_i q_k^i \;\middle|\; \lambda_i \geq 0,\ \sum_{i=1}^{n} \lambda_i = 1,\ q_k^i \in \mathcal{E}_k \right\} \\ &= \{q \mid \begin{bmatrix} H_k^x & H_k^z \end{bmatrix} q \leq h_k\}, \end{aligned} \tag{4.10}$$

where $H_k^x \in \mathbb{R}^{p \times n_x}$ and $H_k^z \in \mathbb{R}^{p \times n_z}$. Note that the resulting set $\mathcal{C}_k$ now includes augmented states that were not included in $\mathcal{D}$, and have therefore not yet been explicitly shown in previous tasks to be controllable to the task goal.

Step 3: Evaluate at a particular scenario parameter

The convex hull sets $\mathcal{C}_k$ are in the space of augmented states (4.9). Given a particular scenario parameter $z_k = \bar{z}$ at time k, an approximation of the controllable system states can be determined by obtaining a slice of the set $\mathcal{C}_k$ at $\bar{z}$:

$$\begin{aligned} \mathcal{C}_k(z = \bar{z}) &= \{x \mid [x^\top, \bar{z}^\top]^\top \in \mathcal{C}_k\} \\ &= \{x \mid H_k^x x \leq h_k - H_k^z \bar{z}\}. \end{aligned}$$

The resulting set $\mathcal{C}_k(\bar{z})$ is a polytope, and therefore remains convex. An example is shown in Fig. 4.2a.

Step 4: Fit an ellipse to the convex hull

Polytopic sets can be challenging to parameterize, especially if the set is multi-faceted or high-dimensional (since either each vertex or hyperplane must be stored in order to characterize the set). To reduce the complexity of storing our approximated controllable sets $\mathcal{C}_k(z)$, as a last step we find the largest ellipsoid contained inside each $\mathcal{C}_k(z)$, i.e. a vector $c_k(z)$ and matrix $P_k(z)$ such that:

$$\mathcal{X}_k(z = \bar{z}) = \text{Ell}(c_k(\bar{z}), P_k(\bar{z})) \subset \mathcal{C}_k(\bar{z}).$$

These can be computed via the parameterized semi-definite program

$$\begin{aligned} g_k(z) = \arg \max_{c_k(z), P_k(z) \succ 0} \quad & \log\det P_k(z) \\ \text{s.t.} \quad & \|P_k(z) h_{k,i}^x\|_2 + {h_{k,i}^x}^\top c_k(z) \leq h_{k,i} - {h_{k,i}^z}^\top z \qquad \forall i = 1, \ldots, p. \end{aligned} \tag{4.11}$$

Where $h_{k,i}^x$ and $h_{k,i}^z$ are the i-th columns of ${H_k^x}^\top$ and ${H_k^z}^\top$ and $h_{k,i}$ is the i-th element of h_k. When solving (4.11), we may additionally constrain $P_k(z)$ to be a diagonal matrix in order to ensure that the resulting ellipse is axis-aligned. An example is shown in Fig. 4.2b.

These steps result in a collection of ellipsoidal, data-driven approximations of controllable sets, constructed using data stored in $\mathcal{D}$. We re-emphasize that each of these ellipsoids, parameterized by a center and shape matrix, approximates a set of system states at a particular time k and a specific scenario parameter z_k which are controllable to the goal set in $(T - k)$ steps.

Remark 3. *For systems with linear dynamics and convex state- and environment-dependent constraints, any point in the time-indexed convex hull (4.10) of controllable states is also controllable to the task goal. (The proof follows from Thm. 3, shown in A.2.) For such systems, this approach therefore results in an inner approximation of the true controllable sets, i.e.* $\text{Ell}(c_k(z_k), P_k(z_k)) \subset \mathcal{K}_{T-k}(\mathcal{X}(z_T))$. *In general, however, points in the convex hull of states that are individually controllable to a goal set are not necessarily also controllable to that set [124], though they provide an approximation [46, 78, 47].*

4.5 Learning Strategies To Approximate Controllable Sets

Section 4.4 discussed the first step of our approach: using stored data to find approximations of controllable sets to the task goal. These sets are ellipsoids parameterized by their center and shape matrix. When solving the new task $\mathcal{T}^{n+1}$, we will need to find these parameters for new ellipsoidal controllable sets at each time step, based on the current scenario parameter forecast (4.6). While it is certainly possible to find such parameters by evaluating g_k (4.11) for each new scenario parameter observed, it can be difficult to do so in real-time for high-dimensional systems and/or convex hulls with many facets (see Tab. 4.1, which lists the

	$p = n^3$	$p = n^4$	$p = n^5$
$n = 3$	0.81 [s]	0.88 [s]	1.16 [s]
$n = 4$	1.52 [s]	2.66 [s]	4.34 [s]
$n = 5$	9.33 [s]	30.65 [s]	80.14 [s]
$n = 6$	183 [s]	1001.7 [s]	3964.9 [s]

Table 4.1: As the state dimension n and number of convex hull facets p increases, the solve time for (4.11) increases significantly. Depending on the application, these are not suitable for real-time control tasks. Numbers represent the average duration of ten trials each calculated using Yalmip in Matlab.

computation time required to solve (4.11) in Matlab for a variety of state dimensions and facet numbers). Thus, our next aim is to find *efficient* mappings $\bar{g}_k$ from a new scenario parameter z_k^{n+1} to a controllable set parameterization.

We propose to use data stored in $\mathcal{D}$ to obtain such functions $\bar{g}_k$ which can be efficiently evaluated and approximate the functions g_k defined in (4.11). This will allow for quick computation of controllable ellipsoidal MPC terminal sets $\mathcal{X}_N(s_k)$ given a new scenario parameter. Specifically, we propose to use Gaussian processes (GPs) for this function approximation.

Gaussian Process Regression

A GP maintains a probability distribution over functions, and is completely specified by a mean function $\mu(x)$ and a scalar covariance function $k(x, x'|\theta)$ (also called the kernel) with hyperparameters θ. Suppose we are given a GP with $\mu(x) = 0$ and some kernel function $k(\cdot, \cdot|\theta)$ and want to estimate an unknown function $\psi : \mathbb{R}^n \to \mathbb{R}$. Then given a data set $\mathbf{X} = [x^1 \dots x^n]^\top$ and $\mathbf{Y} = [y^1 \ \dots \ y^n]^\top$ corresponding to noisy evaluations of the unknown function as $y^i = \psi(x^i) + w$ where $w \sim \mathcal{N}(0, \sigma_n^2)$, the posterior distribution of $\psi(x)$ is given by a GP specified by

$$\mu(x|\mathbf{X}, \mathbf{Y}, \theta) = \mathbf{k}(x)^\top (\mathbf{K} + \sigma_n^2 I)^{-1} \mathbf{Y} \tag{4.12}$$

$$\mathrm{Var}(x|\mathbf{X}, \mathbf{Y}, \theta) = k(x, x|\theta) - \mathbf{k}(x)^\top (\mathbf{K} + \sigma_n^2 I)^{-1} \mathbf{k}(x), \tag{4.13}$$

where

$$[\mathbf{K}]_{ij} = k(x^i, x^{(j)}|\theta)$$
$$\mathbf{k}(x) = [k(x, x^1|\theta), \dots, k(x, x^n|\theta)]^\top.$$

As more training data becomes available, the posterior GPs approximate the unknown function $\psi(\cdot)$ more accurately provided that it belongs to the *Reproducing Kernel Hilbert Space*

(RKHS) [89] induced by the kernel $k(\cdot,\cdot|\theta)$. Moreover, the kernel hyperparameters may be optimized by maximizing the marginal likelihood of the training observations using gradient-based methods.

GPs have been used in recent predictive control literature to obtain data-driven estimates of unknown nonlinear dynamics [61, 50, 54, 60]; here we use them to approximate g_k.

Training GPs

Using the available dataset $\mathcal{D}$ containing successful executions of n control tasks, the GP training data for each $\bar{g}_k$ consists of:

$$\begin{aligned}\mathbf{X}_k &= \{(x_k^i, z_{k+N}^i)\}_{i=1}^n \\ \mathbf{Y}_k &= \left\{\left(c_{k+N}(z_{k+N}^i)P_{k+N}(z_{k+N}^i)\right)\right\}_{i=1}^n .\end{aligned} \tag{4.14}$$

The output labels Y contain the center and shape matrix of the ellipse (4.11) approximating the $(T-(k+N))$-step controllable set from the scenario parameter z_{k+N}^i. These centers are calculated using the data from $\mathcal{D}$ as described in Sec. 4.4. Because GPs best approximate functions with scalar outputs, we train one GP for each dimension of the strategy state, for a total number of $(T-N)\cdot 2n_{\tilde{x}}$ GPs. Each GP uses the same training input data.

Once training data is formatted, we use Bayesian optimization to learn optimal kernel hyperparameters that best match the training data (4.14). A variety of solvers exist to automate this optimization, including GPyTorch and SKLearn in Python and the Machine Learning toolbox in Matlab. If desired, new hyperparameters can be learned whenever more executions become available.

Remark 4. *As stated in Asm. 7, we consider the case where at any time step k we have an exact measurement of z_{k+N}, the scenario parameter N steps into the future. If this is not the case, e.g. if we only have estimates of the future environment, then the entire N-step environment prediction $\boldsymbol{z}_{k:k+N}$ may be used as input to the GP. This additional information regarding the expected scenario trajectory may allow the GP to make more informed decisions in the face of environment uncertainty.*

Evaluating GPs

After training, the GPs can be evaluated on state and scenario data from the new task. At time k of a new task $\mathcal{T}^{n+1}$, we evaluate the GPs at a new query point $(x_k^{n+1}, z_{k+N}^{n+1})$. Each GP returns a one-dimensional Gaussian distribution over output scalars, parameterized by the posterior mean (4.12) and variance (4.13):

$$\mu(q_k) = [\bar{\tilde{c}}_{k+N}(z_{k+N}^{n+1}), \bar{\tilde{P}}_{k+N}(z_{k+N}^{n+1})] \tag{4.15}$$

$$\mathrm{Var}(q_k) = \mathrm{diag}([\sigma^2(\tilde{c}_{k+N}(z_{k+N}^{n+1})), \sigma^2(\tilde{P}_{k+N}(z_{k+N}^{n+1}))]). \tag{4.16}$$

The posterior mean (4.15) represents the GP's best guess based on the training data for the parameters of the ellipsoidal approximation to the controllable set from the scenario parameter z_{k+N}^{n+1}. The posterior variance (4.16) provides additional information about the uncertainty in each dimension of the parameter estimates. The ability to calculate this uncertainty using (4.13), and thus how confident the GP is in its prediction at a particular input, is a useful benefit of using GPs.

These statistics (4.15-4.16) represent the estimated strategy s_k at time k:

$$s_k = [\mu(q_k), \mathrm{Var}(q_k)] = \bar{g}_k(x_k, z_{k+N}, \theta^\star). \tag{4.17}$$

Next, we describe how we find controllable terminal sets $\mathcal{X}_N(s_k)$ once we have these estimates of the ellipsoid's parameters.

4.6 Applying Learned Strategies

The last step in our approach is to use the GP's calculated strategy state s_k (the estimates of ellipsoid parameters) to construct approximations of the $(T-(k+N))$-step controllable set to the task goal. This set can then be used as a terminal set in a low-level MPC. In particular, at each time k of solving the new task $\mathcal{T}^{n+1}$, we construct ellipsoidal sets $\mathcal{X}_N(s_k)$, parameterized as:

$$\mathcal{X}_N(s_k) = \mathrm{Ell}(\hat{c}_k, \hat{P}_k). \tag{4.18}$$

We would like these ellipsoids to be as similar as possible to the ellipsoids $\mathrm{Ell}(c_k, P_k)$ constructed in Sec. 4.4. Here we describe how to determine $\hat{c}_k$ and $\hat{P}_k$ from s_k to ensure that, with high probability, $\mathrm{Ell}(\hat{c}_k, \hat{P}_k) \subseteq \mathrm{Ell}(c_{k+N}(z_{k+N}^{n+1}), P_{k+N}(z_{k+N}^{n+1}))$.

Ellipse Center

At each time k of solving the new task, the state x_k and environmental forecast $\mathbf{z}_{k:k+N}^{n+1}$ are available. This information is used as input to the GPs (4.15-4.16), which provides estimates of the controllable ellipsoid's center and shape matrix. We define the center of $\mathcal{X}_N(s_k)$ to be the posterior mean of the estimated center:

$$\hat{c}_k = \bar{\tilde{c}}_{k+N}(z_{k+N}^{n+1}).$$

Ellipse Semi-Axes

As in Sec. 4.4, we restrict the semi-axes of the ellipsoid $\mathcal{X}_N(s_k)$ to be axis-aligned. This restriction ensures that the ellipsoidal shape matrix is a diagonal matrix, resulting in simplifications in the calculations required to ultimately construct $\mathcal{X}_N(s_k)$ in Sec. 4.6 (see, for example, the simplification introduced in Corollary 1). A downside to enforcing axis-aligned

ellipsoids is that the resulting controllable set estimate may provide less coverage of the true controllable set than an arbitrarily-aligned ellipsoid. To minimize this conservatism as much as possible, the axes should be chosen carefully according to the shapes of the time-varying constraints.

In particular, we want to choose the ellipsoid's shape matrix $\hat{P}_k$ (and thereby the axis lengths) such that with high probability $(1-\delta)$ the resulting ellipsoid is contained within the data-driven ellipsoidal approximation to the true controllable set $\mathcal{X}_{k+N}(z_{k+N})$:

$$\begin{aligned} \hat{P}_k = \arg\max_{\hat{P}} \quad & \log\det \hat{P} \\ \text{s.t.} \quad & \mathbb{P}[\text{Ell}(\hat{c}_k, \hat{P}) \subseteq \text{Ell}(c_{k+N}(z_{k+N}^{n+1}), P_{k+N}(z_{k+N}^{n+1}))] \geq (1-\delta). \end{aligned} \tag{4.19}$$

Our approach to solve (4.19) consists of two steps:

1. rewriting the set containment constraint as a function of the distance between two ellipsoid centers $\hat{c}_k$ and $c_{k+N}(z_{k+N}^{n+1})$, and
2. finding a probabilistic bound on this distance between the ellipsoid centers.

We first rewrite the containment constraint. For this, we require the following result.

Lemma 1. *Consider the nondegenerate ellipsoids* $\mathcal{E} = \{x \in \mathbb{R}^n \mid (x-c)^\top P^{-1}(x-c) \leq 1\}$ *and* $\hat{\mathcal{E}} = \{x \in \mathbb{R}^n \mid (x-\hat{c})^\top \hat{P}^{-1}(x-\hat{c}) \leq 1\}$*, where* $\|c-\hat{c}\|_2 \leq \epsilon$ *and* $P \succeq \Gamma$*. Then* $\hat{\mathcal{E}} \subseteq \mathcal{E}$*, if there exist* $\lambda > 0$ *and* $\hat{P} \succ 0$ *such that the following inequalities hold:*

$$\begin{aligned} \lambda\Gamma - \hat{P} &\succ 0 \\ \frac{\lambda\epsilon^2}{\lambda_{\min}(\lambda\Gamma - \hat{P})} + (\lambda - 1) &\leq 0. \end{aligned}$$

The proof makes use of the Schur complement and matrix inversion lemma, and is included in Sec. A.3.

For axis-aligned ellipsoids, the following corollary allows us to apply element-wise bounds on the center and shape matrix diagonal:

Corollary 1. *If the ellipsoids in Lemma 1 are axis-aligned, i.e.* $P = \text{diag}(p_1, \ldots, p_n)$ *and* $\hat{P} = \text{diag}(\hat{p}_1, \ldots, \hat{p}_n)$*, where* $|c_i - \hat{c}_i| \leq \epsilon_i$*, and* $p_i \geq \gamma_i$ *for* $i = 1, \ldots, n$*. Then* $\hat{\mathcal{E}} \subseteq \mathcal{E}$*, if there exist* $\lambda > 0$ *and* $\hat{p}_i > 0$ *such that the following inequalities hold:*

$$\begin{aligned} \lambda\gamma_i - \hat{p}_i &> 0 \\ \sum_{i=1}^{n} \frac{\lambda\epsilon_i^2}{\lambda\gamma_i - \hat{p}_i} + (\lambda - 1) &\leq 0, \ \forall i \in \{1, n\}. \end{aligned}$$

Given an upper bound ϵ on the distance between $c_{k+N}(z_{k+N}^{n+1})$ and $\hat{c}_k$, the centers of two ellipsoids, and a lower bound Γ on the size of the shape matrix, Lemma 1 provides

conditions on the shape matrix $\hat{P}$ such that the resulting ellipse $\text{Ell}(\hat{c}_k, \hat{P}_k)$ is contained within $\text{Ell}(c_{k+N}(z_{k+N}^{n+1}), P_{k+N}(z_{k+N}^{n+1}))$. The values for ϵ and Γ are determined from s_k using the uniform error bound provided in [68], which we reproduce below in Lemma 2. Specifically, Lemma 2 provides conditions on the function g_{k+N} (4.11) and the GP kernel $k(\cdot, \cdot|\theta)$ such that the deviation between the outputs of the GP mean function (4.17) evaluated at an input and the function g_k evaluated at the same input can be uniformly bounded, i.e. that there exist constants α and β such that

$$P\left[|g_{k+N,j}(z) - \bar{g}_{k,j}(z)| \leq \alpha\sqrt{\sigma_j^2(z)} + \beta, \quad \forall x \in \mathcal{X}\right] \geq 1 - \delta$$

holds jointly over the domain $\mathcal{X}$ of g_{k+N} for a chosen probability level $(1-\delta)$, where the subscript j indicates the j-th element of the vector valued functions.

Lemma 2. *(Theorem 3.1 in [68]) Consider a zero mean Gaussian process defined through the continuous covariance kernel $k(\cdot, \cdot)$ with Lipschitz constant L_k on the compact set $\mathbb{X}$. Furthermore, consider a continuous unknown function $f : \mathbb{X} \to \mathbb{R}$, sampled from a Gaussian process $GP(0, k(x, x'))$, with Lipschitz constant L_f and $B \in \mathbb{N}$ observations $y_i = f(x_i) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. Then, the posterior mean function $\mu(\cdot)$ and standard deviation $\sigma(\cdot)$ of a Gaussian process conditioned on the training data $\{(x_i, y_i)\}_{i=1}^B$ are continuous with Lipschitz constant L_μ and modulus continuity $\omega_\sigma(\cdot)$ on $\mathbb{X}$. Moreover, pick $\delta \in (0, 1)$, $\tau \in \mathbb{R}_+$ and set*

$$\begin{aligned}\beta(\tau) &= 2\log\frac{M(\tau, \mathbb{X})}{\delta}\\ \gamma(\tau) &= (L_\mu + L_f)\tau + \sqrt{\beta(\tau)}\omega_\sigma(\tau).\end{aligned}$$

Then, it holds that

$$P\left[|f(x) - \mu(x)| \leq \sqrt{\beta(\tau)}\sigma(x) + \gamma(\tau), \ \forall x \in \mathbb{X}\right] \geq 1 - \delta.$$

Details on how to compute the constants in Lemma 2 can be found in Sec. 4.6. Thus, Lemma 2 provides a probabilistic bound on the deviation between the posterior mean of a Gaussian process and the function g_k from which the GP training data was sampled, which holds jointly over the entire input space $\mathbb{X}$. The proof of Lemma 2 utilizes the Lipschitz continuity of the posterior mean and modulus of continuity in order to extend previous found bounds for a discrete input domain [111] to a continuous input domain. Critically, this Lipschitz-based approach results in bounds that contain easily calculable parameters, rather than other related works which utilize RKHS norms.

Using the bounds

$$\begin{aligned}\epsilon &= \text{Var}(c_{k+N}(z_{k+N}))\sqrt{\beta(\tau)} + \gamma(\tau)\\ \Gamma &= \mu(P_{k+N}(z_{k+N})) - \text{Var}(P_{k+N}(z_{k+N}))\sqrt{\beta(\tau)}\end{aligned}$$

in Lemma 1, we can rewrite (4.19) as:

$$\begin{aligned} \hat{P}_k = \arg \max_{\lambda>0, \hat{P}\succ 0} \quad & \log\det \hat{P} \\ \text{s.t.} \quad & \lambda\Gamma - \hat{P} \succ 0 \\ & \frac{\lambda\epsilon^2}{\lambda_{\min}(\lambda\Gamma - \hat{P})} + (\lambda - 1) \leq 0. \end{aligned} \tag{4.20}$$

If $\text{Ell}(c_{k+N}(z_{k+N}^{n+1}), P_{k+N}(z_{k+N}^{n+1})) \subseteq \mathcal{K}_{T-(k+N)}(\mathcal{P}^{n+1})$, then the resulting ellipsoid $\text{Ell}(\hat{c}_k, \hat{P}_k)$ is with probability $(1-\delta)$ contained within the controllable set.

Remark 5. *Lemma 2 provides a probabilistic bound on the deviation between the mean function of a GP and the function g_{k+N} from which the GP training data was sampled. Here, g_{k+N} is the function mapping a state and forecast to the center of the ellipse $\mathcal{X}_{k+N}(z_{k+N})$, formed as described in Sec. 4.4. Note that each function g_{k+N} is only defined in the range of the collected task data $\mathcal{C}_{k+N}$. Thus, although the GP can be evaluated outside this range, the guarantees provided by (4.20) only hold in the span of collected task data.*

Implementation

The uniform error bound described by Lemma 2 requires that the function being approximated is Lipschitz continuous with constant L_f. We now show that the function which maps an environment parameter $\bar{z}$ to the center of the maximum volume inscribed ellipsoid of $\mathcal{C}_k(\bar{z})$ (as described in Sec. 4.4) satisfies this requirement.

Lemma 3. *Consider the following parameterized optimization problem and let $\mathcal{Z} \subseteq \mathbb{R}^{n_z}$ be a compact set of parameters such that the set of feasible solutions within the compact set $x \in \mathcal{X} \subseteq \mathbb{R}^n$ is nonempty,*

$$\begin{aligned} g(z) = \arg\min_x \quad & l(x,z) \\ \text{s.t.} \quad & h_i(x,z) \leq 0,\ i \in \{1,\dots,m\} \end{aligned} \tag{4.21}$$

where $l : \mathbb{R}^n \times \mathcal{Z} \mapsto \mathbb{R}$ and $h_i : \mathbb{R}^n \times \mathcal{Z} \mapsto \mathbb{R}$ are convex w.r.t. x with $\nabla_x^2 l(x,z) \succ 0$, $\forall z \in \mathcal{Z}$. All functions are $\mathcal{C}^2$ in x and $\mathcal{C}^1$ in z. If for all $z \in \mathcal{Z}$, the optimal solution $x^\star$ is unique and LICQ and complementary slackness holds, then the map $g : \mathcal{Z} \mapsto \mathbb{R}^n$ is continuously differentiable.

Proof. Define the function $\xi : \mathbb{R}^n \times \mathbb{R}^m \times \mathcal{Z} \mapsto \mathbb{R}^n \times \mathbb{R}^m$:

$$\xi(x,\lambda,z) = \begin{bmatrix} \nabla_x l(x,z) + \sum_{i=1}^m \lambda_i \nabla_x h_i(x,z) \\ \lambda_1 h_1(x,z) \\ \vdots \\ \lambda_m h_m(x,z) \end{bmatrix}.$$

It is clear from the statement of the Lemma that for any $\bar{z} \in \mathcal{Z}$, (4.21) is a convex optimization problem for which strong duality holds. Therefore, for the primal and dual solution $x^\star$ and $\lambda^\star$ corresponding to $\bar{z}$, we have that $\xi(x^\star, \lambda^\star, \bar{z}) = 0$. Taking the Jacobian of ξ w.r.t. x and λ, we have

$$D_{x,\lambda}\xi(x^\star, \lambda^\star, \bar{z}) = \begin{bmatrix} A(x^\star, \lambda^\star, \bar{z}) & B(x^\star, \bar{z}) \\ \Lambda B(x^\star, \bar{z})^\top & C(x^\star, \bar{z}) \end{bmatrix},$$

where

$$\begin{aligned} A(x^\star, \lambda^\star, \bar{z}) &= \nabla_x^2 l(x^\star, \bar{z}) + \sum_{i=1}^{m} \lambda_i^\star \nabla_x^2 h_i(x^\star, \bar{z}) \\ B(x^\star, \bar{z}) &= \begin{bmatrix} \nabla_x h_1(x^\star, \bar{z}) & \dots & \nabla_x h_m(x^\star, \bar{z}) \end{bmatrix} \\ C(x^\star, \bar{z}) &= \operatorname{diag}(h_1(x^\star, \bar{z}), \dots, h_m(x^\star, \bar{z})) \\ \Lambda &= \operatorname{diag}(\lambda_1^\star, \dots, \lambda_m^\star). \end{aligned}$$

The block matrix $D_{x,\lambda}\xi$ is invertible when A and $C - \Lambda B^\top A^{-1} B$ (the Schur complement of A) are both invertible. It is straightforward to see that A is invertible from the assumption on $\nabla_x^2 l$, i.e. $A \succ 0$.

Let $\mathcal{I}(x^\star, \bar{z}) = \{1, \dots, \bar{m}\}$ denote the set of active inequality constraints at $x^\star$ and $\bar{z}$. Due to the complementary slackness condition, we have that $h_i(x^\star, \bar{z}) = 0$ and $\lambda_i^\star > 0$ for $i \in \mathcal{I}(x^\star, \bar{z})$ (note that any active constraints, where both $h_i(x^\star, \bar{z}) = 0$ and $\lambda_i^\star = 0$, may be removed from (4.21) without affecting the optimal solution) and $h_i(x^\star, \bar{z}) < 0$ and $\lambda_i^\star = 0$ for $i \notin \mathcal{I}(x^\star, \bar{z})$. We therefore have that

$$\begin{aligned} &C - \Lambda B^\top A^{-1} B \\ &\quad = \begin{bmatrix} \mathbf{0} & \\ & C_2 \end{bmatrix} - \begin{bmatrix} \Lambda_1 B_1^\top A^{-1} B_1 & \Lambda_1 B_1^\top A^{-1} B_2 \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \\ &\quad = \begin{bmatrix} -\Lambda_1 B_1^\top A^{-1} B_1 & -\Lambda_1 B_1^\top A^{-1} B_2 \\ \mathbf{0} & C_2 \end{bmatrix}, \end{aligned}$$

where we have partitioned the matrices B, C, and Λ into submatrices corresponding to the active and inactive inequality constraints respectively. Since $x^\star$ satisfies LICQ, B_1 is full column rank, which implies that $\Lambda_1 B_1^\top A^{-1} B_1 \succ 0$. Finally, the complementary slackness requirement ensures that $C_2 \prec 0$, and we have that $D_{x,\lambda}\xi$ is full rank and therefore invertible at $(x^\star, \lambda^\star, \bar{z})$.

By the implicit function theorem, in an open neighborhood U which contains $\bar{z}$, there exists a unique continuously differentiable map $g' : U \mapsto \mathbb{R}^n \times \mathbb{R}^m$ such that $g'(\bar{z}) = (\bar{x}, \bar{\lambda})$ and $\xi(\bar{x}, \bar{\lambda}, \bar{z}) = 0$. By strong duality, we have that $(\bar{x}, \bar{\lambda})$ must be the solution to (4.21) at $\bar{z}$. Define a function which returns the portion of $g'(\bar{z})$ corresponding to the primal solution. Since the choice of $\bar{z} \in \mathcal{Z}$ was arbitrary, this function is equivalent to g as defined in (4.21), which concludes the proof. □

While (4.11) does have a unique solution [49], it does not immediately satisfy the requirements of Lemma 3 since the Hessian of the cost function is not positive definite. To achieve this, we can easily modify the objective in (4.11) to arrive at the following strictly convex optimization problem

$$\begin{aligned} \arg\max_{c_k, P_k \succ 0} \quad & \log\det P_k - \frac{\alpha}{2}\|c_k\|_2^2 - \frac{\beta}{2}\|P_k\|_F^2 \\ \text{s.t.} \quad & \|P_k a_{k,i}\|_2 + a_{k,i}^\top c_k \leq b_{k,i} - h_{k,i}^\top \bar{z},\ i = 1, \ldots, p, \end{aligned} \tag{4.22}$$

where we have added regularization terms for c_k and P_k in the cost function with constants $\alpha, \beta > 0$. Using (4.22) in place of (4.11) when constructing approximate controllable sets from data may result in slightly different ellipsoidal sets and, therefore, GP training data. However, we note that as $\alpha, \beta \to 0$, the solution of (4.22) converges to that of (4.11).

Another potential issue with (4.11) and (4.22) is that for a large number of constraints, i.e. a polytope with a large number of facets, it is possible that the number of active constraints exceeds the dimension of the decision space, which would violate the assumption of the satisfied LICQ conditions required in Lemma 2. However, since we are obtaining the convex hull from data, we may always limit the number of data points used (at the cost of conservatism) in the construction of the convex hull. An alternative would be to further modify (4.22) such that we may obtain unique solutions which are suboptimal with respect to (4.11), but are strictly in the interior of the feasible set. We can then use Slater's condition to establish strong duality in Lemma 3 and obtain the same result.

Lastly, Lemma 3 requires that complementary slackness holds at each optimal solution $x^\star$ to (4.11), i.e. that no active constraints are also redundant. In the context of (4.11), this means that no hyperplanes of the convex hull are tangent to the largest-volume ellipse without constraining the maximum volume the ellipse could have had. While it is theoretically possible that such a situation occurs, in which case the complementary slackness condition is not satisfied, this is extremely unlikely for randomly chosen initial conditions of the task executions and multi-dimensional systems. Furthermore, as in the case of the LICQ requirement, by pruning the number of data points used to construct the convex hull, redundant active constraints can easily be removed in order to satisfy complementary slackness.

Lemma 2 also requires that each unknown function g_k is a sample from a Gaussian process with zero mean and kernel k. Note that many so-called universal kernels [112], including the squared exponential kernel, can be used to represent continuous functions with arbitrary precision. Given the results in Lemma 3 confirming the continuity of g_k, this requirement is therefore not restrictive.

Finally, Lems. 1-2 depend on various measures and constants that must be chosen appropriately for good performance and applicability.

- L_f: the Lipschitz constant of g_k (existence of which is guaranteed by Lemma 3) can be estimated from collected data. Alternatively, a probabilistic approach for estimating L_f from collected data is provided in [68].

- L_μ and ω_σ: these values can be calculated from the Lipschitz constant of the kernel k and L_f. Formulas for determining these values, which depend only on the training data and kernel expressions, are included in [68].

- τ and $M(\tau, \mathbb{X})$: here, τ represents a grid constant which is used to derive the theorem in [68]. The covering number $M(\tau, \mathbb{X})$ is the minimum number of points in a grid on $\mathbb{X}$ with grid constant τ. While this number can be difficult to calculate in general, upper bounds can be easily computed. Note that $\beta(\tau)$ only grows logarithmically with diminishing τ, so that τ can be chosen arbitrarily small so that the effect of $\gamma(\tau)$ becomes negligible compared to $\sqrt{\beta(\tau)}\omega_\sigma(\tau)$.

Thus, the quantities in Lemma 1 and Lemma 2 are all easily computed or estimated. This is in contrast with various previous approaches for bounding GP learning errors, which often require knowing the maximal information gains of training data and RKHS norms of the functions to be estimated. These are very difficult to calculate, which has historically resulted in limited rigorous application of the methods in control research.

The Time-Invariant Case

So far we have considered the case where the scenario dynamics function ϕ explicitly depends on time k. If this is not the case, i.e. if $z^i_{k+1} = \phi(z^i_k)$ does not depend on the current time step k, a simpler approach than the one presented thus far can be taken.

Rather than constructing k polyhedral approximations to k controllable sets, one convex set $\mathcal{C}$ can be found by taking the convex hull of *all* recorded data (states and environment parameters), across all k timesteps. This polyhedron can then be sliced and ellipsoids fitted as described, but unified across all timesteps. A single GP function $\bar{g}$ can then be used to approximate the mapping from a new environment parameter z to the strategy state s_k. Besides the construction of a single convex set $\mathcal{C}$ and a single GP, the rest of the framework does not change.

This approach reduces the complexity of estimating controllable sets as well as the number of different strategy functions that have to be stored. However, because training time of GPs scales with the cube of the input dimension (4.14), training time may significantly increase a GP is trained with all data points.

4.7 Low-Level Controller

The low-level MPC controller calculates the input to be applied to the system at each time k. This input is calculated based on the terminal set $\mathcal{X}_N(s_k)$ constructed around the strategy state s_k, calculated using the GP. If no feasible input sequence ending in $\mathcal{X}_N(s_k)$ can be found, a safety controller is applied. Here, we formalize this control approach. Guarantees of recursive feasibility for linear systems are provided in Sec. 4.8.

MPC Formulation

At each time k of solving the new task $\mathcal{T}^{n+1}$, we construct a terminal set $\mathcal{X}_N(s_k)$ according to Secs. 4.5-4.6. These sets are used as a terminal set in an MPC,

$$\begin{aligned} \boldsymbol{u}^\star(x_k, z_{k:k+N}) = \underset{u_{k|k},\ldots,u_{k+N-1|k}}{\arg\min} \quad & \sum_{k=0}^{N-1} p(x_{k+t|k}, u_{k+t|k}) + q(x_{k+N|k}) \\ \text{s.t.} \quad & x_{k+t+1|k} = f(x_{k+t|k}, u_{k+t|k}) \\ & u_{k+t|k} \in \mathcal{U},\ \forall t \in \{0,\ldots,N-1\} \\ & x_{k+t|k} \in \mathcal{X} \cap \mathcal{X}(z_{k+t}),\ \forall t \in \{0,\ldots,N\} \\ & x_{k|k} = x_k \\ & x_{k+N|k} \in \mathcal{X}_N(s_k) \end{aligned} \tag{4.23}$$

$$u_k = u^\star_{k|k}, \tag{4.24}$$

which searches for a feasible state and input trajectory that minimizes a chosen stage cost p and terminal cost q, and ends in the terminal set $\mathcal{X}_N(s_k)$. This optimization problem (4.23) is solved at each time k of solving the new task, and if a feasible input trajectory exists, the first optimal input $u^\star_{t|t}$ is applied to the system.

Safety Control

It can occur that (4.23) is an infeasible problem; in other words, that no state-input trajectory satisfying system and environmental constraints exists that ends in the constructed terminal set. When this happens, a safety control is activated for time k only, and we choose case

$u_k = u^s_{k|k}$, where

$$\boldsymbol{u}^s(x_k, z_{k:k+N}) = \underset{u_{k|k},\ldots,u_{k+N-1|k},\boldsymbol{\lambda}}{\arg\min} \sum_{k=0}^{N-1} p(x_{k+t|k}, u_{k+t|k}) + q(x_{k+N|k}) \tag{4.25}$$

$$\begin{aligned} \text{s.t.} \quad & x_{k+t+1|k} = f(x_{k+t|k}, u_{k+t|k}) \\ & u_{k+t|k} \in \mathcal{U},\ \forall t \in \{0,\ldots,N-1\} \\ & x_{k+t|k} \in \mathcal{X} \cap \mathcal{X}(z_{t+k}),\ \forall t \in \{0,\ldots,N\} \\ & x_{k|k} = x_k \\ & x_{k+N|k} = \sum_{i=1}^{|\mathcal{E}_{k+N}|} \lambda^i x^i_{k+N} \\ & \boldsymbol{\lambda} \geq 0,\ \sum_{i=1}^{|\mathcal{E}_{k+N}|} \lambda^i = 1 \end{aligned}$$

$$u_k = u^s_{k|k}. \tag{4.26}$$

The only difference between the safety controller (4.25-4.26) and the strategy MPC (4.23-4.24) is the terminal set constraint. The safety controller searches for a state and input trajectory that ends in the convex hull of states from previous trajectories, $\mathcal{E}_{k+N}$. (Note that if ϕ is not time-varying, the safety controller terminal constraint will be the convex hull of all stored data.)

The safety controller controls the system in a safe manner until a time when a feasible terminal set $\mathcal{X}_N(s_k)$ is found. Our control approach outlined in Alg. 3 ensures that the safety controller can always be applied. We show this in Sec. 4.8.

Algorithm

Algorithm 3 summarizes the proposed control policy. Importantly, the approach only requires a local N-step forecast of the new task environment, rather than the entire environment scenario. Gaussian processes, trained offline on trajectories from past control tasks, are used to construct data-driven estimates of controllable sets to the task goal set $\mathcal{P}$. These sets are constructed at each time step, and used as terminal sets in an MPC controller. When no feasible input sequence can be found, a safety controller is used at that time step instead.

4.8 Properties of PSCS Policies

For systems with linear dynamics (4.1) and convex state and input constraints (4.2), solving tasks with convex environment constraints (4.4) evolving according to a linear scenario dynamics function ϕ, the convex hull approximation made in Step 3 in Sec. 4.4 is exact. As

Algorithm 3 PSCS Control Policy

parameters: $\lambda, \Gamma, N, \delta$
input: f, $\mathcal{X}$, $\mathcal{U}$, $\{\text{Ex}(\mathcal{T}^1, \mathcal{S}^1), ..., \text{Ex}(\mathcal{T}^M, \mathcal{S}^M)\}$
output: policy π for $\text{Ex}(\mathcal{T}^{n+1}, \mathcal{S}^{n+1})$

offline:
construct approximate controllable sets as in Sec. 4.4
train GPs using stored executions as in Sec. 4.5

online:
for each time step k **do**
  collect $(x_k^{n+1}, z_{k:k+N}^{n+1})$
  evaluate $s_k = \bar{g}_k(x_k^{n+1}, z_{k+N}^{n+1})$ using (4.15)
  construct $\mathcal{X}_N(s_k)$ using (4.18)
  solve MPC (4.23)
  if (4.23) is feasible **then**
    $u_k = u_{k|k}^\star$ (4.24)
  else
    solve safety MPC (4.25)
    $u_k = u_{k|k}^s$ (4.26)
end

a result, $\text{Ell}(c_{k+N}(\bar{z}), P_{k+N}(\bar{z})) \subseteq \mathcal{K}_{T-(k+N)}(\mathcal{P}^{n+1})$, and $\text{Ell}(\hat{c}_k, \hat{P}_k) \subseteq \text{Ell}(c_{k+N}(\bar{z}), P_{k+N}(\bar{z}))$ for any $\bar{z} = z_{k+N}^{n+1} \subseteq \mathcal{C}_{k+N}$ with probability $(1-\delta)$. This is stated in Thm. 4.

Assumption 9. *The system (4.1) has linear dynamics*

$$x_{k+1} = Ax_k + Bu_k, \tag{4.27}$$

and is subject to convex system state and input constraints (4.2)

$$H_x x_k \leq h_x$$
$$H_u u_k \leq u_x.$$

Furthermore, the task scenario function ϕ is also linear

$$z_{k+1} = C_k z_k, \tag{4.28}$$

and imposes convex environmental state constraints

$$H_z(x_k - z_k) \leq h_z. \tag{4.29}$$

Theorem 4. *Consider a system and set of $n+1$ tasks satisfying Asms. 6-9. The trajectories of the system solving the first n tasks are stored in $\mathcal{D}$, and data-driven estimates of controllable sets are calculated according to Secs. 4.4-4.6.*

If a new state and scenario parameter for task $\mathcal{T}^{n+1}$ are chosen such that $q_0^{n+1} = [x_0^{n+1}, z_0^{n+1}] \in \mathcal{C}_0$, then

$$\mathbb{P}[\text{Ell}(\hat{c}_k, \hat{P}_k) \subseteq \mathcal{K}_{T-(k+N)} \quad \forall k \in 0, ..., T-N] \geq 1-\delta \ . \tag{4.30}$$

The proof is detailed in Sec. A.3, and utilizes the fact that for linear systems the time-indexed convex hull of controllable states is also a controllable set to a task goal.

Theorem 4 states that for linear systems with convex system and environment constraints, the methods outlined in Secs.4.4-4.6 produce data-driven estimates of controllable sets that are, with chosen probability $(1-\delta)$, subsets of the true controllable set to the task goal with chosen probability. It follows that using $\mathcal{X}_N(s_k) = \text{Ell}(\hat{c}_k, \hat{P}_k)$ as an MPC terminal set for these types of systems results in a feasible execution of the new task with probability $(1-\delta)$. This is stated in Thm. 5.

Theorem 5. *Consider a system and set of $n+1$ tasks satisfying Asms. 6-9. If a new state and scenario parameter for task $\mathcal{T}^{n+1}$ are chosen such that $q_0^{n+1} = [x_0^{n+1}, z_0^{n+1}] \in \mathcal{C}_0$, then the control approach outlined in Alg. 3 results in a feasible execution of the new task* $\text{Ex}(\mathcal{T}^{n+1}, \mathcal{S}^{n+1})$ *with at least probability* $(1-\delta)$.

The proof can be found in Sec. A.3. The key idea is to show that if the initial augmented state q_0^{n+1} is in the convex hull of stored trajectory data, using the GP-based controller will result in a closed-loop trajectory that remains within the convex hull of data. Thus the safety controller can be applied whenever necessary, resulting in a successful closed-loop trajectory.

Remark 6. *The probabilistic bounds provided by Thm. 5 hold true for any function g_k with bounded Lipschitz constant. No linearity or convexity are required. However, Asms. 6-9 are required to ensure that g_k as defined in (4.11) is, in fact, the center of an ellipsoidal set that is controllable to the task goal. The controllable set approximations undertaken in Sec. 4.4 to define g_k are only guaranteed to be true for linear systems with convex constraints.*

4.9 Application: Integrator System

We evaluate Alg. 3 in a simulation example. Specifically, we examine the probability of feasibility in comparison to the safety controller. In Sec. 4.10, we further analyze the computational complexity of Alg. 3 and compare with similar approaches.

Problem Formulation

We consider a discrete-time, four-dimensional integrator system with state x_k and input u_k, evolving according to

$$x_{k+1} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ dt & 0 \\ 0 & dt \end{bmatrix} u_k,$$

where $dt = 0.1[s]$ corresponds to the discretization sampling time. The system state and input are box constrained as

$$-10 \leq x_k \leq 10$$
$$-5 \leq u_k \leq 5$$

for all time steps k.

In addition to the box system constraints, the system must satisfy time-varying environmental constraints. At each time k, the state is additionally constrained according to the element-wise inequality

$$x_k \in \mathcal{X}(z_k) = \{x \mid |x - z|_i \leq 1, \ \forall i \in n_x\}.$$

Thus at each time step k, the system state must be inside a square box of length 2 centered at z_k. Furthermore, z_k evolves according to

$$z_{k+1} = A_z z_k,$$

where A_z is the optimal LQR feedback to steer a point mass model to the origin with unity state and input costs. This feedback can be calculated using the discrete-time algebraic riccati equation. In each new task, the initial environment parameter z_0 changes, as does the resulting environment constraint sequence.

PSCS Implementation and Results

We consider the availability of trajectories solving $n = 5000$ different instances of this prob-Lemma Each trajectory is of length $T = 50$, and is achieved corresponds to the system in closed-loop with a unity-cost LQR controller with horizon $N = T = 50$. Thus the stored trajectories are solved using a batch approach, not receding horizon, and correspond to the best possible system response for each specific scenario parameter sequence.

These trajectories are used to construct approximate controllable sets as outlined in Sec. 4.4. In order to minimize the computational effort required to find the convex hull of such a large number of stored data points, we used a sampling-based approximation method to first reduce the cardinality of the data before finding the convex hull. Once the convex

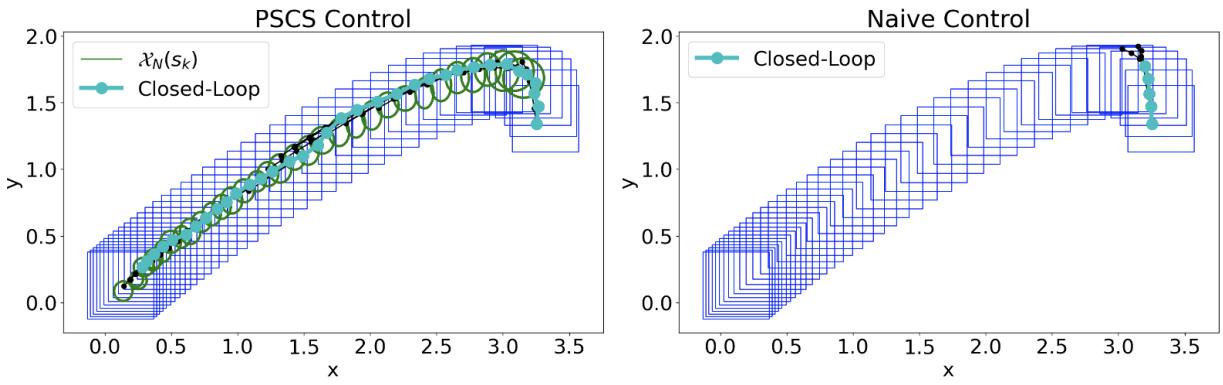


Figure 4.3: The PSCS Alg. 3 proposes a controllable set to the system at each time step, shown in green. Using this ellipsoidal set as a a terminal set in a low-level MPC results in a feasible closed-loop trajectory satisfying all time-varying constraints (blue boxes). The naive MPC without a terminal set fails to complete the task.

hulls were constructed, the training data for GPs was created. Specifically, we trained GPs with scaled Matern 1.5 kernels. The resulting GPs provided estimates of controllable sets to the task goal given a new scenario parameter. All trajectory creation and GP training was conducted in Python.

After GP training, the strategy functions are used to find controllable set estimates for a new task, $\mathcal{T}^{n+1}$, with environmental state constraints evolving from the initial parameter z_0^{n+1}. We select a desired constraint satisfaction rate of 99%. At each time step k, a 4-step environmental forecast is used to evaluate the GPs and construct appropriate terminal sets according to (4.18). These sets are then used in a low-level MPC controller (4.23) with control horizon $N = 4$.

An example trajectory is depicted in Fig. 4.3, which shows the system's closed-loop trajectory through the sequence of time-varying box constraints as it moves towards the task goal set at the origin. At each time step, the GP-constructed terminal set steers the system towards a part of the state space from which a feasible input sequence exists to the task goal set. The size of the terminal constraint set shrinks as we approach the end of the task—this can be seen both on the x-y trajectory plot and the individual state plots (Fig. 4.4). This occurs because as the system approaches the origin, the GP becomes less uncertain in its predictions; all previously seen task environments converged to the origin, so the GP has seen this situation before. As a result, the GP variances decrease and the terminal set according to (4.18) shrinks.

As is clear in Fig. 4.3, the system moves smoothly through the task, and is well within the task constraints at each time step. However, when we solved the task using a naive

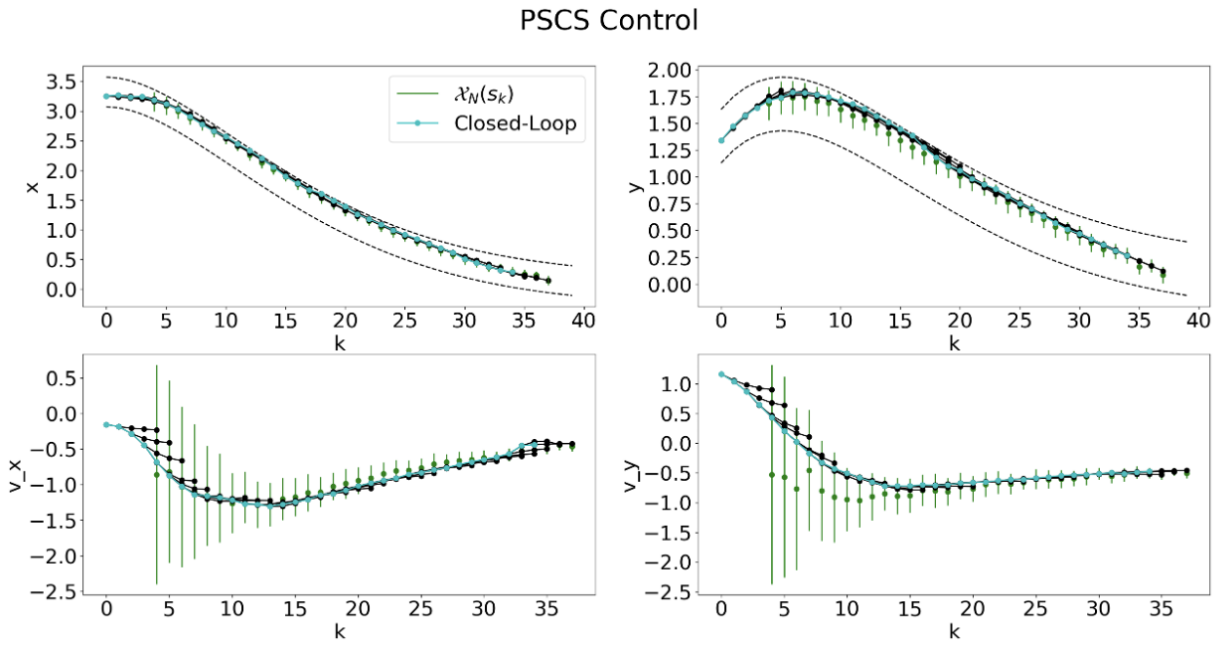


Figure 4.4: As the system solves the new task, it plans 4-step open-loop trajectories ending in the GP-constructed terminal sets (shown in green). The terminal sets ensure that the system will be able to satisfy future unknown environmental constraints.

MPC controller with planning horizon $N = 4$ and no terminal constraint, the closed-loop system becomes highly unstable. Because the short-sighted controller only plans to satisfy constraints up to $N = 4$ steps in advance, the controller is purely focused on minimizing the control objective. Shortly upon beginning the task, the controller becomes infeasible. In an evaluation of 850 tasks, the PSCS control found a feasible execution for 848 tasks, while the naive MPC feasibly completed only 462 tasks.

Even with added noise or state uncertainty, the system using the PSCS controller would be able to complete this task with use of the suggested terminal set. For further analysis, we evaluated our controller in a series of ten new tasks when subjecting the system to additive noise sampled from $\mathcal{N}(0, 0.02)$. The PSCS controller successfully completed all ten tasks, while the naive MPC controller became infeasible in each one. This clearly demonstrates the necessity for using an appropriate terminal in changing environments, and the PSCS provides a structured way of using data to construct such sets.

4.10 Discussion

Other Approaches

Traditional approaches for tackling the unknown environment with limited horizon problem would typically fall into three categories: *(i)* solving a short-horizon MPC problem without a terminal set, *(ii)* estimating ϕ from data, or *(iii)* interpolating between the stored data using convex hull methods. Here we compare our proposed algorithm with each such method.

Approach 1: Solving MPC with limited forecast

An alternative approach is to simply solve an MPC problem using the limited environment forecast $\boldsymbol{z}_{k:k+N}$ at each time step, without a controllable terminal set, as implemented by [55, 83]. While this method is easy to implement, there are no guarantees the controller is recursively feasible. Therefore it is possible that the closed-loop system will become unsafe at some point during the task. This can be prevented with chosen probability $(1-\delta)$ using Alg. 3.

Approach 2: Estimating ϕ from data

Given information from stored trajectories about how the scenario parameters z evolved in previous tasks, one approach is to estimate ϕ from this transition data. In the case of linear scenario functions, this is a simple regression problem, which can be completed entirely offline. Once an estimate $\hat{\phi}$ has been determined, the entire scenario $\mathcal{S}^{n+1}$ can be determined from the initial parameter z_0^{n+1}, and controllable sets to $\mathcal{P}$ can be calculated according to Def. 1. These controllable sets can then be used as terminal sets in an MPC. If ϕ can be estimated exactly, then using the calculated controllable sets as terminal sets in an MPC will result in a feasible task execution. This is explored in [73, 71, 76].

Unless ϕ is linear or otherwise clearly parameterized, estimating ϕ may not be straightforward. Additionally, calculating reachable sets for nonlinear dynamics is known to be very challenging. While the estimation of ϕ from stored data can be completed entirely offline, the scenario propagation and controllable set calculation can only be completed once z_0^{n+1} is known. This can be problematic, as the controllable set calculation can be time-intensive, and may not finish before the first input u_0 must be applied to the system. If the calculation does not finish quickly enough, the safety controller (4.25-4.26) can be applied for the first few time steps, which may lead to sub-optimal control performance.

If ϕ cannot be estimated exactly, then robust backwards reachable sets must be calculated. Depending on the uncertainty in $\hat{\phi}$, these robust backward reachable sets may be very small or even empty, resulting in an infeasible MPC. In contrast, the controller outlined in Alg. 3 evaluates very quickly, and does not require any backward propagation, with or without uncertainty.

Approach 3: Interpolating between stored data

A third approach is to write the initial state x_0^{n+1} and scenario parameter z_0^{n+1} as a convex combination of stored data points, and apply the corresponding convex combination of stored inputs. Note that this is a similar approach to our proposed safety controller (4.26). As shown in [98], if Asms. 6-9 hold, this results in a recursively feasible trajectory that satisfies all system and task constraints. Otherwise, there are no feasibility guarantees using this approach.

Depending on the dimension of the system and the size of the dataset $\mathcal{D}$, finding appropriate convex hull multipliers can be very slow. As was the case with Approach 2, this interpolation may take too long for a real-time control scenario. Additionally, because the input sequence is fixed once convex multipliers are found, no additional control objectives can be taken into account. In contrast, Alg. 3 only affects the terminal constraint; the control objective function can be chosen as desired.

Our Approach

In contrast with the above approaches, the framework outlined in Alg. 3 has three main benefits:

1. not requiring any online set propagation at the start of the control task,
2. taking uncertainty into account in a probabilistic way, thus allowing the control designer to trade off acceptable risk and performance, and
3. allowing for any control objective function.

Complexity Evaluation

Offline Evaluation

The proposed Alg. 3 requires significant offline calculation. Finding the convex hull of stored trajectory data (as required to find approximate controllable sets in Sec. 4.4) becomes very slow as the state dimension and number of states increases. Various approximation techniques exist to reduce the computational burden, including data pruning techniques before calculating the convex hull, but it remains a significant bottleneck. As demonstrated in Tab. 4.1, finding the largest ellipsoid inside a convex hull of points (as required to create the training data labels) scales similarly poorly. The last offline step is to train several GPs, one for each state dimension and trajectory time step. As previously described, GP training time scales with the number of training samples cubed.

Thus, the overhead required to train a function that estimates controllable sets is extensive. However, in contrast with the task decomposition methods proposed in Ch. 2-3, the algorithm outlined here does not have to repeated for each new task environment. The offline

calculations only have to be performed once in order to solve a variety of new tasks, and only need to be repeated as desired (e.g. to take more new task information into account).

Online Evaluation

At each time step k of solving the new task, Alg. 3 evaluates n_x GPs, solves an SDP in order to find the ellipse shape matrix, and solves a QCQP optimization problem to plan a trajectory ending in the ellipsoidal terminal set. We compare the online evaluation speed of the PSCS method with two other controllers:

1. an MPC controller that solves g_k (4.11) to exactly find the largest ellipse inside the convex hull, and utilizes it as a terminal set, and
2. the safety controller detailed in (4.25).

We evaluate these controllers on a double integrator system with dynamics

$$x_{k+1} = I_n x_k + I_{un} u_k,$$

where I_{n_x} is the identity matrix of state dimension n_x, and I_{un_x} a stacked vector with $n_x - 1$ zeros atop a single one. We consider various state dimension sizes n_x, using the same MPC horizon of $N = 10$. In each case, we consider the availability of p previously recorded trajectories, and that the convex hull of this stored trajectory data has already been calculated offline.

Results are shown in Tab. 4.2. Note that each entry contains the average evaluation time over 100 trials, normalized by the evaluation time required by the safety controller. For reference, the safety controller with $(n = 2, p = 8)$ required 0.02 seconds to evaluate, and $(n = 7, p = 7^6)$ required 0.04 seconds. As the state dimension or number of stored trajectories increases, solving g_k to find the MPC terminal set quickly becomes excessively slow. For a four-dimensional system with 64 recorded task trajectories, solving g_k to find the largest ellipse within the convex hull of the stored task data is more than nine times slower than using PSCS control. This demonstrates the value of using GPs to quickly approximate the g_k function. As a result of the excessive calculation times, we only tested the g_k MPC on a small subset of scenarios.

The trend is more complicated for PSCS. For systems with low state dimension n and small dataset sizes p, the PSCS controller takes nearly twice as long as the safety controller. However, as the state dimension or dataset size increase, the performance gap shrinks, until eventually the PSCS controller outperforms the safety controller in several scenarios. This is because convex hull calculations become very inefficient for large state dimensions and numbers of vertices—as the dataset grows, the size of the safety control optimization problem grows proportionally, requiring more auxiliary variables. In contrast, the size of the PSCS optimization problem only depends on the state dimension. The PSCS controller does require a GP evaluation, but this can be very efficiently implemented using programs such as GPyTorch.

PSCS / g_k MPC	$p = n^3$	$p = n^4$	$p = n^5$	$p = n^6$	$p = n^8$
$n = 2$	**1.81** / 3.44	**1.84** / 3.55	**1.90** / —	**1.97** / —	**1.83** / —
$n = 3$	**1.71** / 5.78	**1.69** / 6.36	**1.67** / —	**1.59** / —	**0.93** / —
$n = 4$	**1.80** / 16.6	**1.72** / 33.3	**1.50** / —	**1.11** / —	—
$n = 5$	**1.53** / 107	**1.64** / —	**1.20** / —	**0.49** / —	—
$n = 6$	**1.68** / —	**1.43** / —	**0.74** / —	—	—
$n = 7$	**1.57** / —	**1.15** / —	**0.47** / —	—	—

Table 4.2: We compare the online evaluation time required at each time step for three different control approaches. Numbers represent the average duration of 100 trials each calculated using Mosek in Python, normalized by the evaluation time required by the safety controller for each n and p combination. For reference, the safety controller with $(n = 2, p = 8)$ required 0.02 seconds to evaluate, and $(n = 7, p = 7^5)$ required 0.04 seconds.

Controller Region of Attraction

GPs are typically used to represent nonlinear functions, for which the convex hull is an inexact approximation. However, the results in Tab. 4.2 demonstrate that PSCS can offer utility even for linear systems with convex constraints. While for low-dimensional systems it is faster to replace the GP approximation with the convex hull, for high-dimensional systems it is more efficient to use the GP approximation.

However, we note that the PSCS controller does have a smaller region of attraction than the convex hull safety controller. This is a direct result of the PSCS controller learning to construct sets that are inner approximations of the true controllable sets (see Sec. 4.4). Therefore there may be parts of the state space from which an MPC controller with a convex hull terminal constraint is feasible, but an MPC with a (smaller) PSCS-constructed terminal set is not. The degree to which the PSCS region of attraction is smaller than that of the convex hull controller of course depends on how well axis-aligned ellipsoids can cover the area of the controllable sets.

In simulation trials of 850 different tasks for which the convex hull controller was feasible, 848 were successful using the PSCS controller in conjunction with the backup safety controller (i.e. as described in Alg. 3). When we additionally tested the performance of a pure-PSCS controller (i.e. no safety controller was ever implemented; if the PSCS-constructed terminal set was infeasible at any time step, the task execution ended as a failure), it resulted in successful completion of 786 tasks. Thus, much of the shrinkage in the region of attraction can be well-mitigated by utilizing the safety controller when necessary.

4.11 Conclusion

In this chapter, we proposed a data-driven hierarchical framework for safe model predictive control in unknown environments with time-varying constraints. We consider a discrete-time dynamical system and the availability of state-input trajectories that solve a variety of tasks in various environments. In each task, the system state and input constraints are identical, but a task-specific environment model generates additional task-specific state constraints which are satisfied by the respective trajectories. Our framework provides a method for estimating controllable sets from stored trajectories solving previous tasks, as a function of the parameterized environment model. While we assume that the environment constraints in previous tasks were all different, the approach requires that the environment constraints evolve according to a shared time-varying dynamics function and that all tasks share a common task goal.

Offline, before beginning the new task, Gaussian process strategy functions are learned from stored data. Online, while solving the new task, the strategy functions are evaluated at a short-term environmental forecast, and the output is used to construct an ellipsoidal approximation of the controllable set to the task goal set. Finally, the set estimates are used as terminal sets in a low-level MPC; the MPC objective function can be designed as desired,

irrespective of the estimated controllable set. For linear systems with convex system and environment constraints, we prove that the ellipsoidal sets are subsets of the true controllable sets with arbitrarily high probability, and are thus true controllable sets to the task goal. As a result, we can demonstrate our approach is guaranteed to result in a feasible execution for the new task. We evaluated this approach in various applications, and provide extensive comparisons with other methods for estimating controllable sets and performing MPC in time-varying environments.

In the next chapter, we more broadly consider the notion of *strategies* in navigation tasks, how they may be extracted from trajectory data, and safely integrated in a low-level MPC.

Extensions

The approach presented in this chapter considers a specific instantiation of the changing environment problem, and makes several assumptions about the problem setup. Here we briefly discuss how Alg. 3 could be extended in various ways.

Variable task length:

At the beginning of this chapter we made the assumption that all recorded task trajectories are of the same finite length T. This is critical as our proposed approach utilizes time-indexed strategy functions $\bar{g}_k$ to construct controllable set estimates. If the task goal set $\mathcal{P}$ is an invariant set, our approach still holds for variable task lengths. Otherwise, a single more general, time-independent strategy function could be used. This function $\bar{g}$ could consider the entire environment forecast $\boldsymbol{z}^i_{k:k+N}$ as input, which may provide additional insight, though retaining probabilistic containment guarantees is not straightforward in this case.

Multiple possible scenario dynamics functions ϕ:

Here we have assumed that in each considered task, the environment constraints evolve according to a fixed function ϕ. Another worthy extension is to consider the situation where each task evolves according to some $\phi^i \in \Phi$, where Φ is a finite set of functions each ϕ^i could be. In such a case, ϕ^i could be estimated from the environment forecast $\boldsymbol{z}^i_{k:k+N}$, either explicitly before estimating the strategy or implicitly by including it as additional input to the GP. Alternatively, an initial classifier scheme could convert the environment forecast into a probability distribution over Φ, and the p number of most likely $\phi \in \Phi$ could be considered individually.

Expanding guarantees to nonlinear systems:

While Alg. 3 can certainly be used to construct controllable set approximations for nonlinear systems, the probabilistic guarantees described in Thms. 4-5 only hold for linear systems. The linear dynamics requirement stems from the fact that the entire convex hull of controllable states is guaranteed to also be controllable only if the system is linear and constraints

convex. Thus the controllable set approximation described in Sec. 4.4 is only exact for linear systems. If another way could be proposed for constructing approximate controllable sets that is also exact for nonlinear systems, our guarantees could hold. However, this is a difficult problem in general. Note that an additional consideration for nonlinear systems would be the design of an appropriate safety controller (4.25).

Chapter 5

Hierarchical Predictive Learning

5.1 Introduction

The previous three chapters proposed different ways for using stored task data to find approximations of controllable sets for the new task, either by directly using trajectory libraries or a function encoding the trajectory libraries. These controllable set estimates could then be used as a terminal set in a low-level MPC.

In Chapter 4, we began considering the notion of using strategies to represent controllable sets. Now, we formalize this notion of using stored data to learn generalizable control strategies for navigation tasks. The inspiration for this work was how humans learn navigation tasks and then generalize them: by learning environment-dependent strategies, rather than specific actions.

We will introduce a hierarchical predictive learning architecture that learns such strategies from stored task data and then applies the strategies to the new task. At each time step, based on a local forecast of the new task environment, the learned strategy consists of a target region in a reduced-dimension state space and input constraints to guide the system evolution to this target region. These target regions are used as terminal sets by a low-level model predictive controller.

In this chapter, we show how to *(i)* design the target sets from past data, and then *(ii)* incorporate them into a model predictive control scheme with shifting horizon that ensures safety of the closed-loop system when performing the new task. We prove the feasibility of the resulting control policy, and apply the proposed method to robotic path planning, racing, and computer game applications. We will conclude with a discussion on the benefits using a hierarchical framework can offer for control generalizability, modularity, transparency, and safety.

This chapter is organized as follows. Section 5.2 formalizes the background information and problem statement. Section 5.3 introduces the Hierarchical Predictive Learning Control Framework, which is detailed further in Secs. 5.4-5.7. We conclude with a proof of the feasibility of our proposed policy and three example applications. We conclude this chapter

with a discussion in Sec. 5.12.

The results presented in this chapter have also appeared in:

- C. Vallon and F. Borrelli. "Data-driven hierarchical predictive learning in unknown environments." In: *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. 2020, pp. 104-109.
- C. Vallon and F. Borrelli. "Data-driven strategies for hierarchical predictive control in unknown environments." To appear in: *2021 Transactions on Automation Science and Engineering (TASE)*.

5.2 Problem Formulation

As before, we consider a discrete-time system with dynamical model

$$x_{k+1} = f(x_k, u_k), \tag{5.1}$$

subject to system state and input constraints

$$x_k \in \mathcal{X},\ u_k \in \mathcal{U}. \tag{5.2}$$

The system (5.1) solves a series of n control tasks $\{\mathcal{T}^1, \dots, \mathcal{T}^n\}$. Each control task $\mathcal{T}^i$ is defined by the tuple

$$\mathcal{T}^i = \{\mathcal{X}, \mathcal{U}, \mathcal{P}^i, \mathbf{\Theta}^i\},$$

where $\mathcal{X}$ and $\mathcal{U}$ are the system state and input constraints. $\mathcal{P}^i \subset \mathcal{X}$ denotes the task target set the system needs to reach in order to complete task $\mathcal{T}^i$.

Recall that for each task $\mathcal{T}^i$, the environment descriptor function $\mathbf{\Theta}^i$ maps the state x_k at time k to a description of the local task environment. In this chapter we only consider state-dependent, but time-invariant, functions $\mathbf{\Theta}^i(x_k)$. The set of states satisfying the environmental constraints imposed by this local task environment are denoted by $E(\mathbf{\Theta}^i(x_k))$. We write the joint system and environment constraints as

$$x_k \in \mathcal{X}(\mathbf{\Theta}^i(x_k)) = E(\mathbf{\Theta}^i(x_k)) \cap \mathcal{X}. \tag{5.3}$$

For notational simplicity, wherever it is obvious we will drop the state dependence and denote the combined system and environment constraints (5.3) as $\mathcal{X}(\mathbf{\Theta}^i)$.

The control architecture proposed in this chapter relies on forecasts of the task environment, which can be obtained using the environment descriptor function. At time k, we can use the system model (5.1) to predict the system state across a horizon T,

$$\hat{x}^i_{k:k+T} = [\hat{x}^i_k, \hat{x}^i_{k+1}, \dots, \hat{x}^i_{k+T}],$$

and evaluate the environment descriptor function along this forecast state trajectory. This provides a forecast of both the upcoming task environment $\boldsymbol{\Theta}^i(\cdot)$ and environmental constraints $E(\boldsymbol{\Theta}^i(\cdot))$:

$$\theta^i_{k:k+T} = [\boldsymbol{\Theta}^i(\hat{x}^i_k), \ldots, \boldsymbol{\Theta}^i(\hat{x}^i_{k+T})]. \tag{5.4}$$

In the context of the autonomous racing example, we can evaluate $\boldsymbol{\Theta}^i$ at forecasts of the system state in order to get predictions for the upcoming racetrack curvature, which can be evaluated from e.g. camera images (Fig. 1.1).

Problem Definition

Given a dynamical model (5.1) with state and input constraints (5.2), (5.3), and a collection of feasible executions (2.5) that solve a series of n control tasks, $\{\text{Ex}(\mathcal{T}^1, \boldsymbol{\Theta}^1), ..., \text{Ex}(\mathcal{T}^n, \boldsymbol{\Theta}^n)\}$, our aim is to find a data-driven control policy $\pi(x)$ that results in a feasible and high-performance execution of a new task in a new environment: $\text{Ex}(\mathcal{T}^{n+1}, \boldsymbol{\Theta}^{n+1})$.

In addition to satisfying the new environment constraints, the execution should try to minimize a desired objective function $J(\boldsymbol{x}^{n+1}, \boldsymbol{u}^{n+1})$. In contrast with Chapter 2-3, in this section we do not explicitly take a cost function J into account. Instead, we assume that the stored executions from previous tasks $\mathcal{T}^i$ were collected using control policies that aimed to minimize the same cost function $J(\boldsymbol{x}^i, \boldsymbol{u}^i)$. Thus the desired objective function J is implicitly associated with the stored data. If this assumption does not hold, the method proposed in this chapter will still provide a feasible trajectory.

5.3 Hierarchical Predictive Learning Control

We propose a data-driven controller that uses stored executions from previous tasks to find a feasible policy for a new task in a new environment. Instead of simply adapting the stored executions from previous tasks to the changed environmental constraints of the new task, we learn generalizable and interpretable *strategies* from past task data, and apply them to the new task. Our approach is inspired by how navigation tasks are typically explained to humans, who can easily generalize their learning to new environments by learning strategies.

A Motivating Example

Consider learning how to race a vehicle around a track. If a human has learned to race a vehicle by driving around a single track, they can easily adapt their learned strategy when racing a new track.

A snippet of common racing strategies[1] taught to new racers is depicted in Fig. 5.1. The most basic rules are guidelines for how to find the racing line, or the fastest possible path

[1] As taught at online racing schools such as Driver 61: `https://driver61.com/uni/racing-line/`

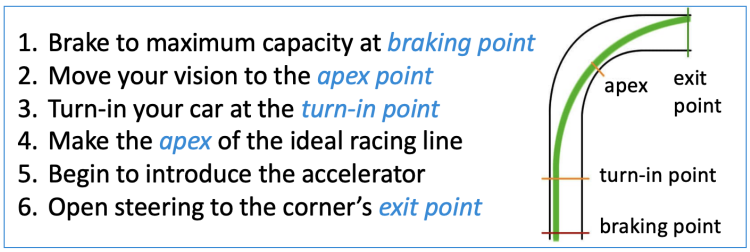


Figure 5.1: Sample of strategies taught at various online racing schools.

around the track, which is directly determined by the track curvature. Drivers are instructed where along the track to brake, steer, and accelerate, and how to find these locations for curves of different shapes.

The environmental constraints imposed on the vehicle (lane boundaries) are parameterized by the track curvature. This environmental descriptor gives rise to physical areas along the track with respect to which racing rules are then explained, e.g. "brake at the *braking point*, then cut the curve at the *apex* and aim for the outside of the straightaway." The strategies here consist of sections of the track towards which to aim the vehicle as well as acceleration profiles to apply along the way. Importantly, the locations of these regions only depend on the local curvature; the track curvature a mile away has little impact on the location of the apex in the curve directly ahead.

We also note that the strategies are explained using only a subset of the state space: the distance from the centerline. Given guidelines on this subset, the driver is free to adjust other states and inputs such as vehicle velocity and steering in order to satisfy environmental constraints.

Principles of Strategy

Based on this real-life intuition, we propose three principles of navigation strategy:

1. Strategies are a function of a local environment forecast
   (*e.g. radius of curvature of an upcoming track segment*)

2. Strategies work in a reduced-order state space
   (*e.g. distance from center lane*)

3. Strategies provide target regions in the (reduced-order) state space for which to aim, and input guidelines for getting there
   (*e.g. "braking point", "turn-in point", "exit point")*

The control architecture proposed in this chapter formalizes the above principles of strategy, and shows how to incorporate such strategies into a hierarchical learning control framework. In particular, we focus on two aspects. First, we show how to learn generalizable strategies

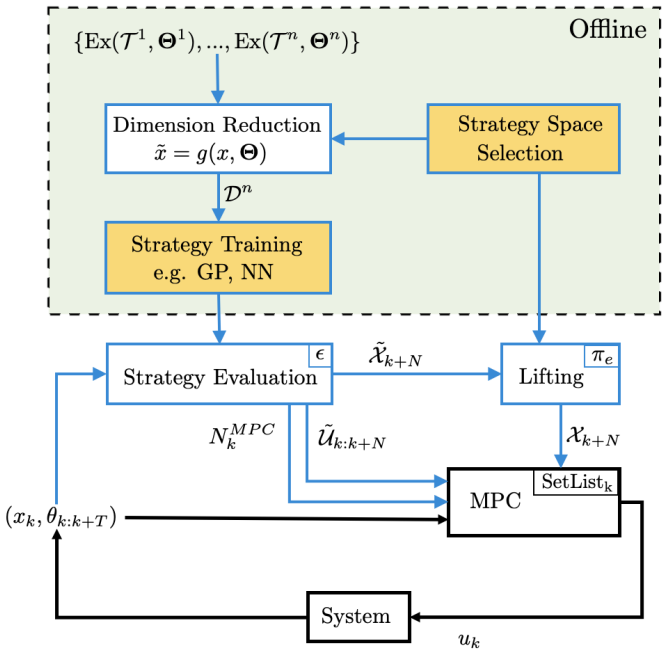


Figure 5.2: The Hierarchical Predictive Learning (HPL) control architecture. At time k, the state x_k and T-step environment forecast $\theta_{k:k+T}$ are used to evaluate the control strategy. A strategy consists of reduced dimensions sets, $\tilde{\mathcal{X}}_{k+N}$ and $\tilde{\mathcal{U}}_{k:k+N}$, towards which to steer the system in the next N time steps and input guidelines for getting there. These sets are used to construct a full-dimension target set, $\mathcal{X}_{k+N}$, used as a terminal set in an MPC controller with horizon N^{MPC}. At each time k, SetList_k determines the relationship between N and N^{MPC}. The low-level control loop is drawn in black, and shaded yellow blocks indicate control design choices.

from stored executions of previous control tasks. Second, we show how to integrate the learned strategies in an MPC framework so as to guarantee feasibility when solving a new control task in real-time.

Implementation

Hierarchical Predictive Learning (HPL) is a data-driven control scheme based on applying high-level strategies learned from previous executions of different tasks. The HPL controller modifies its behavior whenever new strategies become applicable, and operates in coordination with a safety controller to ensure constraint satisfaction at all future time steps.

An overview of the control architecture is shown in Fig. 5.2. Offline, before beginning the new control task, stored executions from previous tasks are used to train strategy functions of a desired parameterization (Sec. 5.4). After training, the controller can solve new tasks.

Online, at each time k, an T-step local environment forecast $\theta_{k:k+T}$ (5.4) is used to determine if a new high-level control strategy is available (Sec. 5.5). A strategy consists of state and input sets in reduced dimensions, $\tilde{\mathcal{X}}_{k+N}$ and $\tilde{\mathcal{U}}_{k:k+N}$, that provide a set towards which to steer the system in the next N timesteps, as well as input guidelines for getting there. The strategy sets are used to construct a target set in the full state space, $\mathcal{X}_{k+N}$. We note the difference between the environment forecast horizon T and the strategy prediction horizon N (Sec. 5.4). Lastly, an MPC controller with prediction horizon N^{MPC} calculates a low-level input u_k to reach the target set (Sec. 5.6).

There are five key design and control challenges:

1. representing the strategy mathematically,
2. choosing appropriate horizons N, T, and N^{MPC},
3. lifting the reduced-dimension strategy set into a full-dimension target set,
4. ensuring closed-loop strategy effectiveness at solving the new task, and
5. ensuring constraint satisfaction and recursive feasibility of the receding horizon control problem.

In the following sections, we address each of these challenges in detail and prove the feasibility of the resulting HPL control law. First, we show how to learn generalizable strategies from stored executions of previous tasks. Second, we show how to integrate the learned strategies in an MPC framework so as to guarantee feasibility when solving a new control task in real-time.

5.4 Learning Strategies From Data

This section addresses the first aim of our chapter: learning generalizable strategies from stored data of previously solved tasks. We consider strategies to be maps from a state and environment forecast to reduced-dimension strategy sets:

$$(\tilde{\mathcal{X}}_{k+N},\ \tilde{\mathcal{U}}_{k:k+N}) = \mathcal{S}(x_k,\ \theta_{k:k+T}), \tag{5.5}$$

where x_k is the system state at time k and $\theta_{k:k+T}$ the T-step environment forecast. $\tilde{\mathcal{X}}_{k+N}$ represents a strategy set in reduced dimension that the system should be in N timesteps into the future, and $\tilde{\mathcal{U}}_{k:k+N}$ provides constraint guidelines for a reduced dimension of the input as the system travels towards $\tilde{\mathcal{X}}_{k+N}$.

There are several ways of representing the strategy function $\mathcal{S}$ in (5.5), including model-based methods that use an explicit model for how variations in task environments affect the optimal control input [14]. In this work we instead opt for a data-driven approach, using stored executions (2.5) that solve related tasks.

Strategy States and Inputs

The strategy function outputs reduced-dimension state and input sets. We refer to the space where these reduced-dimensional sets lie as "strategy state space" and "strategy input space."

We define the strategy state at each time k as

$$\tilde{x}_k = g(x_k, \boldsymbol{\Theta}) \in \mathbb{R}^{n_{\tilde{x}}}, \tag{5.6}$$

where g maps the full-dimensional state x_k into the corresponding lower-dimensional strategy state $\tilde{x}_k$. Similarly, the strategy inputs are

$$\tilde{u}_k = r(u_k, \boldsymbol{\Theta}) \in \mathbb{R}^{n_{\tilde{u}}}, \tag{5.7}$$

where r maps the full-dimensional input at time k into lower-dimensional strategy inputs. These mapping functions may depend on the task's environment descriptor function $\boldsymbol{\Theta}$ - for example, a strategy state $\tilde{x}$ measuring the distance from a race track centerline depends on the shape of the centerline, which is described by $\boldsymbol{\Theta}$. We denote the strategy state and strategy input spaces as

$$\tilde{\mathcal{X}} = \{\tilde{x} \mid \tilde{x} = g(x, \boldsymbol{\Theta}),\ x \in \mathcal{X}\} \tag{5.8}$$

$$\tilde{\mathcal{U}} = \{\tilde{u} \mid \tilde{u} = r(u, \boldsymbol{\Theta}),\ u \in \mathcal{U}\}. \tag{5.9}$$

Remark 7. *The functions g and r may be any functions mapping $\mathcal{X}$ and $\mathcal{U}$ to a reduced-dimension space. They are not limited to indexing functions, i.e. we do not require $\tilde{x}_k$ to be a subset of $\mathcal{X}$.*

We use strategy states and inputs (rather than the full states and inputs directly) to allow for simplification and generalization. While all system states and inputs affect the system's trajectory via the dynamics (5.1), when solving complex tasks a subset of states and inputs are likely to be especially informative as to how the system should respond to the upcoming environment.

The choice of strategy states and inputs is therefore critical. Strategy states and inputs must be meaningful to solving the task at hand, and it must be reasonably expected that a strategy mapping (5.5) from an environment forecast to these strategy spaces can be formulated. If the chosen strategy states and inputs are not correlated with (i.e. can not be predicted from) the environment forecast, new strategy states must be chosen.

Initially, all system states and environment descriptions can be considered as candidate strategy states. Because reducing the number of strategy states can improve generalizability, this list should then be trimmed as much as possible (or as much as required for computational tractability). Human knowledge can provide insight into what states are likely to have an impact on optimal behavior, and therefore which strategy states and inputs to choose. In the autonomous racing task, for example, a control designer will know that it is easier to use the track curvature forecast to predict the vehicle's future distance along and from

the centerline than a future yaw rate. Note that strategy states should encode information about both task objective and task safety.

If no human intuition can be incorporated, data-driven methods could instead be used to find appropriate strategy states and inputs. Clustering methods or dimensionality-reduction methods such as PCA can estimate what states are most important for determining the strategy. Recent work [12], for example, proposes forming aggregate (or representative) features out of system states in order to reduce the problem dimension in Dynamic Programming - this approach could similarly be applied to finding strategy states for HPL. Using data-driven methods to determine the best choice of strategy states, particularly for complex tasks, is an avenue for future research.

In general, we suggest using intuition about the task to narrow the list of potential strategy states as much as possible, and using data-driven feature selection methods in the final stages if strategy performance using only hand-derived features is lacking. Sections 5.9-5.11 provide several examples for choosing strategy states for navigation tasks.

Remark 8. *It may also be beneficial to use the current strategy state $\tilde{x}_k$, rather than the current state x_k, as an input to the strategy function (5.5). This can further improve the generalizability of the learned strategy, as fewer values have to match between the new environment and the stored task data (2.5). The notation in the remainder of this chapter constructs the strategy input using the full state x_k, but the approach and theory remain the same if the strategy state space is used instead.*

Representing the Strategy

Thanks to an immense amount of machine learning research, there are myriad ways to represent the strategy function $\mathcal{S}$ in (5.5). We propose using Gaussian processes (GPs). GPs have frequently been used in recent predictive control literature to provide data-driven estimates of unknown nonlinear dynamics [50, 54, 60]. Specifically, GPs are used to approximate vector-valued functions with real (scalar) outputs.

Given training data (input vectors and output values), GPs use a similarity measure known as "kernel" between pairs of inputs to learn a nonlinear approximation of some true underlying input-output mapping $\psi : \mathbb{R}^n \to \mathbb{R}$. The kernel $k(x, x'|\theta)$ represents the learned covariance between two function evaluations x and x', and is parameterized by a set of hyperparameters θ. Once the hyperparameters of the kernel have been optimized, typically by maximizing the marginal likelihood of the training observations using gradient-based methods, the GP can be queried at a new input vector. Given a training data set $\mathbf{X} = [x^1 \dots x^M]^\top$ and $\mathbf{Y} = [y^1 \ \dots \ y^M]^\top$ corresponding to noisy evaluations of the unknown function as $y^i = \psi(x^i) + w$ where $w \sim \mathcal{N}(0, \sigma_n^2)$, the posterior distribution of $\psi(x)$ is given by a GP specified by

$$\mu(x|\mathbf{X}, \mathbf{Y}, \theta) = \mathbf{k}(x)^\top (\mathbf{K} + \sigma_n^2 I)^{-1} \mathbf{Y} \tag{5.10}$$

$$\sigma^2(x|\mathbf{X}, \mathbf{Y}, \theta) = k(x, x|\theta) - \mathbf{k}(x)^\top (\mathbf{K} + \sigma_n^2 I)^{-1} \mathbf{k}(x), \tag{5.11}$$

where

$$[\mathbf{K}]_{ij} = k(x^i, x^{(j)}|\theta)$$
$$\mathbf{k}(x) = [k(x, x^1|\theta), \ldots, k(x, x^M|\theta)]^\top.$$

As more training data becomes available, the posterior GPs approximate the unknown function $\psi(\cdot)$ more accurately provided that it belongs to the *Reproducing Kernel Hilbert Space* (RKHS) [89] induced by the kernel $k(\cdot, \cdot|\theta)$. When evaluated at a new input, the GP returns a Gaussian distribution over output estimates; thus GPs provide a best guess for the output value corresponding to an input (mean) and a measure of uncertainty about the estimate (variance). This allows us to gauge how confident the GP is in its prediction at a particular input, a critical component of the HPL framework. A review of GPs in control is provided in [61].

We note that the robust MPC community often prefers stochastic models with bounded support [21] to GPs, in order to have strict safety guarantees. Our approach can be extended to these types of models as well.

Training the Strategy (Offline)

We train GPs to predict the values of the strategy states (5.6) and inputs (5.7) at N timesteps into the future, based on the current state and T-step environment forecast. Each GP approximates the mapping to one strategy state or input:

$$\mu(\tilde{x}), \sigma^2(\tilde{x}) = \text{GP}(x_k, \theta^i_{k:k+T}), \tag{5.12}$$

where μ and σ^2 represent statistics of the Gaussian distribution over strategy state estimates. GPs best approximate functions with scalar outputs, so we train one GP for each strategy state and input (a total of $n_{\tilde{x}} + n_{\tilde{u}}$ number of GPs). The learned GPs capture high-level strategies that were common to a variety of previously solved, related tasks. We note that the GPs are evaluated only at the current state and environment forecast—they are time-invariant and do not depend on the new task beyond the environment forecast.

We use the stored executions (2.5) from previous control tasks to create GP training data. The training output data for each GP contains the strategy state or input the GP is learning to predict. After solving n control tasks, the training data consists of:

$$\begin{aligned} \mathcal{D} = \{\boldsymbol{z} &= [z^1_0, z^1_1, \ldots, z^1_{D^1-N}, z^2_0, \ldots, z^n_{D^n-N}]^\top \\ \boldsymbol{y} &= [y^1_0, y^1_1, \ldots, y^n_{D^n-N}]^\top. \end{aligned} \tag{5.13}$$

Each input vector z^i_k corresponds to the output y^i_k, where

$$z^i_k = [(x^i_k)^\top, (\theta^i_k)^\top, (\theta^i_{k+1})^\top, ..., (\theta^i_{k+T})^\top],\ i \in [1, n],\ k \in [0, D^i - N], \tag{5.14}$$

and θ^i_k denotes the local environment at time k of the ith control task. We note that in (5.14), each GP uses the same training input data, but this need not be the case.

The corresponding output entry y_k^i contains the value of the strategy state of interest at N time steps in the future:

$$y_j^i = (\tilde{x}_{k+N}^i)^\top,\ i \in [1, n],\ k \in [0, D^i - N].$$

The input strategy set can be similarly parameterized in a number of different ways, such as minimum and maximum values realized over the N-step trajectory.

Once the training data (5.14) is collected, the GP kernel hyperparameters are optimized using maximum log-likelihood regression. In this chapter, we use the squared-exponential kernel, though different kernels can be chosen depending on the expected form of the task-specific strategy equation (5.5). Given two entries of $\boldsymbol{z}$ in (5.13), the squared-exponential kernel evaluates as

$$k(z_i^o, z_j^w) = \sigma_f^2 \exp -\frac{1}{2} \sum_{m=1}^{n_x+T+1} \frac{(z_i^o(m) - z_j^w(m))^2}{\sigma_m^2},$$

where $z_i^o(m)$ is the mth entry of the vector z_i^o. Many software packages exist that automate Bayesian optimization of the hyperparameters, including the Machine Learning Toolbox in Matlab and SciKit Learn or GPyTorch in Python.

Evaluating the Strategy (Online)

Once trained, the GPs can be evaluated on data from a new task. At time k of a new task $\mathcal{T}^{n+1}$, we evaluate the GPs at the new query vector z_k^{n+1}, formed as in (5.14), to construct hyperrectangular strategy sets in reduced-dimension space.

Each GP returns a one-dimensional Gaussian distribution over output scalars, parameterized by a mean μ and variance σ^2. Specifically, these are evaluated as:

$$\mu(z_k^{n+1}) = \boldsymbol{k}(z_k^{n+1})\bar{K}^{-1}\boldsymbol{y} \tag{5.15}$$

$$\sigma(z_k^{n+1})^2 = k(z_k^{n+1}, z_k^{n+1}) - \boldsymbol{k}(z_k^{n+1})\bar{K}^{-1}\boldsymbol{k}^\top(z_k^{n+1}), \tag{5.16}$$

where

$$\boldsymbol{k}(z_k^{n+1}) = [k(z_k^{n+1}, z_0^1), \ldots, k(z_k^{n+1}, z_{D^n-N}^n)], \tag{5.17}$$

and the matrix $\bar{K}$ is formed out of the covariances between training data samples such that

$$\bar{K}_{i,j} = k(\boldsymbol{z}_i, \boldsymbol{z}_j). \tag{5.18}$$

Given means and variances, we form one-dimensional bounds on each ith strategy state and jth strategy input as

$$\tilde{\mathcal{X}}_{k+N}(i) = [\mu^i(z_k^{n+1}) \pm \eta\sigma^i(z_k^{n+1})],\ \forall i \in [1, n_{\tilde{x}}] \tag{5.19}$$

$$\tilde{\mathcal{U}}_{k:k+N}(j) = [\mu^j(z_k^{n+1}) \pm \eta\sigma^j(z_k^{n+1})], \quad \forall j \in [n_{\tilde{x}} + 1, n_{\tilde{x}} + n_{\tilde{u}}].$$

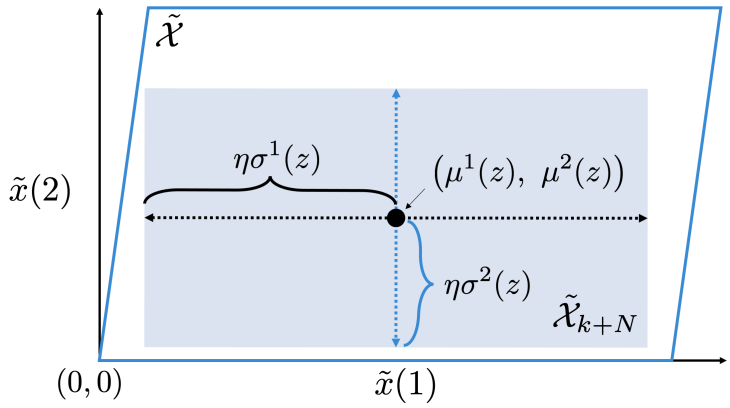


Figure 5.3: Each dimension $\tilde{x}(i)$ of the strategy set $\tilde{\mathcal{X}}_{k+N}$ is bounded using the mean and variance of a GP evaluation (5.19).

In (5.19), $\mu^i(z_k^{n+1})$ and $\sigma^i(z_k^{n+1})$ are the means and standard deviations computed by the ith GP evaluated at z_k^{n+1}. The parameter $\eta > 0$ determines the size of the range. When these one-dimensional bounds are combined for all strategy states and strategy inputs, hyperrectangular strategy sets are formed in strategy space, with each dimension constrained according to (5.19). An example is shown in Fig. 5.3. The hyperrectangular strategy sets are denoted $\tilde{\mathcal{X}}_{k+N}$ and $\tilde{\mathcal{U}}_{k:k+N}$, and indicate where (in strategy space) the system should be in N timesteps and what inputs to apply to get there.

Other Strategy Models

There are many benefits to using GPs to model the strategy function (5.5), including the interpretability of the output variance as a level of strategy confidence. One downside to using GPs is that they are very computationally inefficient to train: the hyperparameter optimization time scales as the cube of the training data size. This forces a trade-off between computational complexity and the effectiveness of the learned strategy, which benefits from seeing as much training data as possible. It also prevents easy online fine-tuning of the learned strategy as more task data becomes available.

If training complexity becomes too cumbersome, the strategy function (5.5) may be replaced by a different data-driven model that best fits the requirements of the considered task. In addition to computational complexity, there are two main considerations for choosing a strategy model for HPL:

1. the model should be able to approximate our best reasonable guess for the true shape of the underlying function (e.g. bounded, continuous, etc.), and

2. the model should have a way of estimating the confidence in a particular strategy evaluation.

Any data-driven model that appropriately satisfies these two requirements may be used in place of the GP without affecting the theory presented in this chapter.

With regard to the first consideration, using universal function approximators such as neural networks is easy and appealing, as they can be composed to represent functions of most shapes (at the cost of more hyperparameters to optimize). However, estimating the confidence associated with a particular strategy prediction is much more difficult for neural networks than for GPs [81]. Common approaches include bootstrapping [85] or training additional network layers to estimate the confidence [59]. These approaches are often tuned to the considered task, and can be poorly calibrated [86]. Various efforts to improve confidence estimation in neural networks have been made, but these methods often increase the size of the learned network, which worsens computational complexity [127, 30]. A promising new avenue is recent work in model-based neural networks [97], but further validation is required.

Strategy Horizons

The strategy mapping uses two different horizons: the environment forecast horizon T and the strategy prediction horizon N. T determines how much information about the future environment the strategy takes into account, while N determines how far out the strategy predicts the values of strategy states and inputs. The two need not be the same. In fact, for many navigation tasks it makes sense to choose $T > N$.

In the autonomous racing example, $T > N$ corresponds to looking further ahead along the track curvature and then only planning a trajectory along the first part of the visible track. This choice means that the strategy can suggest a strategy set to aim for that is also likely to be feasible in and optimal for the immediate future, since the upcoming environmental constraints were already implicitly considered. Choosing $N = T$, in contrast, means that the strategy needs to predict a strategy set as far out as the environment is known. If the environment changes immediately beyond the horizon T, the proposed strategy set $\tilde{X}_{k+N}$ may have been a poor choice.

As with choosing strategy states, human intuition can also play a role in choosing appropriate strategy horizons. Alternatively, the collected executions (2.5) from previous tasks can be examined to determine appropriate values for T and N, by evaluating how great environmental differences must be before the system's trajectory changes. This can typically be estimated from the available task data, and fine-tuned as necessary using cross-validation. In general, it is wise to use the largest computationally-allowable T.

Sections 5.9-5.11 provide several examples for choosing strategy horizons.

Remark 9. *As described thus far, both horizons N and T are time horizons, i.e. they forecast the environment and predict the strategy state at certain numbers of time steps in the future. It is also possible to, instead, forecast the environment and strategy state at a*

fixed distance into the future, or use a mix of time- and space-forecasting for N and T. In the racing example, this could correspond to always seeing a fixed number of meters ahead, no matter the vehicle speed. This approach is considered in Sec. 5.10.

5.5 Safely Applying Learned Strategies

We now address the second aim of our chapter: using the strategy sets in a low-level controller while maintaining safety guarantees. Our approach consists of *i)* lifting the reduced-dimension strategy sets (5.19) back into the full-dimensional state space, and *ii)* integrating the lifted strategy set with a safety controller. The result is a target set that can be used in a low-level MPC controller.

Assumption 10. *There exists a safety control policy that can prevent the system (5.1) from violating both system- and task-specific environment constraints (5.3). In particular, there exists a safe set*

$$\mathcal{X}_E \subseteq \mathcal{X}(\boldsymbol{\Theta}), \tag{5.20}$$

and a corresponding safety control policy

$$u = \pi_{\mathrm{e}}(x, \boldsymbol{\Theta}), \tag{5.21}$$

such that $\forall x \in \mathcal{X}_E,\ f(x, \pi_{\mathrm{e}}(x, \boldsymbol{\Theta})) \in \mathcal{X}_E$.

Remark 10. *Given a safety controller (5.21), a safe set (5.20) may be found using a variety of data-driven methods, such as sample-based forward reachability from a gridded state space $\mathcal{X}$ [23, 132, 25].*

Lifting Strategy Sets to Full-Dimensional Target Sets

At each time k of solving a task $\mathcal{T}^{n+1}$, new reduced-dimension strategy sets are constructed according to (5.19). These strategy sets must be lifted to target sets in the full-dimensional state space so they can be used as a terminal constraint in a low-level MPC controller. Critically, the target set must belong to the safe set (5.20). This ensures that once the system has reached the target set, there will always exist at least one feasible input (the safety control (5.21)) that allows the system to satisfy all state constraints. Given strategy sets (5.19), we find a corresponding lifted strategy set:

$$\mathcal{X}_{k+N} = \{x \in \mathcal{X}_E \mid g(x, \boldsymbol{\Theta}) \in \tilde{\mathcal{X}}_{k+N}\}, \tag{5.22}$$

where $g(x, \boldsymbol{\Theta})$ is the projection of the full-dimensional state x onto the set of chosen strategy states, as in (5.6). $\mathcal{X}_{k+N}$ is a full-dimensional set in which the strategy states (5.6) lie in the GP's strategy sets (5.19) and the remaining states are in the safety set. Thus for any state $x_k \in \mathcal{X}_{k+N}$, the safety control (5.21) can be applied if necessary to ensure constraint satisfaction in future time steps. Figure 5.4 depicts the difference between a strategy set and its lifted strategy set.

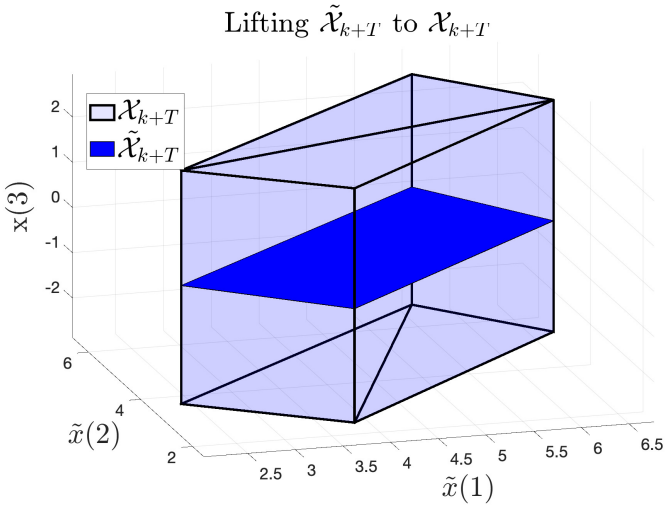


Figure 5.4: In the lifted strategy set (5.22), the strategy states $\tilde{x}(1)$ and $\tilde{x}(2)$ are constrained to lie in the strategy set, with additional states like $x(3)$ constrained according to $\mathcal{X}_E$.

Tuning Risk

If desired, control designers can specify a maximum risk level $\beta \in [0, 1]$ to control how conservative the target set $\mathcal{X}_{k+N}$ is with regards to incorporating the safety control. The above formulation (5.22) for the terminal set corresponds to *no* risk (i.e. $\beta = 0$), since for all states in $\mathcal{X}_{k+N}$ there exists at least one feasible input sequence (the safety control) that will result in a safe closed-loop state evolution.

If this is too restrictive, the target set can be chosen to be a convex combination of the safety set $\mathcal{X}_E$ and the environmental state constraint set:

$$\mathcal{X}_{k+N} = \{x \in \beta\mathcal{X}_E + (1-\beta)\mathcal{X}(\boldsymbol{\Theta}) \mid g(x, \boldsymbol{\Theta}) \in \tilde{\mathcal{X}}_{k+N}\},$$

By varying β in the range between 0 and 1, we vary how conservative our approach needs to be, depending on the cost of task failure. As the value of β increases, the target set converges to the state constraints imposed on the system by the task environment ($\beta = 1$).

An example of this interpolation is shown in Fig. 5.5, which depicts how the constraints for the third entry of the system state $x(3)$ vary across different points in the strategy set (in the $\tilde{x}(1) - \tilde{x}(2)$ plane) and as β varies. We see that as β ranges from 0 to 1 the size of the non-strategy state constraint sets increases. When $\beta = 0$, plotted in dark blue, the target set is as in Fig. 5.4.

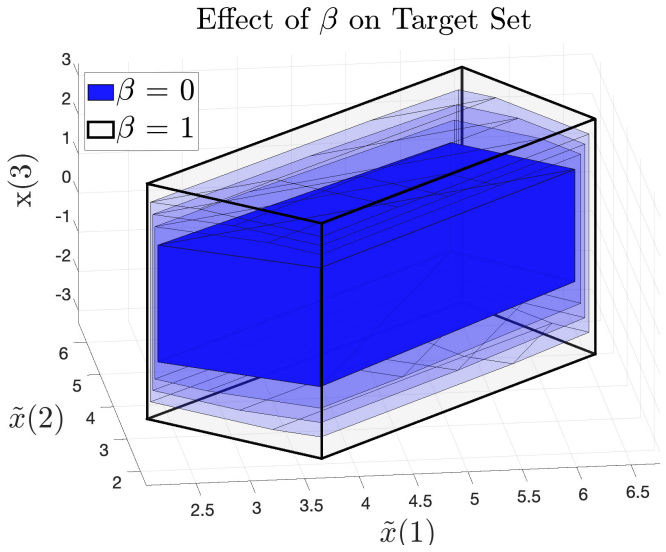


Figure 5.5: As β varies, the constraints on the non-strategy state $x(3)$ are imposed either through $\mathcal{X}_E$ (if $\beta = 0$), $\mathcal{X}(\theta)$ (if $\beta = 1$), or a combination of both (if $0 < \beta < 1$).

Incorporating the Uncertainty Measure

A benefit of using GPs to represent the strategy is that the standard deviation around an estimate may be used to evaluate how confident the GP is in its prediction at a particular input. At time k, consider a vector $\mathcal{C}_k$ containing the standard deviations of the evaluated GPs:

$$\mathcal{C}_k = [\sigma^1(z_k^{n+1}), \ldots, \sigma^{n_{sx}+n_{su}}(z_k^{n+1})]. \tag{5.23}$$

If the GPs return a strategy set with standard deviations larger than a chosen threshold d_{thresh}, we may opt not to use this strategy. We expect $\mathcal{C}_k > d_{\text{thresh}}$ if either

1. the system did not encounter a similar environment forecast in a previous control task (training data), or

2. in previous control tasks this environment forecast did not lead to a single coherent strategy, resulting in a wide distribution of potential future strategy states.

With high uncertainty measures, the strategy sets are not likely to contain valuable control information for the system. In this case, the target set is set to be *empty*: $\mathcal{X}_{k+N} = [\,]$. The next section 5.6 explains that this results in a horizon shift for the low-level MPC, and the system (5.1) re-uses the target set from the previous time step.

5.6 Low-level Controller Design

The low-level MPC controller calculates the input to be applied to the system at each time k. This input is calculated based on the sequence of target sets (5.22) found during the last N timesteps.

Target Set List

At each time k of solving task $\mathcal{T}^{n+1}$, a new target set (5.22) is constructed by lifting the strategy sets (5.19). However, if the standard deviations (5.23) are too high, or there is no feasible input sequence to reach the target set $\mathcal{X}_{k+N}$, the target set for time k will be empty: $\mathcal{X}_{k+N} = [\,]$.

The target set list keeps track of the target sets (empty or not) which were constructed during the most recent T timesteps:

$$\text{SetList}_k = [\mathcal{X}_{k+1}, \mathcal{X}_{k+2}, \ldots, \mathcal{X}_{k+N}]. \tag{5.24}$$

At each new time step, the first set is removed and the target set found at the current time step k is appended to the end. In this way, the target set list (5.24) always maintains exactly N sets, though some (including the last set $\mathcal{X}_{k+N}$) may be empty. This list is used to guide the objective function and constraints of the MPC controller.

Shifting Horizon MPC Formulation

We formulate an MPC controller to calculate our input at each time step:

$$\begin{aligned}
\mathbf{u}^\star(x_k) = \underset{u_{k|k},\ldots,u_{k+N_k^{MPC}-1|k}}{\arg\min} \quad & \sum_{j\in\mathcal{S}_k}^{N_k^{MPC}-1} \text{dist}\big(x_{j|k}, \mathcal{X}_{k+j}\big) \\
\text{s.t.} \quad & x_{k+t+1|k} = f(x_{k+t|k}, u_{k+t|k}) \\
& u_{k+t|k} \in \mathcal{U}\ \forall t \in \{0,\ldots,N-1\} \\
& x_{k+t|k} \in \mathcal{X}(\boldsymbol{\Theta}^{n+1})\ \forall t \in \{0,\ldots,N\} \\
& x_{k|k} = x_k \\
& x_{k+N_k^{MPC}|k} \in \mathcal{X}_{k+N_k^{MPC}},
\end{aligned} \tag{5.25}$$

where $\mathcal{S}_k$ is the set of indices with non-empty target sets,

$$\mathcal{S}_k = \{s \mid \text{notEmpty}(\mathcal{X}_{k+s-1})\}.$$

The MPC objective function (5.25) penalizes the Euclidean distance from each predicted state to the target set corresponding to that prediction time. For a smoother cost, the objective could be augmented to take the input effort into account.

The MPC uses a time-varying shifting horizon $0 < N_k^{MPC} \leq N$ that corresponds to the largest time step into the future for which a non-empty target set results in feasibility of (5.25):

$$N_k^{MPC} = \max s \ : \ \{s \in \mathcal{S}_k, \ (5.25) \text{ is feasible with } N_k^{MPC} = s\}. \tag{5.26}$$

This ensures that the MPC controller (5.25) has a non-empty terminal constraint and the optimization problem is feasible. To avoid unnecessary repeated computations, all target sets in the target set list (5.24) which lead to infeasibility of (5.25) when used as the terminal constraint are set as *empty* in (5.24). At time step k, we apply the first optimal input to the system:

$$u_k = u_{0|k}^{\star}. \tag{5.27}$$

Remark 11. *Here, new target sets (5.22) are found at the same frequency as the controller update (5.25)-(5.27), but this can easily be adapted for asynchronous loops as in [102].*

Remark 12. *Evaluating GPs to calculate the strategy is fast when optimized, and constructing strategy sets according to (5.22) is straightforward once GPs have been evaluated. Incorporating strategies will thus not have significant impacts on the closed-loop run time of the MPC controller (5.25)-(5.27).*

Safety Control

If no target sets in (5.24) can feasibly be used as a terminal constraint in (5.25), all sets in the target set list will be empty, and the MPC horizon is $N_k^{MPC} = 0$. When this occurs, the system enters into *Safety Control mode*. The safety controller (5.21) controls the system until a time when a satisfactory target set is found (at which point the MPC horizon resets to $N_k^{MPC} = N$). The HPL algorithm in Sec. 5.7 ensures that whenever $N_k^{MPC} = 0$, the system will be in the safe set (5.20) and the safety controller may be used. We prove this in Sec. 5.8.

5.7 The HPL Algorithm

Alg. 4 summarizes the HPL control policy. Gaussian processes, trained offline on trajectories from past control tasks, are used online to construct reduced-dimension strategy sets based on new environment forecasts. Target sets, computed by intersecting lifted strategy sets with the safety set, are used as terminal sets in a shifting-horizon MPC.

5.8 Properties of HPL Conrol

We prove that Alg. 4 outputs a feasible execution for a new control task $\mathcal{T}^{n+1}$.

Algorithm 4 HPL Control Policy

parameters: $d_{\text{thresh}}, T, N, \mathcal{X}_E, \pi_E$
input: f, $\mathcal{X}$, $\mathcal{U}$, $\{\text{Ex}(\mathcal{T}^1, \mathbf{\Theta}^1), ..., \text{Ex}(\mathcal{T}^n, \mathbf{\Theta}^n)\}$, $\mathbf{\Theta}^{n+1}$
output: $\text{Ex}(\mathcal{T}^{n+1}, \mathbf{\Theta}^{n+1})$

offline:
train GPs using stored executions as in Sec.5.4

online:
initialize $k = 0$, $N_k^{MPC} = N$, SetList = [ ]
for each time step k **do**
  collect $(x_k, \theta_{k:k+T}^{n+1})$
  find $[\tilde{\mathcal{X}}_{k+N}, \tilde{\mathcal{U}}_{k:k+N}, \mathcal{C}_k]$ (5.12) - (5.19)
  if $\mathcal{C}_k < d_{\text{thresh}}$ **then**
    $\mathcal{X}_{k+N} = [\,]$
  else
    construct $\mathcal{X}_{k+N}$ (5.22)
  append $\mathcal{X}_{k+N}$ to SetList and shift sets
  if all sets in SetList are **empty then**
    $u_k = \pi_e(x_k, \mathbf{\Theta}^{n+1})$
  else
    calculate N_k^{MPC} (5.26)
    solve MPC with horizon N_k^{MPC}
    $u_k = u_{0|k}^\star$ (5.27)

Theorem 6. *Let Assumption 1 hold. Consider the availability of feasible executions (2.5) by a constrained system (5.1)-(5.2) of a series of control tasks* $\{\mathcal{T}^1, \ldots, \mathcal{T}^n\}$ *in different environments* $\{E(\mathbf{\Theta}^1), \ldots, E(\mathbf{\Theta}^n)\}$. *Consider a new control task* $\mathcal{T}^{n+1}$ *in a new environment* $E(\mathbf{\Theta}^{n+1})$. *If* $x_0^{n+1} \in \mathcal{X}_E$, *then the output of Alg. 4 is a feasible execution of* $\mathcal{T}^{n+1}$*:* $\text{Ex}(\mathcal{T}^{n+1}, \mathbf{\Theta}^{n+1})$.

The complete proof is detailed in Sec. A.4. The key idea is that by using the safety controller to lift reduced-dimensional sets to full-dimensional target sets, the system always plans trajectories that end in the domain of the safety controller. Thus, any time the strategy proposes an infeasible terminal set for N consecutive time steps, the safety controller may be applied until a feasible strategy is found. We use recursion to demonstrate that this results in a feasible execution of the new task.

5.9 Application 1: Robotic Manipulator Navigation

We evaluate HPL in the robotic path planning example introduced in Chapter 2.

Here we again consider control of a UR5e[2] robotic arm. The UR5e has high end-effector reference tracking accuracy, allowing us to use a simplified end-effector model in place of a discretized second-order model [110]. At each time step k, the state of the system is x_k,

$$x_k = [q_{0_k},\ \dot{q}_{0_k},\ y_k,\ \dot{y}_k],$$

where q_{0_k} is the robot's first horizontal joint angle, y_k the height of the end-effector, and $\dot{q}_{0_k}$ and $\dot{y}_k$ their respective velocities (see Fig. 2.5). The inputs to the system are

$$u_k = [\ddot{q}_{0_k},\ \ddot{y}_k], \tag{5.28}$$

the accelerations of the end-effector in the q_0 and y direction, respectively.

We model the base-and-end-effector system as a quadruple integrator:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ A &= \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 \\ dt & 0 \\ 0 & 0 \\ 0 & dt \end{bmatrix}, \end{aligned} \tag{5.29}$$

where $dt = 0.01$ seconds is the sampling time. The system state and input constraints are

$$\begin{aligned} \mathcal{X} &= \left\{ x : \begin{bmatrix} -\pi \text{ rad/s} \\ \dot{y}_{\min} \text{ m/s} \end{bmatrix} \leq \begin{bmatrix} \dot{q}_{0_k} \\ \dot{y}_k \end{bmatrix} \leq \begin{bmatrix} \pi \text{ rad/s} \\ \dot{y}_{\max} \text{ m/s} \end{bmatrix} \right\} \\ \mathcal{U} &= \left\{ u : \begin{bmatrix} -\pi \text{ rad/s}^2 \\ \ddot{y}_{\min} \text{ m/s}^2 \end{bmatrix} \leq \begin{bmatrix} \ddot{q}_{0_k} \\ \ddot{y}_k \end{bmatrix} \leq \begin{bmatrix} \pi \text{ rad/s}^2 \\ \ddot{y}_{\max} \text{ m/s}^2 \end{bmatrix} \right\}, \end{aligned}$$

where

$$\dot{y}_{\max,k} = C_1 \sin\left(\arccos\left(\frac{y_k}{d_1} \right) \right) \tag{5.30}$$

$$\dot{y}_{\min,k} = -\dot{y}_{\max,k} \tag{5.31}$$

$$\ddot{y}_{\max,k} = C_2 \sin\left(\arccos\left(\frac{y_k}{d_2} \right) + \frac{y_k}{d_3} \right) \tag{5.32}$$

$$\ddot{y}_{\min,k} = -\ddot{y}_{\max,k}, \tag{5.33}$$

with C_1, C_2, d_1, d_2 and d_3 dependent on setup parameters and joint limits provided by the manufacturer. This model accurately represents the system dynamics as long as we

[2] https://www.universal-robots.com/products/ur5-robot/

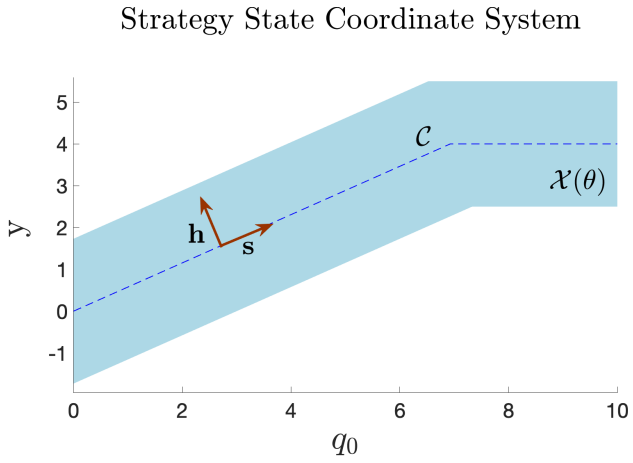


Figure 5.6: The end-effector is constrained to stay in the light blue tube $\mathcal{X}(\boldsymbol{\Theta})$. The strategy states measure the cumulative distance along and the distance from the centerline.

operate within the experimentally characterized region of high end-effector reference tracking accuracy (see Chapter 2 for details).

Note that the states q_{0_k} and y_k are not constrained by the system, but by a particular task environment. Each control task $\mathcal{T}^i$ requires the end-effector to be controlled through a different tube, described using the environment descriptor function $\boldsymbol{\Theta}^i$, as quickly as possible. Here, the function $\boldsymbol{\Theta}^i$ maps a state in a tube segment to the slope of the constant-width tube. Different control tasks $\{\mathcal{T}^1, \ldots, \mathcal{T}^n\}$ correspond to maneuvering through tubes of constant width but different piecewise-constant slopes.

We choose two strategy states for these tasks:

$$\tilde{x}_k = [s_k,\ h_k],$$

where s_k is the cumulative distance along the centerline of the tube from the current point (q_{0_k}, y_k) to the projection onto the centerline, and h_k is the distance from (q_{0_k}, y_k) to the centerline. The strategy states therefore measure the total distance traveled along the tube up to time step k, and the current signed distance from the center of the tube. These strategy states were chosen because they provide information about both task performance (distance traveled along the tube is a measure of task completion speed) and constraint satisfaction (distance from the tube boundaries). Note that in this task, just as in car racing, cutting corners (as measured by h_k) also maximizes distance traveled along the tube.

The system inputs (5.28) are used as strategy inputs.

The safety controller (5.21) is an MPC controller which tracks the centerline of the tube at a slow, constant velocity of 0.5 meters per second. The safe set $\mathcal{X}_E$ (5.20) is determined offline using sampling-based forward reachability.

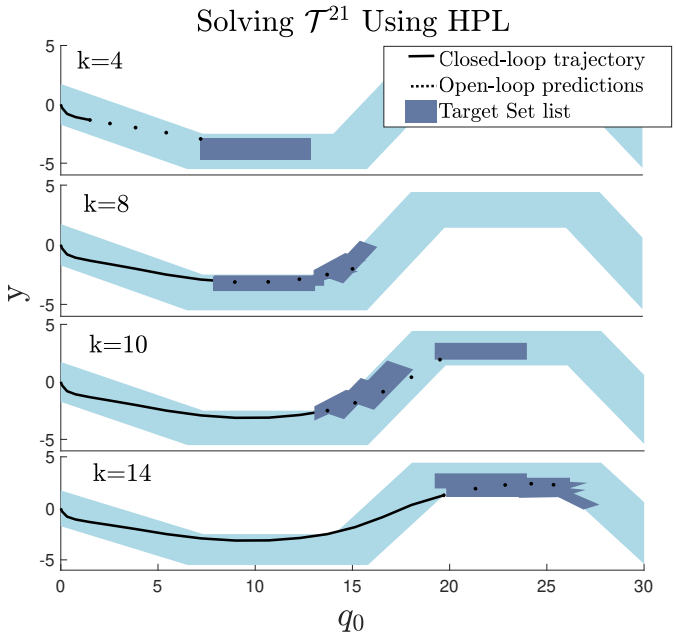


Figure 5.7: At each time step, the target set list (5.24) provides different regions in the task space for the system to track.

Hierarchical Predictive Learning Results

We test the effectiveness of the HPL control architecture (Alg. 4) in simulation. We begin by collecting executions that solve a series of 20 control tasks $\{\mathcal{T}^1, ..., \mathcal{T}^{20}\}$, with each control task corresponding to a different tube shape. The executions are closed-loop trajectories completed by a Learning Model Predictive Controller (LMPC). This reference-free iterative learning controller is initialized with a conservative, feasible trajectory and then improves its closed-loop performance at each iteration of a task. For each control task $\mathcal{T}^i$, we find an initial (suboptimal) execution using the centerline-tracking MPC safety controller. This execution is used to initialize the LMPC, which then runs for five iterations on the task $\mathcal{T}^i$. At each iteration, the LMPC uses the trajectory data from the previous iteration to improve the closed-loop performance at the current iteration with respect to a chosen cost function (in our instance, time required to reach the end of the tube). The execution corresponding to the fifth LMPC iteration of each task $\mathcal{T}^i$ is added to our training data set.

We use an environment forecast horizon of $T = 10$ seconds and a control horizon of $N = 5$ seconds. These horizons were chosen based on the approximate required time to traverse an average tube segment in the previous tasks. The forecast horizon was chosen to be twice the control horizon in order to provide information about both the current and subsequent tube segments. GPs using the squared-exponential kernel are then optimized to approximate the strategies learned from solving the 20 different control tasks. Matlab 2018a was used for all

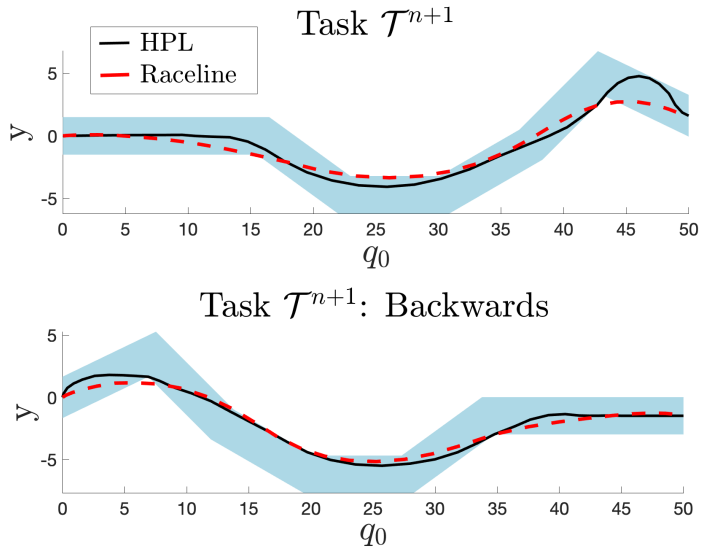


Figure 5.8: The HPL execution is compared to the raceline (the fastest possible execution), as determined by an LMPC [100]. Respective execution times in [s] are 6.5 (LMPC), 8.8 (HPL), and 12.8 (Centerline-tracking π_e).

data collection and GP training.

Figure 5.7 shows the closed-loop trajectory and target set list at various time steps of solving a new task, $\mathcal{T}^{21}$, using the HPL framework. At each time step, the final predicted state lies within the last set in the target set list; the other predicted states track any non-empty target sets as closely as possible. The formulation allows us to visualize what strategies have been learned, by plotting at each time step where the system thinks it should go. Indeed, we see that the system has learned to maneuver along the insides of curves, and even takes the direct route between two curves going in opposite directions.

Figure 5.8 shows the resulting executions for a new task solved forwards and backwards. We emphasize that HPL generalizes the strategies learned from training data to unseen tube segments. Specifically, for the tasks shown here the GPs were trained on executions solving tasks in the forward direction, i.e. constructed left to right using tube segments as shown in the top images. For example, tube segments of certain slopes had only been traversed upwards in previous control tasks, never down. The HPL control architecture was able to handle this change very well. As in Fig. 5.7, the forward and backwards trajectories demonstrate good maneuvering strategies, including moving along the insides of curves and cutting consecutive corners. In fact, the HPL controller finds an execution that is very close to the minimum time execution as determined by an iterative learning MPC controller [99].

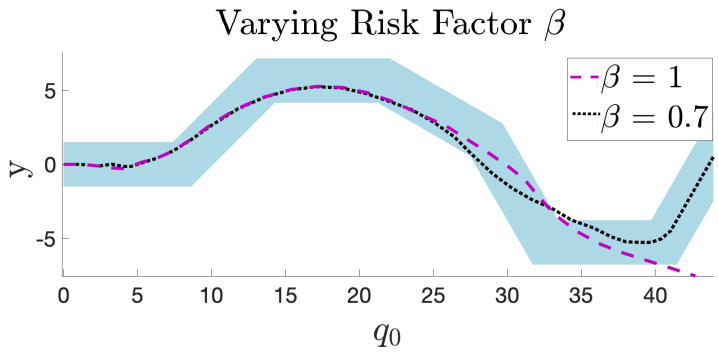


Figure 5.9: The same task is performed using two different values of β. For this task, using $\beta = 1$ led to eventual constraint violation because the terminal constraints were not formed in conjunction with the safety controller.

Implementation details

Section 5.5 introduced an approach for tuning the conservativeness with which the strategy sets (5.19) are lifted into target sets (5.22), by varying the importance of intersecting with the safety set $\mathcal{X}_E$ (5.20). To examine the effects of varying the acceptable risk level β on the system's closed-loop trajectory, we solve a single task using $\beta = 1$ and a more conservative $\beta = 0.7$. The results are depicted in Fig. 5.9, and demonstrate the benefits of incorporating the safety set $\mathcal{X}_E$ during the formation of full-dimensional target sets.

Both trajectories ($\beta = 0.7$ and $\beta = 1$) follow the same path at the beginning of the task. However, during a sharp curve towards the latter portion of the task, the trajectory corresponding to $\beta = 1$, plotted in purple, is no longer able to satisfy the task constraints.

The more conservative approach, using $\beta = 0.7$ and plotted in black, takes the safety constraints $\mathcal{X}_E$ into account and is able to maneuver the sharp turn. We see that the center-tracking safety controller activates at the end of the task and keeps the system within the constraints (whereas the $\beta = 1$ trajectory leaves the task constraints at this point).

5.10 Application 2: Formula 1 Racing

We evaluate HPL control in an autonomous racing task, where a small RC car is controlled as quickly as possible around 1/10 scale Formula 1 race tracks.

System and Task Description

Consider a small RC car[3] whose dynamics are modeled in the curvilinear abscissa reference frame [94] with a nonlinear Pacejka tire model, with states and inputs at time step k:

$$\begin{aligned} x_k &= [v_{x_k}\ v_{y_k}\ \dot{\psi}_k\ e_{\psi_k}\ s_k\ e_{y_k}]^\top \\ u_k &= [a_k\ \delta_k]^\top, \end{aligned} \tag{5.34}$$

where v_{x_k}, v_{y_k}, and $\dot{\psi}_k$ are the vehicle's longitudinal velocity, lateral velocity, and yaw rate, respectively, at time step k, s_k is the distance traveled along the centerline of the road, and e_{ψ_k} and e_{y_k} are the heading angle and lateral distance error between the vehicle and the path. The inputs are longitudinal acceleration a_k and steering angle δ_k.

The system dynamics are described using an Euler discretized dynamic bicycle model [101]. The vehicle is subject to system-imposed state and input constraints given by

$$\mathcal{X} = \left\{ x : \begin{bmatrix} 0 \\ -10\ \text{m/s} \\ -\frac{\pi}{2}\ \text{rad} \\ -\frac{\pi}{3}\ \text{rad} \\ -\frac{l}{2}\ \text{m} \end{bmatrix} \leq \begin{bmatrix} v_x \\ v_y \\ w_z \\ e_\psi \\ e_y \end{bmatrix} \leq \begin{bmatrix} 10\ \text{m/s} \\ 10\ \text{m/s} \\ \frac{\pi}{2}\ \text{rad} \\ \frac{\pi}{3}\ \text{rad} \\ \frac{l}{2}\ \text{m} \end{bmatrix} \right\} \tag{5.35}$$

$$\mathcal{U} = \left\{ u : \begin{bmatrix} -1\ \text{m/s}^2 \\ -0.5\ \text{rad/s}^2 \end{bmatrix} \leq \begin{bmatrix} a \\ \delta \end{bmatrix} \leq \begin{bmatrix} 1\ \text{m/s}^2 \\ 0.5\ \text{rad/s}^2 \end{bmatrix} \right\},$$

where $l = 0.8$ is the track's lane width.

The car's task is to drive around a race track as quickly as possible while satisfying all system and environmental state and input constraints. Each control task $\mathcal{T}^i$ corresponds to a new track, described using the environment descriptor function $\mathbf{\Theta}^i$ which maps the current position along the i-th track to a description of the upcoming track curvature. For training and testing, we use scaled Formula 1 tracks whose geospatial coordinates were made publicly available in [9].

We choose two similar strategy states for the racing task as for the robot navigation task:

$$\tilde{x}_k = [\Delta s_k,\ e_{y_k}], \tag{5.36}$$

where $\Delta s_k = s_k - s_{k-T}$ measures the distance traveled along the track centerline in the last N timesteps.

The safety controller is an MPC controller which tracks the centerline of the race track at a constant velocity of 5 meters per second. The safe set $\mathcal{X}_E$ induced by this safety controller is estimated from Monte Carlo simulations.

[3] `http://www.barc-project.com/`

Hierarchical Predictive Learning Results

We construct ten different 1/10-scale Formula 1 tracks from [9], and then calculate the minimum-time trajectory for each track using a continuous-time vehicle model. The discretized racelines from seven tracks are used to create training data for our strategy, while the remaining three tracks are used for evaluating the performance of the HPL control algorithm.

GPs using the squared-exponential kernel are trained on these racelines for each of the two strategy states. As described in Sec. 5.4, the environment forecast does not have to be parameterized by time. Indeed, for the racing task we use an environment forecast parameterized by a constant distance along the track centerline, regardless of the vehicle's velocity:

$$\theta^i_{k:k+N} = [\mathbf{\Theta}^i(s^i_k), \mathbf{\Theta}^i(s^i_k + 2) \ldots, \mathbf{\Theta}^i(s^i_k + 2T)],$$

where s^i_k is the total distance traveled along the centerline of the track at time k. Given the vehicle's current state (5.34) and this environment forecast, the learned GP strategy predicts the target values of the strategy states (5.36) at $N = 2$ seconds into the future. The forecast and control horizon were chosen by examining past task data and estimating how close an environment change had to be in order to impact the vehicle's locally optimal trajectory. Training two GPs using GPyTorch on a 2017 MacBook Pro with 2.8 GHz Quad-Core Intel Core i7 took 494 seconds. Evaluating these GPs took an average of 0.004 seconds on the same processor.

Figure 5.10 compares two states (v_x and e_y) of the HPL closed-loop trajectory on a new task (the "AE" track from [9]) with those of the optimal minimum-time trajectory. These two states are most informative on whether a good strategy was learned, since they correspond most closely with the chosen strategy states. The learned strategy performs well on this new task, and is able to pick up the pattern between track curvature and the raceline. the GP trained to predict how far the vehicle should travel in the next N steps (affecting the resulting v_x values) performs slightly better than the GP trained to predict deviation from the centerline (e_y). This trend was found in all three test tracks, and is likely due to deviations in optimal steering on straightaways, where the angle of the raceline depends on a longer environment forecast than considered.

The HPL trajectory is plotted and compared to the track's raceline in Fig. 5.11, with a colormap indicating how quickly the car drives in various sections of the track. Confirming the trend in Fig. 5.10, the plot shows the HPL control accurately predicts the ideal vehicle speed, speeding up and slowing down in the same sections of the racetrack as the optimal raceline. It is also clear that the system has learned various smart driving rules introduced in Sec. 5.3, including steering out before cutting the insides of corners and taking direct routes between two curves. More examples of the HPL closed-loop trajectory, with snippets from all three test tracks, are depicted in Fig. 5.12, and further plots are included in the Appendix.

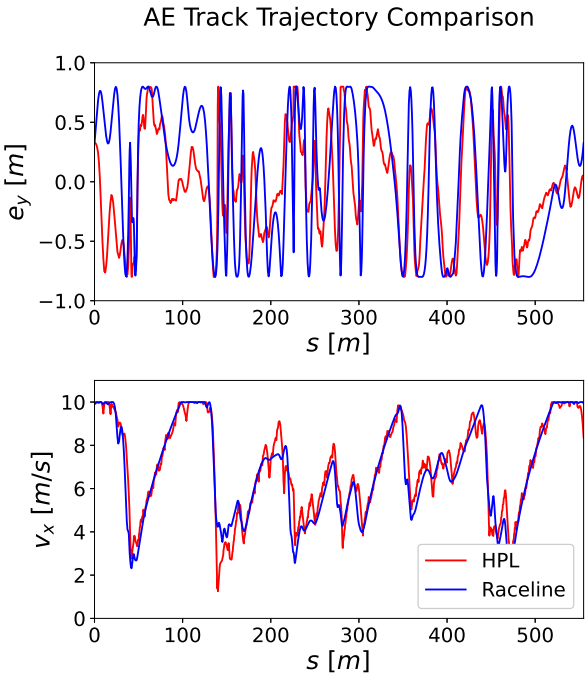


Figure 5.10: The two strategy GPs, trained using minimum-time trajectory data from seven race tracks, are able to predict (red) the centerline deviation and longitudinal velocity of a new, unseen track's raceline (blue).

The HPL control architecture results in feasible and fast executions for all three tested race tracks. Because the learned strategy GPs were able to predict the velocity trend so well (see Fig. 5.10), the three task durations were all within 5% of the respective raceline time (see Table 5.1). We again stress the ability of the strategy GPs to extrapolate patterns seen during training despite the tracks' very varied curvature.

In order to evaluate the benefit of using a hierarchical control framework, we also tested a controller with higher risk factor. As described in Sec. 5.5, this type of controller ($\beta = 1$) does not take the safety set $\mathcal{X}_E$ into account when creating the target set. As a result, the MPC terminal set constrains the strategy states to be in the GP's predicted set and the remaining state to simply be in the allowable state space. The safety set $\mathcal{X}_E$ is not used to guide the process of lifting the reduced-dimension strategy set into the full-dimensional target set. This means there are no guarantees that predicted terminal state will be in the domain of our safety control, and therefore it is possible to lose feasibility later in the task. Indeed, in the racing task (just as in the robotic manipulation task in Sec. 5.9), the simple high-risk controller was unable to complete a single lap without becoming infeasible. These failures occurred early in each of the three test tracks (in the first 10% of the tracks), during sharp corners where the safety controller was required but infeasible. This again demonstrates the

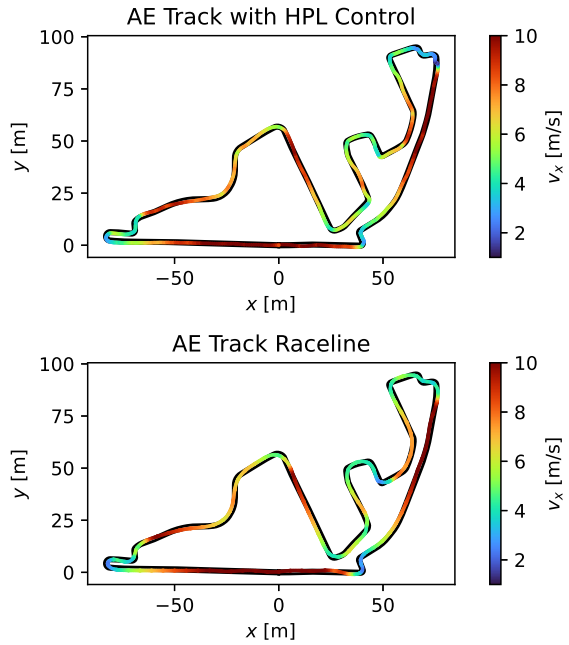


Figure 5.11: The HPL controller is able to match the speed profile and shapes of the minimum time trajectory, slowing down in curves and speeding up to the maximum allowable velocity on straight segments.

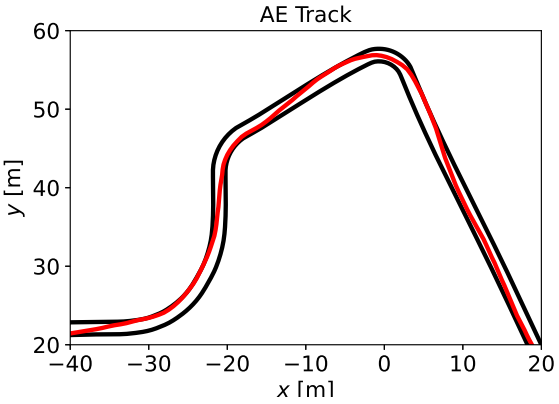
value of using both data-driven and physics-based components, and a hierarchical framework for integrating them in a structured manner.

Track	HPL Lap Time [s]	Raceline Lap Time [s]
AE	87.6	85.1
BE	92.3	89.3
US	85.6	83.3

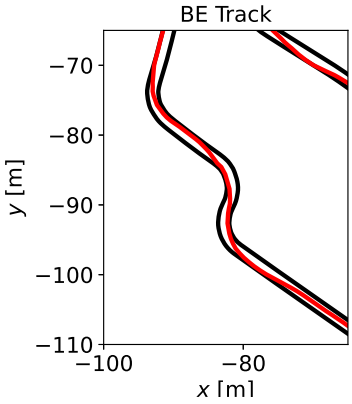
Table 5.1: The HPL controller results in lap times less than 5% longer than the minimum-time trajectory, demonstrating that an effective racing strategy was learned.

Neural Network as Strategy

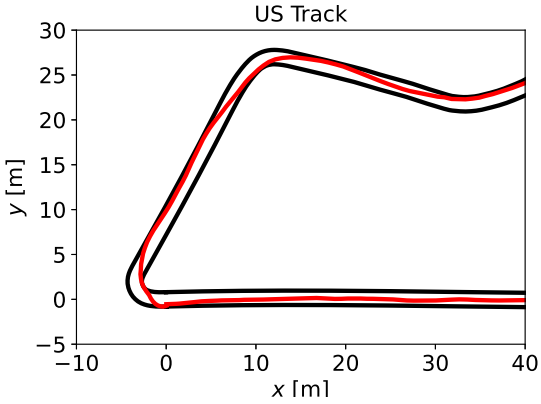
An additional trial was run where a neural network, rather than a GP, was used to represent the strategy. The training data was constructed in the same way as described in Sec. 5.10,



(a)



(b)



(c)

Figure 5.12: The HPL controller uses learned strategies to cut corners in all three test race tracks.

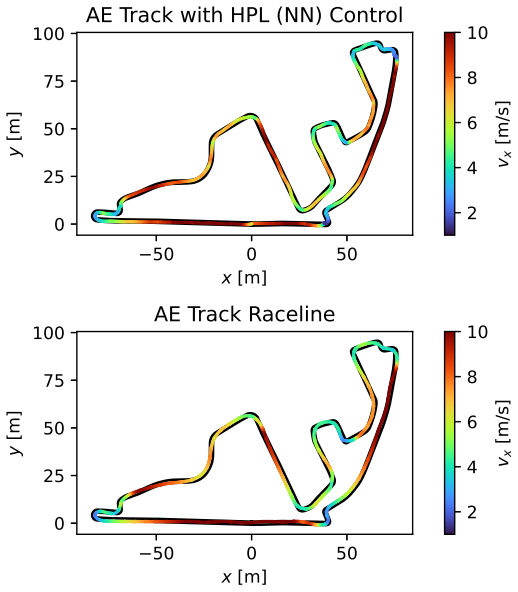


Figure 5.13: The HPL controller is able to match the speed profile and shapes of the minimum time trajectory

and a neural network with five linear hidden layers was trained using the python SKLearn library. One network could be trained to predict both strategy states, requiring 15 seconds of training time using Torch on a Pro 2017 MacBook Pro with 2.8 GHz Quad-Core Intel Core i7.

The resulting HPL trajectory on the AE track is shown and compared to the minimum-time trajectory in Figs. 5.13-5.14. Overall, the neural network strategy performs well, resulting in a lap time of 89.4 seconds, only slightly slower than the GP lap time (87.6 seconds) and the optimal lap time (85.1 seconds). This success validates that the HPL algorithm can easily be used with the designer's choice of strategy parameterizations, keeping in mind the considerations described in Sec. 5.4.

Interestingly, the e_y and v_x differences between the HPL trajectories (both of the GP and neural network) and the minimum time trajectories (as shown in Figs. 5.14 and 5.10) occur in similar regions of the track, including around the $s = 100$ and $s = 300$ meter marks, which are both long straightaway sections. The fact that both strategy parameterizations, the GP and the neural network, mischaracterized the race line in the same track section suggests that the training data could not be successfully exploited to race this section of the new track. Indeed, the ideal raceline along long straightaway sections is determined by the shape of the closest upcoming curve, which may lie beyond the environment forecast horizon and therefore unknown to the HPL control algorithm. One approach to resolve this would be extending the environment forecast horizon, but this inevitably increases the computational complexity of both the offline training and online implementation of the HPL algorithm.

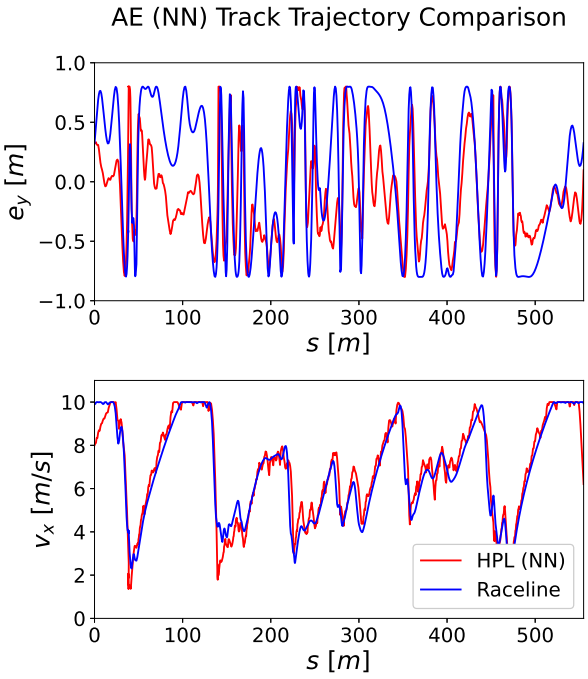


Figure 5.14: A strategy parameterized as a neural network can also capture the true raceline reasonably well, but with more errors in the predicted centerline deviation than the strategy GP.

5.11 Application 3: Flappy Bird

We use Hierarchical Predictive Control to improve performance in the Flappy Bird computer game[4]. Here, the task is to steer a small bird around a series of pipe obstacles by controlling the timing of its wing flaps. The pipe obstacles come in pairs from the bottom and top of the screen, leaving a gap for the bird to carefully fly through. As the bird moves through the task, it sees only a fixed distance ahead: the screen only the shows the two upcoming pairs of pipes. The strategy behind Flappy Bird lies in planning short-term trajectories that are robust to randomness in the heights of the future pipe obstacles still hidden beyond the screen (see Fig. 5.15).

[4] `https://flappybird.io`

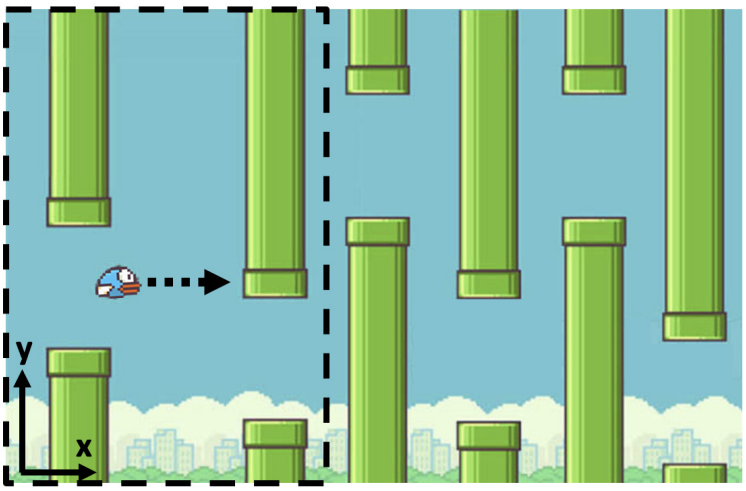


Figure 5.15: The goal of Flappy Bird is to guide the bird through a series of pipe obstacles. Only the pipes visible in the game screen (dashed rectangle) are visible to the bird at any time.

System and Task Description

We use a Flappy Bird simulator designed by Philip Zucker in [139] for our experiments. Following Zucker's work, we model the bird as a simple three-state system with dynamics

$$\begin{aligned} x_{k+1} &= x_k + 4, \\ y_{k+1} &= y_k + v_{y_k} \\ v_{y_{k+1}} &= v_{y_k} - 1 + 16u_k, \end{aligned} \tag{5.37}$$

where x_k and y_k are the horizontal and vertical position of the bird. The bird moves with constant horizontal velocity, while the vertical velocity v_{y_k} is subject to a constant gravitational force and the wing flap control $u_k \in \{0, 1\}$, where $u_k = 1$ implies a wing flap at time k. For each pair of pipe obstacles the bird flies through without crashing, a point is earned.

Different tasks $\mathcal{T}^i$ correspond to different sequences of pipe obstacles, described using $\mathbf{\Theta}^i$, where $\mathbf{\Theta}^i$ maps the bird's state (x_k, y_k) to the set of pipe obstacles $\mathcal{P}_k$ visible at time k.

We choose the strategy states to be the vertical distances between the bird and the two closest upcoming pipe gaps:

$$\tilde{x}_k = [y_k^{p,1} - y_k,\ y_k^{p,2} - y_k], \tag{5.38}$$

where $y_k^{p,1}$ and $y_k^{p,2}$ are the heights of the two closest upcoming lower pipe obstacles. Note that because the bird has a constant horizontal velocity, we only need to consider strategy states that describe the bird's height - the bird's horizontal movement is predetermined, and not regulated by a strategy. The safety control (5.21) is an MPC tracking the interpolated centerline of the visible pipe gaps.

Hierarchical Predictive Learning Results

We collect task executions that solve a series of 15 control tasks $\{\mathcal{T}^1, \ldots, \mathcal{T}^{15}\}$, with each control task corresponding to a new instance of the Flappy Bird game. The executions are used to create training data for a GP, using an environment forecast of $T = 45$ and a control horizon of $N = 30$ time steps (with a sampling frequency of 30 Hz.). This represents an environment forecast of two-thirds of the screen. Because the horizontal velocity is fixed in these tasks, we only train a single strategy GP to predict the target vertical distance to the nearest pipe gap, $(y^{p,1}_{k+N} - y_{k+N})$. The GP training took 327 seconds in GPyTorch with a 2017 MacBook Pro with 2.8 GHz Quad-Core Intel Core i7.

After training the GP model, the HPL algorithm is used to solve 50 different new Flappy Bird games. For comparison, we also evaluate a publicly available center-pipe tracking controller [139] with the same control horizon on the same set of 50 games. Statistics of the scores earned by each controller during the 50 games are recorded below in Table 5.2.

	Mean Score	Med. Score	Min. Score
Standard MPC	28	23	1
HPL Control	161	105	38
pseudo-HPL	74	46	10

Table 5.2: The HPL controller is compared with a publicly available standard MPC controller [139]. In a trial of 50 tasks, the HPL controller significantly outperformed the standard controller.

The HPL controller vastly outperforms the center-pipe tracking controller, with a mean score of 161 (versus 28). In fact, the HPL controller's minimum recorded score (38) is larger than the average score achieved by the standard controller. In the 50 evaluated games, there was only a single instance of the standard controller either outperforming or achieving the same score as the HPL controller on a specific task.

HPL is able to outperform the standard controller by utilizing the safe set $\mathcal{X}_E$ to construct the terminal set at each time step, and ensure that the short-term trajectory plans lead to feasible solutions in future time steps. This is particularly important in the Flappy Bird task, because the pipe heights of two consecutive pipe obstacles are completely uncorrelated. Unlike in the autonomous racing task, where environments change gradually, the Flappy Bird controller needs to consider all possible future pipe heights at each time step.

Figure 5.16 compares the open-loop trajectory predictions for the HPL controller (left image, in red) and the standard controller (right image, in blue) while solving the same task. At the time step shown, the oncoming pipe is still beyond the control horizon (both controllers use the same control horizon). However, because the HPL controller plans trajectories that will be feasible for any kind of upcoming pipe obstacle, its open-loop trajectory aims downwards, towards the center of the screen. From this end position, the controller is more likely to find a feasible next trajectory regardless of the shape of the next obstacle.

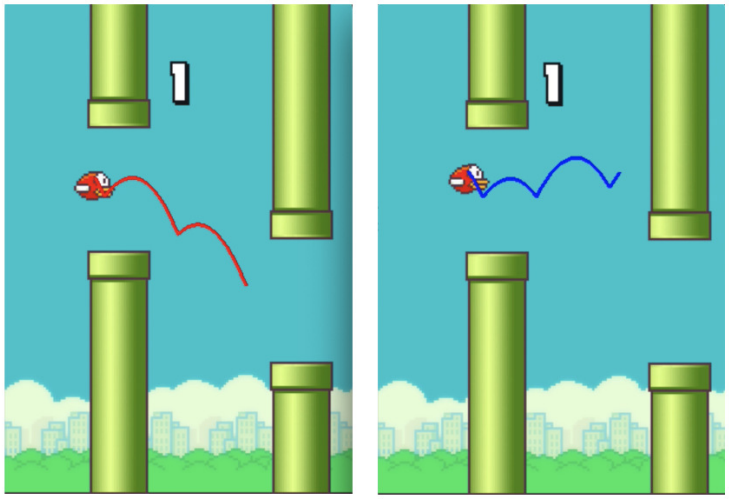


Figure 5.16: The open-loop trajectories of the HPL controller (left image, in red) and the standard MPC controller (right image, in blue) are compared. The HPL controller uses the pre-determined safety set $\mathcal{X}_E$ to plan trajectories that will be feasible regardless of the upcoming pipe obstacle height, resulting in a trajectory towards the center of the screen.

In contrast, the standard controller continues to track the center of the pipe, without explicitly considering that the upcoming obstacle may be of a different shape. Because the control can only increase the rate of acceleration, whereas the bird's downward acceleration due to gravity is fixed, it is dangerous to plan for wing flaps close to still-unknown obstacles. Crashing into the side of pipe obstacle pipes as a result of too-close wing flaps was the most common failure mode of the standard controller in the games considered. These crashes can only be prevented by using a standard controller if the control horizon N is extended, but this will increase the computational complexity of the controller. Therefore it is clear that in Flappy Bird, the controller that plans ahead using strategies and a safety set outperforms one that does not.

Discussion

It is worth noting that in this particular application, there is no difference between safety and performance. As long as the bird satisfies all environmental constraints, the score increases; there are no additional points for playing with a minimum number of wing flaps. Such an additional objective could, of course, be specified in an optimal control design, but it is not intrinsic to the game.

This direct correlation between Flappy Bird *performance* and Flappy Bird *safety* suggests that while the learned strategies may be useful for solving the task, they are not necessary

in order to gain higher scores. In fact, a pseudo-HPL controller which only requires that the terminal state is in the safe set $\mathcal{X}_E$, without applying any learned strategies, also outperformed the standard controller (see Table 5.2). This is likely due to the fact that a good approximation for the true $\mathcal{X}_E$ could be found and implemented in the safety controller alone. It is clear that in tasks with more complex dynamics or environments, where only rougher approximations of the safe set $\mathcal{X}_E$ can be determined, the additional incorporation of strategies to guide the system away from constraint violation will improve performance even further. Therefore, even when performance is directly linked to safety (as in Flappy Bird), all components of the HPL architecture play critical roles.

5.12 Discussion

The HPL algorithm provides a structured framework for solving tasks in new environments. Here, we would like to emphasize the benefits of the algorithm's key features.

Incorporating Human Intuition

Our proposed characteristics of strategy outlined in Sec. 5.3 define strategies as mapping to target sets in a reduced dimension known as strategy space. As detailed in Sec. 5.4, there are myriad ways of determining appropriate strategy states and inputs for a particular task, including using human intuition to select them.

In some cases it can be tedious (or impossible) to hand-design features for control design; indeed this motivates much of the rise in ML for control. However, providing structured opportunities for incorporating human intuition wherever it may be available is still of great benefit. Only in rare cases will real-world systems be deployed with no understanding of their own dynamics or objectives in unknown and time-varying environments. We believe it is much more realistic to have a control framework in mind that offers modular possibilities for human intuition to help shape control parameters whenever that intuition exists, such as the strategy states or forecast horizons in the HPL framework.

Ease of Interpretation

One benefit of choosing intuitively meaningful strategy states is that it makes it easier to interpret the status of the learned strategies. If we can visualize the learned strategy sets and compare them to our human intuition about the task, we can easily understand if more training data is needed, or if the strategy mapping is logical. This analysis is an especially useful tool when trying to pinpoint a failure in the entire HPL control framework.

Using Data Safely

The hierarchical approach of HPL allows for some separation between maximizing performance and ensuring feasibility: effective task performance is encouraged using learned strategies, and feasibility is ensured via the safety controller and safe set projections. (As discussed in Sec. 5.11, there is some overlap between the two, especially in complex systems.)

In our proposed approach, these two hierarchical levels align with the algorithm's boundaries between data-driven and physics-based models. We use the physical model of the system in order to guarantee feasibility, and we use learning to represent complex objectives whose exact relationships are difficult characterize. This framework allows for responsible integration of data, using it to guide long-term behaviors rather than decide immediate control actions.

By varying the risk factor β in our controller formulation, we were able to evaluate the benefits of using a physical model and safety controller to move from reduced-dimension strategy space to full-dimensional space. In all tested applications (see Sec. 5.9-Sec. 5.10), relying only on the data-driven strategy led to incomplete and infeasible task executions. These results emphasize the usefulness and importance of the hierarchical control framework, which allows for a structured combination of data-driven and physics-based methods.

5.13 Conclusion

A data-driven hierarchical predictive learning architecture for control in unknown environments is presented. The HPL algorithm uses stored executions that solve a variety of previous control tasks in order to learn a generalizable control strategy for new, unseen tasks. Based on a local description of the task environment, the learned control strategy proposes regions in the state space towards which to aim the system at each time step, and provides input constraints to guide the system evolution according to previous task solutions. We prove that the resulting policy is guaranteed to be feasible for the new tasks, and evaluate the effectiveness of the proposed architecture in a simulation of a robotic path planning task. Our results confirm that HPL architecture is able to learn applicable strategies for efficient and safe execution of unseen tasks.

5.14 Additional Results

The plots comparing the HPL controller trajectory to the optimal minimum-time trajectory for the remaining two test tracks (BE and US tracks) are shown here. As also shown in the figures in Sec. 5.10, the HPL controller using short-sighted environmental forecasts in conjunction with strategy GPs results in a close-loop trajectory very similar to the raceline.

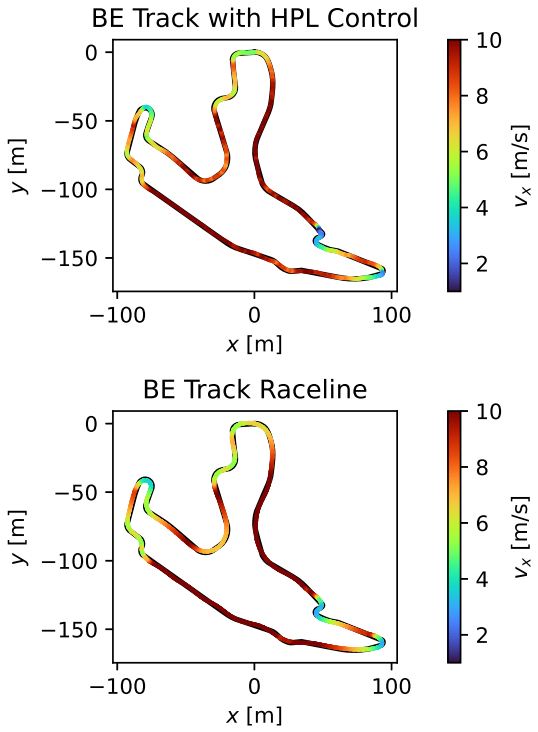


Figure 5.17: The HPL controller matches the speed profile and shape of the minimum time trajectory on the BE track.

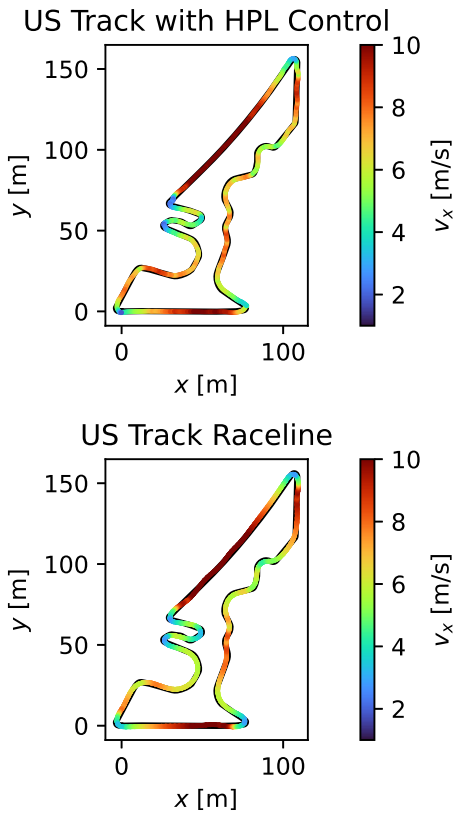


Figure 5.18: The HPL controller matches the speed profile and shape of the minimum time trajectory on the US track.

Chapter 6

Discussion

We presented four methods for approaching the changing environment problem. In each chapter, we consider a discrete-time nonlinear dynamical system solving a series of tasks in different environments. While the state and input constraints are constant across tasks, each task environment imposes additional task-specific constraints, described via an environment descriptor function $\boldsymbol{\Theta}$. Our goal was to use a set of n trajectories solving n previous tasks in different environments in order to find a safe and effective MPC policy for a new task environment described according to $\boldsymbol{\Theta}^{n+1}$.

Each proposed method makes specific assumptions about the system or constraints, and is applicable to situations where the functions $\boldsymbol{\Theta}$ have particular properties (e.g. piecewise constant, time-invariant). Here we review and compare the presented approaches.

6.1 Task Decomposition

Problem Assumptions

1. Assumptions on the system dynamics:
   a) Time-invariant system dynamics.
2. Assumptions on the system constraints:
   a) Time-invariant system constraints.
3. Assumptions on $\boldsymbol{\Theta}$:
   a) $\boldsymbol{\Theta}$ is known before beginning the new control task.
   b) $\boldsymbol{\Theta}$ is time-invariant.
   c) Each task $\mathcal{T}^i$ can be divided into an ordered sequence of subtasks $\{\mathcal{S}_j\}_{j=1}^{M}$, defined over a subtask workspace $\mathcal{X}_j \subseteq \mathcal{X}$. In each subtask, the environment descriptor function $\boldsymbol{\Theta}_j^i$ is the restriction of $\boldsymbol{\Theta}^i$ onto $\mathcal{X}_j$.

d) All considered tasks can be split into different ordered sequences of the *same* M subtasks, e.g.

$$\mathcal{T}^1 = \{\mathcal{S}_j^1\}_{j=1}^M, \quad \mathcal{T}^2 = \{\mathcal{S}_{l_j}^1\}_{j=1}^M, \quad [l_1, l_2, ..., l_M] = \text{shuffle}([1, 2, ..., M]).$$

4. Assumptions on recorded executions:

a) The recorded trajectories include state and input trajectories as well as information about the cost-to-go for each state-input pair.

b) In each recorded trajectory, the controller was optimizing the same objective function. This ensures that the recorded costs-to-go are comparable.

MPC Formulation

Task Decomposition provides a method for using stored executions of previous tasks in order to find a safe set $\mathcal{SS}_{n+1}$ for a task in a new environment $\mathbf{\Theta}^{n+1}$. Note that the algorithm also finds the costs-to-go $\mathcal{SQ}_{n+1}$ for each point in the safe set, as well as a safe input to apply $\mathcal{SU}_{n+1}$. Once these safe sets are found for the new task using Alg. 1, a safe set based ILMPC policy solves the following optimal control problem at each time k of solving the new task:

$$\begin{aligned} \boldsymbol{u}^\star = \min_{u_{k|k},\ldots,u_{k+N-1|k}} \quad & \sum_{t=0}^{N-1} p(x_{k+t|k}, u_{k+t|k}) + \mathcal{SQ}_{n+1}(x_{k+N|k}) \\ \text{s.t.} \quad & x_{k+t+1|k} = f(x_{k+t|k}, u_{k+t|k}) \\ & u_{k+t|k} \in \mathcal{U}, \ \forall t \in \{0, \ldots, N-1\} \\ & x_{k+t|k} \in \mathcal{X}(\mathbf{\Theta}), \ \forall t \in \{0, \ldots, N\} \\ & x_{k|k} = x_k \\ & x_{k+N|k} \in \mathcal{SS}_{n+1} \cup \mathcal{P}^{n+1}. \end{aligned} \tag{6.1}$$

The controller searches for an input sequence over the planning horizon N that controls the system to the state in the determined safe state set or task target set with the lowest cost-to-go reachable from the current state x_k. The controller formulation includes both a data-driven cost estimate (via $\mathcal{SQ}_{n-1}$) and a data-driven terminal set $\mathcal{SS}_{n+1}$. Thus, the task decomposition algorithm utilizes stored trajectory data to ensure both the safety and performance of the new task's controller.

The complexity of solving (6.1) depends in part on the linearity or convexity of the system dynamics, constraints, and stage cost function p. Note that both the terminal constraint and terminal cost require a binary variable: the last predicted state is exactly equal to one entry in $\mathcal{SS}_{n-1}$, and the terminal cost is equal to the stored cost of the chosen entry. Thus, (6.1) is a mixed integer program (MIP). MIPs can be handled by off-the-shelf solvers, though this formulation does provide an additional layer of run-time complexity.

We also emphasize that formulating the controller (6.1) requires knowing the entire new safe set $\mathcal{SS}_{n-1}$. Thus while the controllability analysis of Alg. 1 can be carried out offline,

the resulting safe set must be stored online while solving the new control task. While a large set $\mathcal{SS}_{n-1}$ can improve the closed-loop cost of the resulting execution, it should be noted that the number of entries in the safe set is inversely correlated with the solve time of (6.1).

6.2 Task Decomposition for Piecewise Linear Systems

Problem Assumptions

1. Assumptions on the system dynamics:
   a) Each task $\mathcal{T}^i$ can be divided into an ordered sequence of subtasks $\{\mathcal{S}_j\}_{j=1}^M$, defined over a subtask workspace $\mathcal{X}_j \subseteq \mathcal{X}$. Within each subtask workspace, the system is governed by time-invariant linear dynamics.
2. Assumptions on the system constraints:
   a) Within each subtask workspace, the system constraints are time-invariant and convex.
3. Assumptions on $\boldsymbol{\Theta}$:
   a) $\boldsymbol{\Theta}$ is known before beginning the new control task.
   b) $\boldsymbol{\Theta}$ is time-invariant.
   c) In each subtask, the environment descriptor function $\boldsymbol{\Theta}_j^i$ is the restriction of $\boldsymbol{\Theta}^i$ onto $\mathcal{X}_j$. All considered tasks can be split into different ordered sequences of the *same* M subtasks, e.g.
   $$\mathcal{T}^1 = \{\mathcal{S}_j^1\}_{j=1}^M, \quad \mathcal{T}^2 = \{\mathcal{S}_{l_j}^1\}_{j=1}^M, \quad [l_1, l_2, ..., l_M] = \text{shuffle}([1, 2, ..., M]).$$
   d) Within each subtask workspace, the environment constraints defined by $E(\boldsymbol{\Theta}_j(x, u))$ are convex.
4. Assumptions on recorded executions:
   a) The recorded trajectories include state and input trajectories as well as information about the cost-to-go for each state-input pair.
   b) In each recorded trajectory, the controller was optimizing the same objective function. This ensures that the recorded costs-to-go are comparable.

MPC Formulation

In the reformulation of Task Decomposition for piecewise linear systems, a similar low-level controller emerges. At each time k of solving the new task, the controller solves the optimal control problem:

$$\begin{aligned} \boldsymbol{u}^\star = \underset{u_{k|k},\ldots,u_{k+N-1|k},I,K}{\arg\min} \quad & \sum_{t=k}^{k+N-1} p(x_{k+t|k}, u_{k+t|k}) + \mathcal{SQ}(x_{k+N|k}) \\ \text{s.t.} \quad & x_{k+t+1|k} = f(x_{k+t|k}, u_{k+t|k}) \\ & u_{k+t|k} \in \mathcal{U}_i,\ \forall t \in \{0,\ldots,N-1\} \\ & x_{k+t|k} \in \mathcal{X}(\boldsymbol{\Theta}),\ \forall t \in \{0,\ldots,N\} \\ & x_{k|k} = x_k \\ & x_{k+N|k} \in \mathcal{CK}_{I,K} \cup \mathcal{P}, \end{aligned} \tag{6.2}$$

which searches for an input sequence such that the last predicted state is in a time-indexed convex hull safe set for the new task, found using Alg. 2. The key difference between (6.2) and (6.1) is that (6.2) stores multiple smaller convex safe sets, rather than one discrete safe set for the entire task. The last predicted state is then constrained to be anywhere inside one such time-indexed convex hull set. There is no binary variable required to constrain the terminal state to be exactly equal to one stored state, but a binary variable is required to indicate which time-indexed convex hull set the terminal state is in. Thus, the reformulation is still an MIP. However, the dimension of the required binary variable is smaller than in (6.1) by up to a factor of n, the number of previously stored task trajectories.

6.3 Probabilistically Safe Controllable Sets

Problem Assumptions

1. Assumptions on the system dynamics:
   None to apply method. However, probabilistic guarantees only hold for systems with (time-varying or time-invariant) linear dynamics.

2. Assumptions on the system constraints:
   None to apply method. However, probabilistic guarantees only hold for systems whose constraints are convex at each time-step.

3. Assumptions on $\boldsymbol{\Theta}$:

   a) In each task, $\boldsymbol{\Theta}$ maps a state x_k and time step k to a scenario parameter z_k. This scenario parameter evolves according to a task-invariant scenario dynamics

function:

$$\begin{aligned}\boldsymbol{\Theta}^i(x_k, k) &= z_k^i \\ &= \phi(z_{k-1}^i, k-1) \\ &= \phi(\boldsymbol{\Theta}^i(x_{k-1}, k-1), k-1).\end{aligned}$$

b) $\boldsymbol{\Theta}$ need not be known before beginning the new control task, but at each time step k of solving a task, a limited N-step scenario parameter forecast, $[z_k, z_{k+1}, \ldots, z_{k+N}]$, is known. This forecast is considered to be correct.

4. Assumptions on recorded executions:

   a) Each recorded trajectory is of length T.

   b) All tasks share a common goal set $\mathcal{X}_T$.

MPC Formulation

We train GPs to represent strategy functions, learned from stored task trajectories. Given a local forecast of the time-varying environment, these strategy GPs are used to construct a set that is (with high probability) controllable to the task's target set. At each time k, we solve:

$$\begin{aligned}\boldsymbol{u}^\star(x_k, z_{k:k+N}) = \underset{u_{k|k},\ldots,u_{k+N-1|k}}{\arg\min} \quad & \sum_{k=0}^{N-1} p(x_{k+t|k}, u_{k+t|k}) + q(x_{k+N|k}) \\ \text{s.t.} \quad & x_{k+t+1|k} = f(x_{k+t|k}, u_{k+t|k}) \\ & u_{k+t|k} \in \mathcal{U},\ \forall t \in \{0, \ldots, N-1\} \\ & x_{k+t|k} \in \mathcal{X} \cap \mathcal{X}(z_{k+t}),\ \forall t \in \{0, \ldots, N\} \\ & x_{k|k} = x_k \\ & x_{k+N|k} \in \mathcal{X}_N(s_k).\end{aligned} \tag{6.3}$$

In contrast with (6.1) and (6.2), the optimal control problem (6.3) does not require explicitly storing previous trajectories or a collection of safe states. Instead, we store $n_x \cdot T$ GPs, where n_x is the dimension of the system state and T the length of the task.

At each iteration of the controller, n_x GPs have to be evaluated in order to construct $\mathcal{X}_N(s_k)$ and solve (6.3). Evaluating each GP requires a matrix inverse and a matrix multiplication operation—however, the matrix inverse operation relies only on previous task data and can thus be completed offline before beginning the new control task. Once the GP is evaluated, a convex terminal set is constructed. Thus, if the system dynamics are linear and the objective and constraints are convex, (6.3) is a convex optimization problem (in contrast to 6.1 and 6.2).

Unlike the two Task Decomposition approaches, the controller in (6.3) does not use a data-driven objective function. Here we are only concerned with controllability and ensuring constraint satisfaction of the system, and not necessarily maximizing performance. The control designer is therefore free to select any stage and terminal costs p and q.

6.4 Hierarchical Predictive Learning

Problem Assumptions

1. Assumptions on the system dynamics:
   a) Time-invariant system dynamics.
2. Assumptions on the system constraints:
   a) Time-invariant system and input constraints.
3. Assumptions on $\boldsymbol{\Theta}$:
   a) $\boldsymbol{\Theta}$ need not be known before beginning the new control task, but at each time step k of solving a task, a limited N-step environment forecast, $[\theta_k, \theta_{k+1}, \ldots, \theta_{k_N}]$, is known. This forecast is considered to be correct.
4. Assumptions on recorded executions:
   a) In each recorded trajectory, the controller optimizes the same objective function to a satisfactory degree.

MPC Formulation

A generalization of learning data-driven controllable sets, Hierarchical Predictive Learning uses GPs to represent reduced-dimension target sets for navigation tasks. A total number of $n_{\tilde{x}}$ GPs are trained from stored trajectories; one for each reduced-dimension strategy state. At each time k of solving the new task, the controller evaluates the GPs in order to construct a hyperrectangular terminal set, and then solves:

$$
\begin{aligned}
\mathbf{u}^\star(x_k) = \underset{u_{k|k},\ldots,u_{k+N_k^{MPC}-1|k}}{\arg\min} \quad & \sum_{j\in\mathcal{S}_k}^{N_k^{MPC}-1} \operatorname{dist}\big(x_{j|k}, \mathcal{X}_{k+j}\big) \\
\text{s.t.} \quad & x_{k+t+1|k} = f(x_{k+t|k}, u_{k+t|k}) \\
& u_{k+t|k} \in \mathcal{U},\ \forall t \in \{0,\ldots,N-1\} \\
& x_{k+t|k} \in \mathcal{X}(\boldsymbol{\Theta}^{n+1}),\ \forall t \in \{0,\ldots,N\} \\
& x_{k|k} = x_k \\
& x_{k+N_k^{MPC}|k} \in \mathcal{X}_{k+N_k^{MPC}}.
\end{aligned}
\tag{6.4}
$$

At each iteration of the controller, $n_{\tilde{x}}$ GPs have to be evaluated. Note that since we expect $n_{\tilde{x}} < n_x$, these are fewer GP evaluations than are required for (6.3) at each time step. The matrix inversion operation can again be pre-computed, for improved online optimization speed. Note that in addition to evaluating the GPs, (6.4) also requires finding a new MPC horizon at each time step—but this is computationally trivial.

Whereas (6.3) only uses data to construct a terminal set, (6.4) also alters the MPC's objective function. This encourages the system to track the strategy target sets found at previous time steps. Since the terminal set is a hyperrectangular (and thus convex) set, and the distance function a convex function, the convexity of (6.4) is determined by the system dynamics.

6.5 Performance Comparison

Table 6.1 concisely compares the four proposed control methods on the basis of their algorithm outputs, the ways in which they manipulate the low-level MPC controller, and the time requirements for the proposed online calculations. Specifically, "Online Set Comp." considers the worst-case online terminal set computation time required by each method as a function of the number of previously stored trajectories n, the number of system states n_x, and the length of each stored trajectory D^{max}. Note that because the task decomposition approaches discussed in Ch. 2-3 calculate a safe set offline, the online set computation is negligible. The PSCS and HPL approaches, on the other hand, require evaluating GPs online at each time step of solving the new task. The resulting terminal constraints for these two methods, however, are convex constraints and therefore easy to solve. The terminal constraints in TDMPC and TDMPC-PWL are mixed-integer linear constraints; these are typically NP-hard to solve. Thus there is a trade-off between computation required to find the terminal set and computation required to solve the resulting MPC problem.

Another key difference of note is that TDMPC and TDMPC-PWL require knowing the entire environment description a priori, whereas PSCS and HPL only require a local forecast of the environment. Similarly, TDMPC and TDMPC-PWL learn safe sets that are specific to the new task environment $\mathbf{\Theta}^{n+1}$; if the task changes again, to a new $\mathbf{\Theta}^{n+2}$, the TDMPC and TDMPC-PWL algorithms need to be repeated to find yet another safe set. This is not the case for HPL or PSCS, whose algorithms produce policies that rely on a local environment description to generalize between a variety of tasks. Once trained on a set of n task trajectories, the learned HPL or PSCS strategy functions can be utilized in as many new task environments as desired. As a result, if the goal is to use stored trajectories to quickly adapt to a variety of new tasks, either PSCS or HPL should be chosen. If on the other hand a safe trajectory is needed to warm-start a controller of a desired form for a single new task, the TDMPC or TDMPC-PWL approaches will be of use, since these methods immediately provide a safe trajectory for the new task along with the safe control policy.

		TDMPC	TDMPC-PWL	PSCS	HPL
$\boldsymbol{\Theta}$ Props	Required Forecast	Entire Task	Entire Task	Local	Local
	Time-Invariant	✓	✓		
Alg. Output	Trajectory	✓	✓		
	Policy	✓	✓	✓	✓
MPC Impact	Provides Cost Fnc.	✓	✓		✓
	Provides Term. Set	✓	✓	✓	✓
	Online Set Comp.	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n_x n^2)$ + $\mathcal{O}(NLP)$	$\mathcal{O}(n_x(n\mathcal{D}^{max})^2)$
	Term. Constraint	MILP	MILP	QCQP	LP

Table 6.1: Comparing various features of the presented control methods.

Bibliography

[1] T. Alamo et al. "On the Computation of Robust Control Invariant Sets for Piecewise Affine Systems". In: *Assessment and Future Directions of Nonlinear Model Predictive Control*. Ed. by Rolf Findeisen, Frank Allgöwer, and Lorenz T. Biegler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 131–139.

[2] Tzanis Anevlavis et al. "Controlled invariant sets: implicit closed-form representations and applications". In: *arXiv preprint arXiv:2107.08566* (2021).

[3] Suguru Arimoto, Masahiro Sekimoto, and Sadao Kawamura. "Task-space iterative learning for redundant robotic systems: Existence of a task-space control and convergence of learning". In: *SICE Journal of Control, Measurement, and System Integration* 1.4 (2008), pp. 312–319.

[4] Ashley A Armstrong and Andrew G Alleyne. "A Multi-Input Single-Output iterative learning control for improved material placement in extrusion-based additive manufacturing". In: *Control Engineering Practice* 111 (2021), p. 104783.

[5] Christopher G Atkeson and Jun Morimoto. "Nonparametric representation of policies and value functions: A trajectory-based approach". In: *Advances in Neural Information Processing Systems*. 2003, pp. 1643–1650.

[6] Stanley Bak and Parasara Sridhar Duggirala. "Hylaa: A tool for computing simulation-equivalent reachability for linear systems". In: *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*. ACM. 2017, pp. 173–178.

[7] Somil Bansal et al. "Hamilton-Jacobi reachability: A brief overview and recent advances". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 2242–2253.

[8] Somil Bansal et al. "Hamilton-jacobi reachability: A brief overview and recent advances". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 2242–2253.

[9] Jonathan Beliën. *jbelien/F1-Circuits*. URL: `https://github.com/jbelien/F1-Circuits`.

[10] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. "A robot path planning framework that learns from experience". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 3671–3678.

[11] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. "A robot path planning framework that learns from experience". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 3671–3678.

[12] Dimitri P Bertsekas. "Feature-Based Aggregation and Deep Reinforcement Learning: A Survey and Some New Implementations". In: *arXiv preprint arXiv:1804.04577* (2018).

[13] Dimitri P Bertsekas and Ian B Rhodes. "On the minimax reachability of target sets and target tubes". In: *Automatica* 7.2 (1971), pp. 233–247.

[14] Dimitris Bertsimas and Bartolomeo Stellato. *The Voice of Optimization*. 2018. arXiv: 1812.09991 [math.OC].

[15] Franco Blanchini. "Set invariance in control". In: *Automatica* 35.11 (1999), pp. 1747–1767.

[16] Ruxandra Bobiti and Mircea Lazar. "Automated-sampling-based stability verification and DOA estimation for nonlinear systems". In: *IEEE Transactions on Automatic Control* 63.11 (2018), pp. 3659–3674.

[17] Shaunak D Bopardikar, Francesco Bullo, and Joao P Hespanha. "A cooperative homicidal chauffeur game". In: *Automatica* 45.7 (2009), pp. 1771–1777.

[18] F. Borrelli, A. Bemporad, and M. Morari. *Predictive Control for linear and hybrid systems*. Cambridge University Press, 2017.

[19] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[20] Douglas A Bristow, Marina Tharayil, and Andrew G Alleyne. "A survey of iterative learning control". In: *IEEE Control Systems Magazine* 26.3 (2006), pp. 96–114.

[21] M. Bujarbaruah, X. Zhang, and F. Borrelli. "Adaptive MPC with Chance Constraints for FIR Systems". In: *2018 Annual American Control Conference (ACC)*. June 2018, pp. 2312–2317.

[22] Jesús Velasco Carrau et al. "Efficient implementation of randomized MPC for miniature race cars". In: *2016 European Control Conference (ECC)*. IEEE. 2016, pp. 957–962.

[23] A. Chakrabarty et al. "Data-Driven Estimation of Backward Reachable and Invariant Sets for Unmodeled Systems via Active Learning". In: *2018 IEEE Conference on Decision and Control (CDC)*. 2018, pp. 372–377. DOI: 10.1109/CDC.2018.8619646.

[24] Ankush Chakrabarty et al. "Data-driven estimation of backward reachable and invariant sets for unmodeled systems via active learning". In: *2018 IEEE Conference on Decision and Control (CDC)*. IEEE. 2018, pp. 372–377.

[25] Y. Chen et al. "Data-Driven Computation of Minimal Robust Control Invariant Set". In: *2018 IEEE Conference on Decision and Control (CDC)*. 2018, pp. 4052–4058. DOI: 10.1109/CDC.2018.8619312.

[26] Yuxiao Chen and Necmiye Ozay. "Data-Driven Computation of Robust Control Invariant Sets With Concurrent Model Selection". In: *IEEE Transactions on Control Systems Technology* (2021).

[27] Ignasi Clavera et al. "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning". In: *arXiv preprint arXiv:1803.11347* (2018).

[28] Adam Coates, Pieter Abbeel, and Andrew Y Ng. "Learning for control from multiple demonstrations". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 144–151.

[29] M. K. Cobb et al. "Iterative Learning-Based Path Optimization for Repetitive Path Planning, With Application to 3-D Crosswind Flight of Airborne Wind Energy Systems". In: *IEEE Transactions on Control Systems Technology* (2019), pp. 1–13. DOI: 10.1109/TCST.2019.2912345.

[30] Isidro Cortés-Ciriano and Andreas Bender. "Deep confidence: a computationally efficient framework for calculating reliable prediction errors for deep neural networks". In: *Journal of chemical information and modeling* 59.3 (2018), pp. 1269–1281.

[31] Tianyu Dai and Mario Sznaier. "A Moments Based Approach to Designing MIMO Data Driven Controllers for Switched Systems". In: *2018 IEEE Conference on Decision and Control (CDC)*. 2018, pp. 5652–5657.

[32] Benjamin Decardi-Nelson and Jinfeng Liu. "Computing robust control invariant sets of constrained nonlinear systems: A graph algorithm approach". In: *Computers & Chemical Engineering* 145 (2021), p. 107177.

[33] Alex Devonport and Murat Arcak. "Estimating reachable sets with scenario optimization". In: *Learning for dynamics and control*. PMLR. 2020, pp. 75–84.

[34] Ludvig Doeser et al. "Invariant Sets for Integrators and Quadrotor Obstacle Avoidance". In: *2020 American Control Conference (ACC)*. IEEE. 2020, pp. 3814–3821.

[35] Jian Dong and Bin He. "Novel Fuzzy PID-Type Iterative Learning Control for Quadrotor UAV". In: *Sensors* 19.1 (2019). ISSN: 1424-8220. DOI: 10.3390/s19010024. URL: https://www.mdpi.com/1424-8220/19/1/24.

[36] Parasara Sridhar Duggirala et al. "C2E2: A verification tool for stateflow models". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2015, pp. 68–82.

[37] Ioannis Exarchos, Panagiotis Tsiotras, and Meir Pachter. "UAV collision avoidance based on the solution of the suicidal pedestrian differential game". In: *AIAA Guidance, Navigation, and Control Conference*. 2016, p. 2100.

[38] M. Fiacchini, T. Alamo, and E.F. Camacho. "On the computation of convex robust control invariant sets for nonlinear systems". In: *Automatica* 46.8 (2010), pp. 1334–1338. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2010.05.007. URL: https://www.sciencedirect.com/science/article/pii/S000510981000213X.

[39] Mirko Fiacchini and Mazen Alamir. *Computing control invariant sets is easy*. 2017. arXiv: 1708.04797 [cs.SY].

[40] Mirko Fiacchini, Teodoro Alamo, and Eduardo F Camacho. "On the computation of convex robust control invariant sets for nonlinear systems". In: *Automatica* 46.8 (2010), pp. 1334–1338.

[41] Nathanaël Fijalkow et al. "On the decidability of reachability in linear time-invariant systems". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 2019, pp. 77–86.

[42] Tesca Fitzgerald et al. "Human-guided Trajectory Adaptation for Tool Transfer". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '19. 2019, pp. 1350–1358.

[43] Tesca Fitzgerald et al. "Human-guided Trajectory Adaptation for Tool Transfer". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '19. 2019, pp. 1350–1358.

[44] Alexander Formalskii and A. Sirotin. "On the Geometric Properties of Reachable and Controllable Sets for Linear Discrete Systems". In: *Journal of Optimization Theory and Applications* 122 (Aug. 2004), pp. 257–284. DOI: 10.1023/B:JOTA.0000042521.51456.01.

[45] Thommen George Karimpanal and Roland Bouffanais. "Self-organizing maps for storage and transfer of knowledge in reinforcement learning". In: *Adaptive Behavior* (Dec. 2018), p. 105971231881856. DOI: 10.1177/1059712318818568.

[46] Monique Guignard and Aykut Ahlatcioglu. "The convex hull heuristic for nonlinear integer programming problems with linear constraints and application to quadratic 0–1 problems". In: *Journal of Heuristics* 27.1 (2021), pp. 251–265. DOI: 10.1007/s10732-019-09433-w. URL: https://doi.org/10.1007/s10732-019-09433-w.

[47] Monique Guignard and Aykut Ahlatcioglu. "The convex hull relaxation for nonlinear integer programs with convex objective and linear constraints". In: *Proceedings of the European Workshop on Mixed Integer Nonlinear Programming*. Citeseer. 2010, pp. 149–158.

[48] Ahmed El-Guindy, Dongkun Han, and Matthias Althoff. "Estimating the region of attraction via forward reachable sets". In: *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 1263–1270.

[49] Osman Güler and Filiz Gürtuna. "The extremal volume ellipsoids of convex bodies, their symmetry properties, and their determination in some special cases". In: *arXiv preprint arXiv:0709.0707* (2007).

[50] Lukas Hewing, Juraj Kabzan, and Melanie N Zeilinger. "Cautious model predictive control using Gaussian process regression". In: *IEEE Transactions on Control Systems Technology* (2019).

[51] David J Hoelzle, Andrew G Alleyne, and Amy J Wagoner Johnson. "Basis task approach to iterative learning control with applications to micro-robotic deposition". In: *IEEE Transactions on Control Systems Technology* 19.5 (2010), pp. 1138–1148.

[52] Matthias Hofer, Lukas Spannagl, and Raffaello D'Andrea. "Iterative learning control for fast and accurate position tracking with an articulated soft robotic arm". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 6602–6607.

[53] Roberto Horowitz. "Learning control of robot manipulators". In: *Journal of Dynamic Systems, Measurement, and Control* 115.2B (1993), pp. 402–411.

[54] Achin Jain et al. "Learning and Control Using Gaussian Processes: Towards Bridging Machine Learning and Controls for Physical Systems". In: *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*. ICCPS '18. 2018, pp. 140–149.

[55] Mehdi Jalalmaab et al. "Model predictive path planning with time-varying safety constraints for highway autonomous driving". In: *2015 International Conference on Advanced Robotics (ICAR)*. IEEE. 2015, pp. 213–217.

[56] Juraj Kabzan et al. "Learning-Based Model Predictive Control for Autonomous Racing". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3363–3370.

[57] Eric C Kerrigan and Jan M Maciejowski. "Invariant sets for constrained nonlinear discrete-time systems with application to feasibility in model predictive control". In: *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187)*. Vol. 5. IEEE. 2000, pp. 4951–4956.

[58] Maike Ketelhut et al. "Iterative learning control of ventricular assist devices with variable cycle durations". In: *Control Engineering Practice* 83 (2019), pp. 33–44.

[59] A. Khosravi et al. "Lower Upper Bound Estimation Method for Construction of Neural Network-Based Prediction Intervals". In: *IEEE Transactions on Neural Networks* 22.3 (2011), pp. 337–346. DOI: 10.1109/TNN.2010.2096824.

[60] Edgar D Klenske et al. "Gaussian process-based predictive control for periodic error correction". In: *IEEE Transactions on Control Systems Technology* 24.1 (2015), pp. 110–121.

[61] Juš Kocijan. *Modelling and control of dynamic systems using Gaussian process models*. Springer.

[62] Soonho Kong et al. "dReach: δ-reachability analysis for hybrid systems". In: *International Conference on TOOLS and Algorithms for the Construction and Analysis of Systems*. Springer. 2015, pp. 200–205.

[63] George Konidaris, Ilya Scheidwasser, and Andrew Barto. "Transfer in Reinforcement Learning via Shared Features". In: *The Journal of Machine Learning Research* 13 (Mar. 2012), pp. 1333–1371.

[64] Alexandar Kozarev et al. "Case studies in data-driven verification of dynamical systems". In: *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. 2016, pp. 81–86.

[65] Krisada Kritayakirana and J Christian Gerdes. "Using the centre of percussion to design a steering controller for an autonomous race car". In: *Vehicle System Dynamics* 50.sup1 (2012), pp. 33–51.

[66] Michal Kvasnica et al. "Multi-parametric toolbox (MPT)". In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 2004, pp. 448–462.

[67] J.B. Lasserre. "Reachable, controllable sets and stabilizing control of constrained linear systems". In: *Automatica* 29.2 (1993), pp. 531–536. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(93)90152-J. URL: https://www.sciencedirect.com/science/article/pii/000510989390152J.

[68] A Lederer, J Umlauft, and S Hirche. "Uniform Error Bounds for Gaussian Process Regression with Application to Safe Control". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2019.

[69] Jay H Lee and Kwang S Lee. "Iterative learning control applied to batch processes: An overview". In: *Control Engineering Practice* 15.10 (2007), pp. 1306–1318.

[70] Lucas Liebenwein et al. "Sampling-Based Approximation Algorithms for Reachability Analysis with Provable Guarantees". In: (2018).

[71] Pedro F Lima, Jonas Mårtensson, and Bo Wahlberg. "Stability conditions for linear time-varying model predictive control in autonomous driving". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 2775–2782.

[72] Chenggang Liu and Christopher G Atkeson. "Standing balance control using a trajectory library". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Citeseer. 2009, pp. 3031–3036.

[73] Zonglin Liu and Olaf Stursberg. "Recursive feasibility and stability of MPC with time-varying and uncertain state constraints". In: *2019 18th European Control Conference (ECC)*. IEEE. 2019, pp. 1766–1771.

[74] Jingyi Lu et al. "110th Anniversary: An Overview on Learning-Based Model Predictive Control for Batch Processes". In: *Industrial & Engineering Chemistry Research* 58.37 (2019), pp. 17164–17173.

[75] Ian M. Mitchell and Yoshihiko Susuki. "Level Set Methods for Computing Reachable Sets of Hybrid Systems with Differential Algebraic Equation Dynamics". In: Apr. 2008. DOI: 10.1007/978-3-540-78929-1_51.

[76] T Manrique et al. "MPC-based tracking for real-time systems subject to time-varying polytopic constraints". In: *Optimal Control Applications and Methods* 37.4 (2016), pp. 708–729.

[77] David Q Mayne et al. "Constrained model predictive control: Stability and optimality". In: *Automatica* 36.6 (2000), pp. 789–814.

[78] Samy Missoum, Palaniappan Ramu, and Raphael T. Haftka. "A convex hull approach for the reliability-based design optimization of nonlinear transient dynamic problems". In: *Computer Methods in Applied Mechanics and Engineering* 196.29 (2007), pp. 2895–2906. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2006.12.008. URL: https://www.sciencedirect.com/science/article/pii/S0045782507000813.

[79] Ian M Mitchell, Alexandre M Bayen, and Claire J Tomlin. "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games". In: *IEEE Transactions on automatic control* 50.7 (2005), pp. 947–957.

[80] Ofir Nachum et al. "Data-efficient hierarchical reinforcement learning". In: *Advances in neural information processing systems*. 2018, pp. 3303–3313.

[81] Ashutosh R Nandeshwar. "Models for calculating confidence intervals for neural networks". In: (2006).

[82] Quan Nguyen et al. "Dynamic walking on stepping stones with gait library and control barrier functions". In: *Arbor* 1001 (2016), p. 48109.

[83] Takashi Ohhira and Akira Shimada. "Model predictive control for an inverted-pendulum robot with time-varying constraints". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 776–781.

[84] Tom Oomen and Cristian R Rojas. "Sparse iterative learning control with application to a wafer stage: Achieving performance, resource efficiency, and task flexibility". In: *Mechatronics* 47 (2017), pp. 134–147.

[85] Myle Ott et al. *Analyzing Uncertainty in Neural Machine Translation*. 2018. arXiv: 1803.00047 [cs.CL].

[86] Georgios Papadopoulos, Peter J Edwards, and Alan F Murray. "Confidence estimation methods for neural networks: A practical comparison". In: *IEEE transactions on neural networks* 12.6 (2001), pp. 1278–1287.

[87] Alexandros Paraschos, Gerhard Neumann, and Jan Peters. "A probabilistic approach to robot trajectory generation". In: *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE. 2013, pp. 477–483.

[88] Krzysztof Patan and Maciej Patan. "Neural-network-based iterative learning control of nonlinear systems". In: *ISA transactions* 98 (2020), pp. 445–453.

[89] Vern I. Paulsen and Mrinal Raghupathi. *An Introduction to the Theory of Reproducing Kernel Hilbert Spaces*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2016. DOI: 10.1017/CBO9781316219232.

[90] Karime Pereida, Mohamed K Helwa, and Angela P Schoellig. "Data-efficient multirobot, multitask transfer learning for trajectory tracking". In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 1260–1267.

[91] Luka Petrović, Šandor Iles, and Jadranko Matuško. "Self-learning model predictive control based on the sequence of controllable sets". In: *2017 19th International Conference on Electrical Drives and Power Electronics (EDPE)*. IEEE. 2017, pp. 359–364.

[92] Noelia Pizzi et al. "Probabilistic ultimate bounds and invariant sets in nonlinear systems". In: *Automatica* 133 (2021), p. 109853.

[93] André Platzer and Edmund M Clarke. "The image computation problem in hybrid systems model checking". In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 2007, pp. 473–486.

[94] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[95] Saša V. Rakovic and Miroslav Baric. "Parameterized Robust Control Invariant Sets for Linear Systems: Theoretical Advances and Computational Remarks". In: *IEEE Transactions on Automatic Control* 55.7 (2010), pp. 1599–1614. DOI: 10.1109/TAC.2010.2042341.

[96] Stefan Ratschan and Zhikun She. "Safety verification of hybrid systems by constraint propagation-based abstraction refinement". In: *ACM Transactions on Embedded Computing Systems (TECS)* 6.1 (2007), p. 8.

[97] Alexander Robey, Hamed Hassani, and George J Pappas. "Model-Based Robust Deep Learning". In: *arXiv preprint arXiv:2005.10247* (2020).

[98] U. Rosolia and F. Borrelli. "Learning Model Predictive Control for Iterative Tasks: A Computationally Efficient Approach for Linear System". In: *20th IFAC World Congress*. IFAC. 2017.

[99] Ugo Rosolia and Francesco Borrelli. "Learning Model Predictive Control for Iterative Tasks". In: *CoRR* abs/1609.01387 (2016). arXiv: 1609.01387. URL: http://arxiv.org/abs/1609.01387.

[100] Ugo Rosolia and Francesco Borrelli. "Learning model predictive control for iterative tasks. a data-driven control framework". In: *IEEE Transactions on Automatic Control* 63.7 (2017), pp. 1883–1896.

[101] Ugo Rosolia, Ashwin Carvalho, and Francesco Borrelli. "Autonomous racing using learning model predictive control". In: *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 5115–5120.

[102] Ugo Rosolia, Andrew Singletary, and Aaron D. Ames. *Unified Multi-Rate Control: from Low Level Actuation to High Level Planning*. 2020. arXiv: 2012.06558 [eess.SY].

[103] Ugo Rosolia, Xiaojing Zhang, and Francesco Borrelli. "Simple Policy Evaluation for Data-Rich Iterative Tasks". In: *2019 American Control Conference (ACC)*. 2019, pp. 2855–2860. DOI: 10.23919/ACC.2019.8814765.

[104] Matthias Rungger and Paulo Tabuada. "Computing Robust Controlled Invariant Sets of Linear Systems". In: *IEEE Transactions on Automatic Control* 62.7 (2017), pp. 3665–3670. DOI: 10.1109/TAC.2017.2672859.

[105] Matthias Rungger and Paulo Tabuada. "Computing Robust Controlled Invariant Sets of Linear Systems". In: *IEEE Transactions on Automatic Control* 62.7 (July 2017), pp. 3665–3670. ISSN: 1558-2523. DOI: 10.1109/tac.2017.2672859. URL: http://dx.doi.org/10.1109/TAC.2017.2672859.

[106] Karsten Scheibler, Stefan Kupferschmid, and Bernd Becker. "Recent Improvements in the SMT Solver iSAT." In: ().

[107] Stefan Schupp et al. "Current challenges in the verification of hybrid systems". In: *International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems*. Springer. 2015, pp. 8–24.

[108] Dong Shen and Xuefang Li. "A survey on iterative learning control with randomly varying trial lengths: Model, synthesis, and convergence analysis". In: *Annual Reviews in Control* 48 (2019), pp. 89–102.

[109] Yasser Shoukry et al. "Closed-form controlled invariant sets for pedestrian avoidance". In: *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 1622–1628.

[110] M.W. Spong and M Vidyasagar. *Robot Dynamics And Control*. Jan. 1989.

[111] Niranjan Srinivas et al. "Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting". In: *IEEE Transactions on Information Theory* 58.5 (2012), pp. 3250–3265. DOI: 10.1109/TIT.2011.2182033.

[112] Ingo Steinwart. "On the influence of the kernel on the consistency of support vector machines". In: *Journal of machine learning research* 2.Nov (2001), pp. 67–93.

[113] Martin Stolle and Christopher Atkeson. "Finding and transferring policies using stored behaviors". In: *Autonomous Robots* 29.2 (2010), pp. 169–200.

[114] Martin Stolle and Christopher Atkeson. "Finding and transferring policies using stored behaviors". In: *Autonomous Robots* 29.2 (2010), pp. 169–200.

[115] Walid Taha et al. "Acumen: An Open-source Testbed for Cyber-Physical Systems Research". In: Oct. 2015.

[116] Yuval Tassa, Tom Erez, and William D Smart. "Receding horizon differential dynamic programming". In: *Advances in neural information processing systems*. 2008, pp. 1465–1472.

[117] Andrea Tirinzoni et al. *Importance Weighted Transfer of Samples in Reinforcement Learning*. May 2018.

[118] Ufuk Topcu, Andrew Packard, and Peter Seiler. "Local stability analysis using simulations and sum-of-squares programming". In: *Automatica* 44.10 (2008), pp. 2669–2675.

[119] L. Torrey and J. Shavlik. "Transfer learning". In: *Handbook of Research on Machine Learning Applications* (Jan. 2009). DOI: 10.4018/978-1-60566-766-9.ch011.

[120] Spyros G Tzafestas. *Introduction to mobile robot control*. Elsevier, 2013.

[121] Charlott Vallon and Francesco Borrelli. "Data-driven hierarchical predictive learning in unknown environments". In: *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2020, pp. 104–109.

[122] Charlott Vallon and Francesco Borrelli. "Task Decomposition for Iterative Learning Model Predictive Control". In: *[Submitted]* (2019). URL: http://arxiv.org/abs/1903.07003.

[123] Charlott Vallon and Francesco Borrelli. "Task Decomposition for Iterative Learning Model Predictive Control". In: *2020 American Control Conference (ACC)*. IEEE. 2020.

[124] Charlott Vallon and Francesco Borrelli. "Task Decomposition for MPC: A Computationally Efficient Approach for Linear Time-Varying Systems". In: *IFACPapersOnLine* (2020).

[125] Giorgio Valmorbida and James Anderson. "Region of attraction estimation using invariant sets and rational Lyapunov functions". In: *Automatica* 75 (2017), pp. 37–45.

[126] Jur Van Den Berg et al. "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 2074–2081.

[127] V. T. Vasudevan, A. Sethy, and A. R. Ghias. "Towards Better Confidence Estimation for Neural Models". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 7335–7339. DOI: 10.1109/ICASSP.2019.8683359.

[128] N Wada, H Tomosugi, and M Saeki. "Model predictive tracking control for a linear system under time-varying input constraints". In: *International Journal of Robust and Nonlinear Control* 23.9 (2013), pp. 945–964.

[129] Ying Chung Wang, Chiang Ju Chien, and Chi Nan Chuang. "Backstepping adaptive iterative learning control for robotic systems". In: *Applied Mechanics and Materials*. Vol. 284. Trans Tech Publ. 2013, pp. 1759–1763.

[130] Youqing Wang, Furong Gao, and Francis J Doyle III. "Survey on iterative learning control, repetitive control, and run-to-run control". In: *Journal of Process Control* 19.10 (2009), pp. 1589–1600.

[131] Zheming Wang and Raphaël M Jungers. "Data-driven computation of invariant sets of discrete time-invariant black-box systems". In: *arXiv preprint arXiv:1907.12075* (2019).

[132] Zheming Wang and Raphaël M. Jungers. *Data-driven computation of invariant sets of discrete time-invariant black-box systems*. 2020. arXiv: `1907.12075 [eess.SY]`.

[133] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. "A survey of transfer learning". In: *Journal of Big data* 3.1 (2016), p. 9.

[134] Guofan Wu and Koushil Sreenath. "Safety-critical geometric control for systems on manifolds subject to time-varying constraints". In: *IEEE Transactions on Automatic Control (TAC), in review* (2016).

[135] Xuning Yang et al. "Online adaptive teleoperation via motion primitives for mobile robots". In: *Autonomous Robots* 43.6 (Aug. 2019), pp. 1357–1373.

[136] Dailin Zhang, Zining Wang, and Tomizuka Masayoshi. "Neural-Network-Based Iterative Learning Control for Multiple Tasks". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[137] Weiming Zhi et al. *OCTNet: Trajectory Generation in New Environments from Past Experiences*. 2019. arXiv: `1909.11337 [cs.RO]`.

[138] Fuzhen Zhuang et al. "A comprehensive survey on transfer learning". In: *arXiv preprint arXiv:1911.02685* (2019).

[139] Philip Zucker. *philzook58/FlapPyBird-MPC*. URL: `https://github.com/philzook58/FlapPyBird-MPC`.

[140] Jurgen van Zundert, Joost Bolder, and Tom Oomen. "Optimality and flexibility in iterative learning control for varying tasks". In: *Automatica* 67 (2016), pp. 295–302.

Appendix A

Proofs

A.1 Proofs from Chapter 2

Proof of Thm. 1

Proof. At every state x_k, the ILMPC policy (2.10) searches for a sequence of inputs such that, when applied to the system (4.1), the resulting state x_{k+N} is in $\mathcal{SS}^0_{[l_1 \to l_M]}$ or the target set $\mathcal{R}_{l_M}$.

Since all states in $\mathcal{SS}^0_{[l_1 \to l_M]}$ are either stored as part of feasible trajectories to $\mathcal{R}_{l_M}$ or are directly in $\mathcal{R}_{l_M}$, such a sequence of inputs can always be found, and (2.9) always has a solution:

$$\forall\ x_k \in\ \mathcal{SS}^0_{[l_1 \to l_M]},\ \exists\ [u_k, u_{k+1}, ..., u_{k+N-1}] \in\ \mathcal{U}\ :\ x_{k+N} \in \mathcal{SS}^0_{[l_1 \to l_M]} \subseteq \mathcal{X}.$$

As the terminal constraint set in 2.9 is itself an invariant set, recursive feasibility follows from standard MPC arguments [19]. It follows that the policy $\pi^{ILMPC}_{[l_1 \to l_M]}$ produces feasible trajectories for $\mathcal{T}^2$. □

Proof of Thm. 2

Proof. Define the vectors

$$\boldsymbol{x}^1 = \mathcal{SS}^1_{[1 \to M]} \subseteq \mathcal{SS}^J_{[1 \to M]} \tag{A.1a}$$

$$\boldsymbol{u}^1 = \pi^0(\boldsymbol{x}^1) = \mathcal{SU}^1_{[1 \to M]} \subseteq\ \mathcal{SU}^J_{[1 \to M]}, \tag{A.1b}$$

to be the stored state and input trajectory associated with the implemented policy $\pi^0(\cdot)$. Since $\pi^0(\cdot)$ is also feasible for $\mathcal{T}^2$, when Algorithm 1 is applied, the entire task execution can be stored as a successful execution for $\mathcal{T}^2$ without adapting the policy. It follows that $\mathcal{SS}^1_{[1 \to M]} \subseteq\ \mathcal{SS}^0_{[l_1 \to l_M]}$ and $\mathcal{SU}^1_{[1 \to M]} \subseteq \mathcal{SU}^0_{[l_1 \to l_M]}$, and the returned sample safe sets for $\mathcal{T}^2$ are non-empty.

At the initial state x_0, the ILMPC policy (2.10) optimizes the chosen input so as to minimize the remaining cost-to-go. Consider an MPC planning horizon of $N = 1$ (though this extends directly for any $N \geq 1$). Trivially,

$$\begin{aligned} \min_u \ & h(x_0, u) + V^j(x_p^j) \quad \leq \quad h(x_k, \pi^0(x_0)) + V^j(x_p^j) \\ \text{s.t. } & u \in \mathcal{U} \qquad\qquad\qquad\quad \text{s.t. } f(x_0, \pi^0(x_0)) = x_p^j. \\ & f(x_0, u) = x_p^j. \end{aligned}$$

It follows that the cost incurred by a Task 2 execution with $\pi^{ILMPC}_{[l_1 \to l_M]}$ is no higher than an execution with $\pi^{ILMPC}_{\pi^0}$. □

A.2 Proofs from Chapter 3

Proof of Thm. 3

Proof. We will use induction to prove the feasibility. First, we show that the ILMPC (3.7-3.8) is feasible at time step $k = 0$ of the j-th execution of $\mathcal{T}^2$. By assumption, $\mathcal{CK}^0_{[l_1 \to l_M]}$ is not empty. From (3.5) we have that $\mathcal{CK}^0_{[l_1 \to l_M]} \subseteq \mathcal{CK}^{j-1}_{[l_1 \to l_M]}$ $\forall j \geq 1$, and consequently $\mathcal{CK}^{j-1}_{[l_1 \to l_M]}$ is not empty. Therefore, $x_0 \in \mathcal{CK}^0_{[l_1 \to l_M]} \subseteq \mathcal{CK}^{j-1}_{[l_1 \to l_M]}$, and there exist $I^\star$, $K^\star$, and multipliers $\lambda_p^\star$ such that

$$x_0 = \sum_{p=1}^{|\mathcal{CK}^{j-1}_{I^\star, K^\star}|} \lambda_p^\star x_p, \ x_p \in \mathcal{CK}^{j-1}_{I^\star, K^\star}.$$

We define

$$\bar{u} = \lambda_p^\star u_p \in \mathcal{U}_{I^\star} \tag{A.2}$$

where u_p is the input associated with the state $x_p \in \mathcal{CK}^{j-1}_{I^\star, K^\star}$ in a previous task execution of $\mathcal{T}^1$. Now, note that we have

$$\begin{aligned} \bar{x} = A_{I^\star} + B_{I^\star}\bar{u} = & \sum_{p=1}^{|\mathcal{CK}^{j-1}_{I^\star, K^\star - 1}|} \lambda_p^\star x_p \\ x_p \in & \begin{cases} \mathcal{CK}^{j-1}_{I^\star, K^\star - 1} & K^\star \geq 1 \\ \mathcal{CK}^{j-1}_{I^\star + 1, K^{\star\star}} & K^\star = 0, \end{cases} \end{aligned} \tag{A.3}$$

for some $K^{\star\star}$. The second case ($K^\star = 0$) follows directly from Alg. 2. This procedure (A.2-A.3) can be repeated N times in order to find a feasible input sequence for the initial state x_0 satisfying (3.7). Therefore there exists a feasible solution to the ILMPC (3.7 - 3.8) at time step $k = 0$ of the j-th execution of $\mathcal{T}^2$.

Next, we show that the policy is recursively feasible. Assume that at time step k of the j-th iteration the ILMPC (3.7 - 3.8) is feasible, and let $\boldsymbol{x}^{\star,j}_{k:k+N|k}$ and $\boldsymbol{u}^{\star,j}_{k:k+N|k}$ be the optimal

trajectory and input sequence according to (3.7). From (3.8), the realized state and input at time k of the j-th iteration are given by

$$x_k^j = x_{k|k}^{\star,j},\ u_k^j = u_{k|k}^{\star,j},$$

and the terminal constraint in (3.7) enforces $x_{k+N|k}^{\star,j} \in \mathcal{CK}_{I',K'}^{j-1}$ for some I', K', where $\mathcal{CK}_{I',K'}^{j-1}$ contains states from the previous $j-1$ trajectories. As in (A.2-A.3), define an input and corresponding state

$$\begin{aligned} u' &= \lambda_p^\star u_p \in \mathcal{U}_{I'} \\ x' &= A_{I'} x_{k+N|k}^{\star,j} + B_{I'} u' \in \mathcal{CK}_{I',K'-1}^{j-1}. \end{aligned}$$

We therefore have

$$x_{k+1}^j = x_{k+1|k}^{\star,j}.$$

It follows that at time step $k+1$ of the j-th $\mathcal{T}^2$ execution, the input sequence and related feasible state trajectory

$$\begin{aligned} &[u_{k+1|k}^{\star,j},\ u_{k+2|k}^{\star,j},\ ..., u_{k+N-1|k}^{\star,j}, u'] \\ &[x_{k+1|k}^{\star,j},\ x_{k+2|k}^{\star,j},\ ..., x_{k+N-1|k}^{\star,j}, x'] \end{aligned} \tag{A.4}$$

satisfy input and state constraints in (3.7). Therefore, (A.4) is a feasible solution for (3.7) at time step $k+1$.

We have shown that at the j-th iteration of $\mathcal{T}^2$, $\forall j \geq 1$, the ILMPC is feasible at time step $k=0$ and that if the ILMPC is feasible at time step k, it must also be feasible at time step $k+1$. We can conclude by induction that (3.7 - 3.8) is feasible $\forall j \geq 1$ and $k \in \mathbb{Z}_{0+}$ when initialized with sets output by Alg. 2. □

A.3 Proofs from Chapter 4

Proof of Lem. 1

Proof. Using the $\mathcal{S}$-procedure, we have that the containment $\hat{\mathcal{E}} \subseteq \mathcal{E}$ is equivalent to the existence of $\lambda > 0$ such that

$$\begin{aligned} &\lambda \begin{bmatrix} \hat{P}^{-1} & -\hat{P}^{-1}\hat{c} \\ -\hat{c}^\top \hat{P}^{-1} & \hat{c}^\top \hat{P}^{-1}\hat{c} - 1 \end{bmatrix} - \begin{bmatrix} P^{-1} & -P^{-1}c \\ -c^\top P^{-1} & c^\top P^{-1}c - 1 \end{bmatrix} \succeq 0 \\ &\begin{bmatrix} \lambda\hat{P}^{-1} - P^{-1} & P^{-1}c - \lambda\hat{P}^{-1}\hat{c} \\ c^\top P^{-1} - \lambda\hat{c}^\top \hat{P}^{-1} & \lambda(\hat{c}^\top \hat{P}^{-1}\hat{c} - 1) - (c^\top P^{-1}c - 1) \end{bmatrix} \succeq 0. \end{aligned}$$

Assume that $\lambda\hat{P}^{-1} - P^{-1} \succ 0$. Then using the Schur complement, the above expression is equivalent to

$$\begin{aligned} &\lambda(\hat{c}^\top \hat{P}^{-1}\hat{c} - 1) - (c^\top P^{-1}c - 1) \\ &- (P^{-1}c - \lambda\hat{P}^{-1}\hat{c})^\top (\lambda\hat{P}^{-1} - P^{-1})^{-1}(P^{-1}c - \lambda\hat{P}^{-1}\hat{c}) \geq 0, \end{aligned}$$

which we can rewrite using the matrix inversion lemma as

$$(\hat{c}-c)^\top (P-\frac{1}{\lambda}\hat{P})^{-1}(\hat{c}-c)+(\lambda-1)\leq 0. \tag{A.5}$$

The first term in (A.5) can be upper-bounded using the error bound on the predicted ellipsoid center and the lower bound on the size of the shape matrix:

$$\begin{aligned}(\hat{c}-c)^\top (P-\frac{1}{\lambda}\hat{P})^{-1}(\hat{c}-c) &\leq (\hat{c}-c)^\top (\Gamma-\frac{1}{\lambda}\hat{P})^{-1}(\hat{c}-c)\\ &\leq \|(\Gamma-\frac{1}{\lambda}\hat{P})^{-1}\|_2\|\hat{c}-c\|_2^2\\ &= \frac{\lambda\epsilon^2}{\lambda_{\min}(\lambda\Gamma-\hat{P})},\end{aligned}$$

where $\lambda_{\min}(A)$ denotes the minimum eigenvalue of A. Therefore, if there exists $\hat{P}\succ 0$ and $\lambda>0$ such that

$$\begin{aligned}\lambda\Gamma-\hat{P}&\succ 0\\ \frac{\lambda\epsilon^2}{\lambda_{\min}(\lambda\Gamma-\hat{P})}+(\lambda-1)&\leq 0,\end{aligned}$$

it holds that (A.5) is satisfied and we can follow the previous equivalences to reach the conclusion that $\hat{\mathcal{E}}\subseteq\mathcal{E}$. □

Proof of Thm. 4

Proof. First, note we can write our joint state-parameter system using the augmented state $\bar{x}=[x,z]$:

$$\bar{x}_{k+1}=\begin{bmatrix}A & 0\\ 0 & C\end{bmatrix}\bar{x}_k+\begin{bmatrix}B\\ 0\end{bmatrix}u_k \tag{A.6}$$

$$=\bar{A}\bar{x}_k+\bar{B}u_k, \tag{A.7}$$

with joint constraints

$$\begin{bmatrix}H_x & 0\\ 0 & H_z\end{bmatrix}\bar{x}_k\leq\begin{bmatrix}h_x\\ h_z\end{bmatrix} \tag{A.8}$$

$$H_u u_k\leq h_u. \tag{A.9}$$

Since $\bar{x}_0^{n+1}\in Co(\mathcal{E}_0)$, it can be written as a weighted sum of the points $\bar{x}_i$ in $\mathcal{E}_0$, which all satisfy the joint system and environment constraints and are T-step controllable to the task goal:

$$\bar{x}_0^{n+1}=\sum_{i=1}^{|\mathcal{E}_0|}\lambda_i\bar{x}_i,\quad \lambda_i\geq 0,\quad \sum_i\lambda_i=1. \tag{A.10}$$

Each of these stored points $\bar{x}_i$ has an associated input u_i (satisfying $H_u u_i \leq h_u$) that was applied during a previous task trajectory. Applying the new input $u_0^{n+1} = \sum_{i=1}^{|\mathcal{E}_0|} \lambda_i u_i$ at the current state $\bar{x}_0^{n+1}$ results in a new state

$$\bar{x}_1^{n+1} = \bar{A}\bar{x}_0^{n+1} + \bar{B}\sum_{i=1}^{|\mathcal{E}_0|} \lambda_i u_i \tag{A.11}$$

$$= \bar{A}\sum_{i=1}^{|\mathcal{D}^n|} \lambda_i \bar{x}_i + \bar{B}\sum_{i=1}^{|\mathcal{E}_0|} \lambda_i u_i \tag{A.12}$$

$$= \sum_{i=1}^{|\mathcal{E}_0|} \lambda_i (\bar{A}\bar{x}_i + \bar{B}u_i) \tag{A.13}$$

$$\in Co(\mathcal{E}_1), \tag{A.14}$$

which is also in the convex hull of stored data from time step $t = 1$: $\bar{x}_1^{n+1} \in Co(\mathcal{E}_1)$. It is easily verified that both $\bar{x}_1^{n+1}$ and u_1^{n+1} satisfy all system and environmental state and input constraints.

Continuing to apply the convex combination of previously recorded inputs will result in a feasible closed-loop trajectory from $\bar{x}_0^{n+1}$ that interpolates at each time step between the stored trajectories in $\mathcal{D}^n$. Since all stored trajectories end in the task goal set $\mathcal{X}_T$ per Asm. 6, the resulting trajectory in $\mathcal{T}^{n+1}$ also ends in the task goal set.

Thus, if $\bar{x}_0^{n+1} \in Co(\mathcal{E}_0)$, the state is T-step controllable to the task goal set:

$$Co(\mathcal{E}_0) \subseteq \mathcal{K}_T.$$

It follows that for systems satisfying Asms. 6-9, the ellipsoids $\tilde{\mathcal{X}}_k(z_k)$ formed as described in Sec. 4.4, which are subsets of $Co(\mathcal{E}_k)$, are all $(T-k)$-step controllable sets to the task goal set. Specifically, this means that the center of the ellipse, $\phi_k(z_k)$, is $(T-k)$-step controllable to the task goal set.

The stored ellipse centers $\phi_k(z_k)$ make up the GP training data in Sec. 4.5. Each training data point represents a sample from a function g_k which maps an augmented state $y = [x, z]$ to the center $c_k(z_k)$ of the ellipse $\mathcal{X}_{k+N}(z_{k+N})$. From Lem. 2, for a chosen $\beta_B(\delta)$

$$\mathbb{P}[||g_k(y_k) - \mu(y_k)||_2 \leq \hat{\sigma}(y_k)\sqrt{\beta_B}] \geq (1-\delta),$$

where $g_k(y_k)$ is the center and shape matrix of the ellipse $\mathcal{X}_{k+N}(z_{k+N})$ and $\mu(y_k)$ is the GP mean function evaluated at the new augmented state. Thus with probability $(1-\delta)$, the evaluated mean is within a bounded distance of a controllable state $g_k(y_k)$.

It follows from Lem. 1 that the ellipsoid $\text{Ell}(\hat{c}_k, \hat{P}_k)$, which is centered at $\mu(y_k)$ and constructed according to Sec. 4.6, is thus entirely contained within $\tilde{\mathcal{X}}_{k+N}(z_{k+N}) \subseteq \mathcal{K}_{T-(k+N)}$ with the chosen probability $(1-\delta)$. This completes the proof.

□

Proof of Thm. 5

Proof. We use induction to prove that for all $0 \leq k \leq K$, the MPC controller (5.25) finds an input u_k such that the resulting closed-loop trajectory satisfies system and environment constraints.

Initialization: At time $k = 0$ of the new task $\mathcal{T}^{n+1}$, the augmented state is in the convex hull of previous data, $\bar{x}_0^{n+1} = [x_0^{n+1}, z_0^{n+1}] \in Co(\mathcal{E}_0)$. We consider two cases:

Case 1: the MPC (5.25) using the GP-constructed terminal set (4.18) is feasible at time $k = 0$. In this case, the optimal input sequence calculated by the MPC is a feasible input sequence that satisfies system and environment constraints, and the first input can be safely applied.

Case 2: the MPC (5.25) is not feasible at time $k = 0$, and the safety control must be applied. Since $\bar{x}_0^{n+1} \in Co(\mathcal{E}_0)$, there exist convex hull multipliers $\lambda_i \geq 0,\ \sum_i \lambda_i = 1$ such that

$$\bar{x}_0^{n+1} = \sum_{i=1}^{|\mathcal{E}_0|} \lambda_i x_0^i.$$

Then, one feasible input sequence satisfying this requirement is the interpolation of stored input trajectories,

$$u_j^{n+1} = \sum_{i=1}^{|\mathcal{E}_0|} \lambda_i u_j^i,\ j \in \{0, N-1\}.$$

Thus we have shown that if $\bar{x}_0^{n+1} = [x_0^{n+1}, z_0^{n+1}] \in Co(\mathcal{E}_0)$, there exists a feasible N-step input sequence satisfying all state and task constraints.

Recursion: Next, we show that if a feasible N-step input sequence was found for a state x_k^{n+1} at time k,

$$u_{k:k+N-1}^{n+1} = [\bar{u}_k^{n+1}, \bar{u}_{k+1}^{n+1}, \ldots, \bar{u}_{k+N-1}^{n+1}],$$

then Alg. 3 will also find an N-step input sequence at time $k+1$.

If at time $k+1$, the MPC with GP-constructed terminal set $\mathcal{X}_{k+1+N}(z_{k+1+N})$ is feasible, the calculated MPC input sequence satisfies all state and task constraints, and can be safely applied. However, if the MPC is not feasible, the safety control is applied. Here we show that the safety control will always have a feasible solution. Again, we consider two cases:

Case 1: The MPC with GP-constructed terminal set was feasible at time k, and thus applying the input sequence $u_{k:k+N-1}^{n+1}$ results in a state $\bar{x}_{k+N}^{n+1} \in \mathcal{X}_{k+N}(z_{k+N}^{n+1})$. By Thm. 4, this set $\mathcal{X}_{k+N}(z_{k+N}^{n+1})$ is with probability $(1-\delta)$ a subset of the controllable set, and, particularly, a subset of $\mathcal{E}_{k+N}$. Therefore, there exist convex hull multipliers $\lambda_i \geq 0,\ \sum_i \lambda_i = 1$ such that

$$\bar{x}_k^{n+1} = \sum_{i=1}^{|\mathcal{E}_k|} \lambda_i x_k^i.$$

Then, one feasible input sequence satisfying the safety control requirement is the interpolation of stored input trajectories,

$$u_{k+1}^{n+1} = [u_{k+1:k+N-1}^{n+1}, \ \sum_{i=1}^{|\mathcal{E}_k|} \lambda_i u_{k+N}^i].$$

Since at least one solution exists, the safety control will find a feasible input trajectory.

Case 2: The MPC with GP-constructed terminal set was infeasible at time k, but the safety control was feasible. Therefore, applying the input sequence $u_{k:k+N-1}^{n+1}$ results in a state $\bar{x}_{k+N}^{n+1} \in Co(\mathcal{E}_{k+N})$. Thus, there again exist convex hull multipliers $\lambda_i \geq 0, \ \sum_i \lambda_i = 1$ such that

$$\bar{x}_k^{n+1} = \sum_{i=1}^{|\mathcal{E}_k|} \lambda_i x_k^i.$$

Then, one feasible input sequence satisfying the safety control requirement is the interpolation of stored input trajectories,

$$u_{k+1}^{n+1} = [u_{k+1:k+N-1}^{n+1}, \ \sum_{i=1}^{|\mathcal{E}_k|} \lambda_i u_k^i].$$

Since at least one solution exists, the safety control will find a feasible input trajectory.

Thus we have shown that if a feasible N-step input sequence was found for a state x_k^{n+1} at time t, then Alg. 3 will with probability at least $(1-\delta)$ also find an N-step input sequence at time $k+1$.

We have shown that *i)* Alg. 3 finds a feasible input at time step $k=0$, and *ii)* if the loop finds a feasible input at time step k, it must also find a feasible input at time $k+1$. We conclude by induction that the closed-loop control described by Alg. 3 finds a feasible input $u_k \forall k \in \{0, T\}$ in the new task $\mathcal{T}^{n+1}$. This results in a feasible execution of $\mathcal{T}^{n+1}$.

□

A.4 Proofs from Chapter 5

Proof of Thm. 6

Proof. We use induction to prove that for all $k \geq 0$, the iteration loop (Lines 10-23) in Alg. 4 finds an input u_k such that the resulting closed-loop trajectory satisfies system and environment constraints.

At time $k=0$ of the new task $\mathcal{T}^{n+1}$, the target set list can contain at most one non-empty set, $\mathcal{X}_T$ (5.22). If $\mathcal{X}_T$ is non-empty, and the resulting (5.25) is feasible, then there exists an input sequence $[u_{0|0}, \ldots, u_{N-1|0}]$ calculated by (5.25) satisfying all state and input constraints

(5.3), with $x_{N|0} \in \mathcal{X}_N$. However, if $\mathcal{X}_N$ is empty or (5.25) is infeasible, we instead apply the safety control law $u_0 = \pi_e(x_0, \boldsymbol{\Theta})$. By assumption, $x_0 \in \mathcal{X}_E$, so this input is feasible. Thus we have shown that the iteration loop in Alg. 4 is feasible for $k = 0$.

Next, we show that the iteration loop of Alg. 4 is recursively feasible. Assume that at time $k > 0$, the low-level policy (5.25-5.27) is feasible with horizon N_k^{MPC}, and let $\boldsymbol{x}^\star_{k:k+N_k^{MPC}|k}$ and $\boldsymbol{u}^\star_{k:k+N_k^{MPC}-1|k}$ be the optimal state trajectory and input sequence according to (5.25), such that

$$u_k = u^\star_{k|k} \tag{A.15}$$

$$x^\star_{k+N_k^{MPC}|k} \in \mathcal{X}_{k+N_k^{MPC}}. \tag{A.16}$$

If at time $k+1$ a non-empty target set $\mathcal{X}_{k+N+1}$ is constructed according to (5.22) such that (5.25) is feasible, then there exists a feasible input sequence $[u_{k+1|k+1}, \dots, u_{k+N|k+1}]$ satisfying all state and input constraints such that $x_{k+N+1} \in \mathcal{X}_{k+N+1}$.

If at time $k+1$ the target set is *empty*, or (5.25) is infeasible, we must consider two cases separately:

Case 1: The MPC horizon at time step k is $N_k^{MPC} > 1$. In the absence of model uncertainty, when the closed-loop input u_k (A.15) is applied, the system (5.1) evolves such that

$$x_{k+1} = x^\star_{k+1|k}. \tag{A.17}$$

According to Alg. 4, when the empty target set $\mathcal{X}_{k+N+1}$ is added to the target set list (5.24), the MPC horizon is shortened and the most recent non-empty target set is used again. Since $N_k^{MPC} > 1$, we are guaranteed at least one non-empty target set in (5.24) that may used as a feasible terminal constraint in the low-level controller (5.25). At time step $k+1$, the shifted input sequence $\boldsymbol{u}^\star_{k+1:k+N_k^{MPC}-1|k}$ will be optimal for this shifted horizon optimal control problem (with a corresponding state trajectory $\boldsymbol{x}^\star_{k+1:k+N_k^{MPC}|k}$). At time step $k+1$, Alg. 4 applies the second input calculated at the previous time step: $u_{k+1} = u^\star_{k+1|k}$.

Case 2: $N_k^{MPC} = 1$. In this scenario, the target set list (5.24) at time step $k+1$ is empty, resulting in $N_{k+1}^{MPC} = 0$ (5.26). However, combining the fact that $N_k^{MPC} = 1$ with (A.16) and (A.17), we note that

$$x_{k+1} \in \mathcal{X}_{k+1} \subseteq \mathcal{X}_E,$$

by construction of the target set (5.22). This implies that at time step $k+1$, system (5.1) is necessarily in the safe set (5.20), and so application of the safety controller (5.21) will result in a feasible input, $u_{k+1} = \pi_e(x_{k+1}, \boldsymbol{\Theta})$.

We have shown that *i)* the online iteration loop (Lines 10-23) in Alg. 4 finds a feasible input at time step $k = 0$, and *ii)* if the loop finds a feasible input at time step k, it must also find a feasible input at time $k+1$. We conclude by induction that Alg. 4 finds a feasible input u_k $\forall k \in \mathbb{Z}_{0+}$ in the new task $\mathcal{T}^{n+1}$. This results in a feasible execution of $\mathcal{T}^{n+1}$. □